State of the Art and Potentialities of Graph-level Learning

ZHENYU YANG, School of Computing, Macquarie University, Sydney, Australia
GE ZHANG, School of Computing, Donghua University, Shanghai, China and School of Computing, Macquarie University, Sydney, Australia
JIA WU, School of Computing, Macquarie University, Sydney, Australia
JIAN YANG, School of Computing, Macquarie University, Sydney, Australia
QUAN Z. SHENG, School of Computing, Macquarie University, Sydney, Australia
SHAN XUE, School of Computing, Macquarie University, Sydney, Australia
CHUAN ZHOU, Chinese Academy of Sciences, Beijing, China
CHARU AGGARWAL, Office 4S-A20, IBM T. J. Watson Research Center, Hawthorne, United States
HAO PENG, Beihang University, Beijing, China
WENBIN HU, Wuhan University, Wuhan, China
EDWIN HANCOCK, University of York, York, United Kingdom of Great Britain and Northern Ireland
PIETRO LIÒ, University of Cambridge, Cambridge, United Kingdom of Great Britain and Northern Ireland

Graphs have a superior ability to represent relational data, such as 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. 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, no comprehensive survey 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,

This work was supported by the Australian Research Council Projects Nos. LP210301259, and DP230100899.
Authors' Contact Information: Zhenyu Yang, School of Computing, Macquarie University, Sydney, New South Wales, Australia; e-mail: zhenyu.yang3@hdr.mq.edu.au; Ge Zhang, School of Computing, Donghua University, Shanghai, China and School of Computing, Macquarie University, Sydney, New South Wales, Australia; e-mail: ge.zhang5@hdr.mq.edu.au; Jia Wu (Corresponding author), School of Computing, Macquarie University, Sydney, New South Wales, Australia; e-mail: jia.wu@mq.edu.au; Jian Yang, School of Computing, Macquarie University, Sydney, New South Wales, Australia; e-mail: jian.yang@mq.edu.au; Quan Z. Sheng, School of Computing, Macquarie University, Sydney, Australia; e-mail: michael.sheng@mq.edu.au; Shan Xue, School of Computing, Macquarie University, Sydney, New South Wales, Australia; e-mail: emma.xue@mq.edu.au; Chuan Zhou, Chinese Academy of Sciences, Beijing, Beijing, China; e-mail: zhouchuan@amss.ac.cn; Charu Aggarwal, Office 4S-A20, IBM T. J. Watson Research Center, Hawthorne, NY, USA; e-mail: charu@us.ibm.com; Hao Peng, Beihang University, Beijing, China; e-mail: penghao@buaa.edu.cn; Wenbin Hu, Wuhan University, Wuhan, Hubei, China; e-mail: hwb@whu.edu.cn; Edwin Hancock, University of York, York, United Kingdom of Great Britain and Northern Ireland; e-mail: edwin.hancock@york.ac.uk; Pietro Liò, University of Cambridge, Cambridge, United Kingdom of Great Britain and Northern Ireland; e-mail: Pietro.Lio@cl.cam.ac.uk.

ACM 0360-0300/2024/10-ART28
https://doi.org/10.1145/3695863

and graph pooling. In addition, the evolution and interaction between methods from these four branches within their developments are examined to provide an in-depth analysis. This is followed by a brief review of the benchmark datasets, evaluation metrics, and common downstream applications. Finally, the survey concludes with an in-depth discussion of 12 current and future directions in this booming field.

CCS Concepts: • **Computing methodologies** → **Neural networks**; • **Mathematics of computing** → **Graph algorithms;**

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. 2024. State of the Art and Potentialities of Graph-level Learning. *ACM Comput. Surv.* 57, 2, Article 28 (October 2024), 40 pages. https://doi.org/10.1145/3695863

1 Introduction

Research into graph-structured data started with the Konigsberg bridge problem [1] in the 18th century, that is: "*How can we design a path among seven bridges in Konigsberg 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 Konigsberg bridge problem is proved unsolvable. Since then, graph-structured data has become an indispensable tool for exploring the world. For instance, in a graph that models a chemical molecule, nodes represent atoms and edges describe chemical bonds between atoms. Researchers can model millions of molecules, in which each presents a graph, to analyze their properties (e.g., anti-cancer activity, toxicity) [2]. Such a case illustrates graph-level learning, which learns the underlying patterns among a set of graphs for classification [3, 4], regression [5, 6], generation [7, 8], and so on.

Mining the underlying rules among a set of graphs is tough, 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?*"[1] An enormous number of studies [9–11] focused on this question and concerned it as a candidate for NP-immediate until a quasi-polynomial-time solution was proposed in 2016 [12]. 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 [13], frequently occurring substructure [14]) 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 are required 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.

[1]Two 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 $\mathcal{G}_2$.

Benefiting from these techniques, graph-level learning has applications and promise in many fields. Wang et al. [15] performed graph regression among a set of molecules to predict their properties for discovering 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 [16]. Likewise, graph classification with brain graphs has the potential to distinguish brain structures with neurological disorders from those of healthy individuals [17, 18].

Despite the demonstrated potential proven by successful applications, graph-level learning still lacks a systematic review. The objective of a comprehensive overview of graph-level learning should encompass its broad scope and provide an in-depth analysis. On the discussion of broad scope, graph-level learning involves various mainstream techniques spanning eras. However, most existing surveys have not given a holistic treatment to the topic. For instance, Wu et al. [19] and Zhang et al. [20] concentrated on exploring nodes in the graph while treating graph-level learning as a subsequent by-product. As a result, most graph-level methods they discussed are node-level learning combined with pooling operations. However, graph-level techniques are not confined to this by leveraging all possible elements in the graph to capture graph properties, such as walk sequences or distances (see Section 4.1), subgraphs (see Sections 4.2 and 6.2), and so on. In addition, several works only investigate a single technique, such as graph kernels [21] and graph pooling [22, 23]. The limited scope of these investigations hinders an in-depth analysis, e.g., evolution and interaction in this field, which are novel and essential to delivering high-level insight into this field. As illustrations, the message-passing scheme, aggregating neighborhood information, has evolved in various techniques ranging from graph kernels to GL-GNNs. Understanding this evolution is crucial for clarifying the developments and commonalities of related techniques. The interaction is equally significant, as researchers have leveraged traditional techniques that perceive global topology to empower GL-GNNs (see Sections 6.2 and 6.3), thus breaking the expressivity limitations imposed by local topology.

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 across various periods of graph-level learning and further provides an in-depth review, including the evolution and interaction of these techniques. 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. Additionally, we talk about the evolution and interaction between various techniques.
- **Abundant resources:** This survey provides readers with abundant resources for 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 Background and Definitions

This section provides the background knowledge and some definitions for understanding this article. Bold lowercase characters (e.g., $\mathbf{x}$) are used to denote vectors. Bold uppercase characters (e.g.,

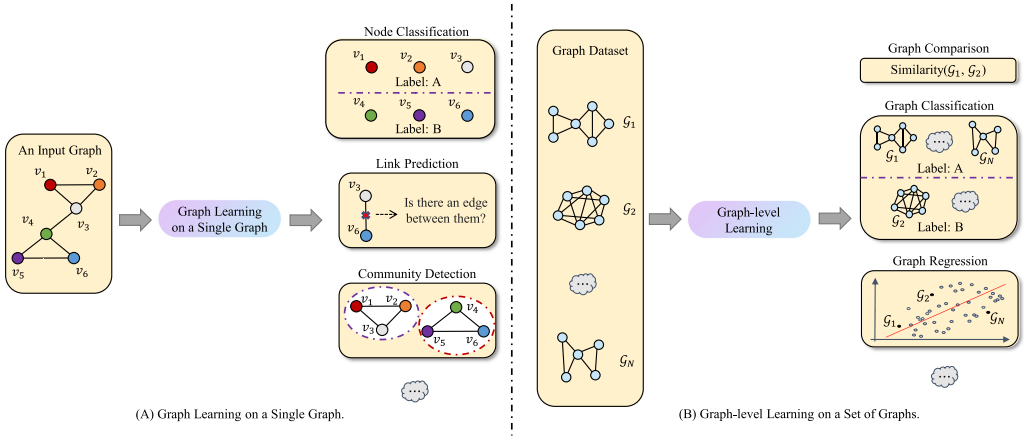


Fig. 1. Illustrative examples of graph learning on a single graph and graph datasets.

$\mathbf{X}$) 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.

Learning on a Single Graph versus Graph-level Learning: Learning on a single graph involves using node attributes and edges from one graph as inputs to understand and make predictions about elements (e.g., nodes and edges) within that graph. Graph-level learning aims to make predictions or extract meaningful insights about the entire graph rather than individual nodes or edges. The differences between learning on a single graph and graph-level learning are shown in Figure 1. Some methods can serve both of them, for instance, most GL-GNNs are adept at learning representations for individual nodes within a graph, while also aggregating these learned node representations to capture properties of the entire graph. In contrast, there are also methods specifically designed for graph-level learning, like graph kernels, which calculate the similarities between pair-wise entire graphs.

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, then 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, then the value of $\mathbf{A}_{u,v}$ refers to the 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 article are undirected attributed graphs.

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

Definition 2.3. **(Subgraph/Substructure)**: Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and $g_m = (\mathcal{V}_{g_m}, \mathcal{E}_{g_m})$ each denote a connected graph, g_m is a subgraph/substructure of $\mathcal{G}$ *iff* there exists an injective function $\phi : \mathcal{V}_{g_m} \to \mathcal{V}$ 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 $\mathbb{G} = \{\mathcal{G}_1, \ldots, \mathcal{G}_N\}$ consisting of N graphs as inputs and returns a function $f(\cdot)$ that maps a graph $\mathcal{G}_i$

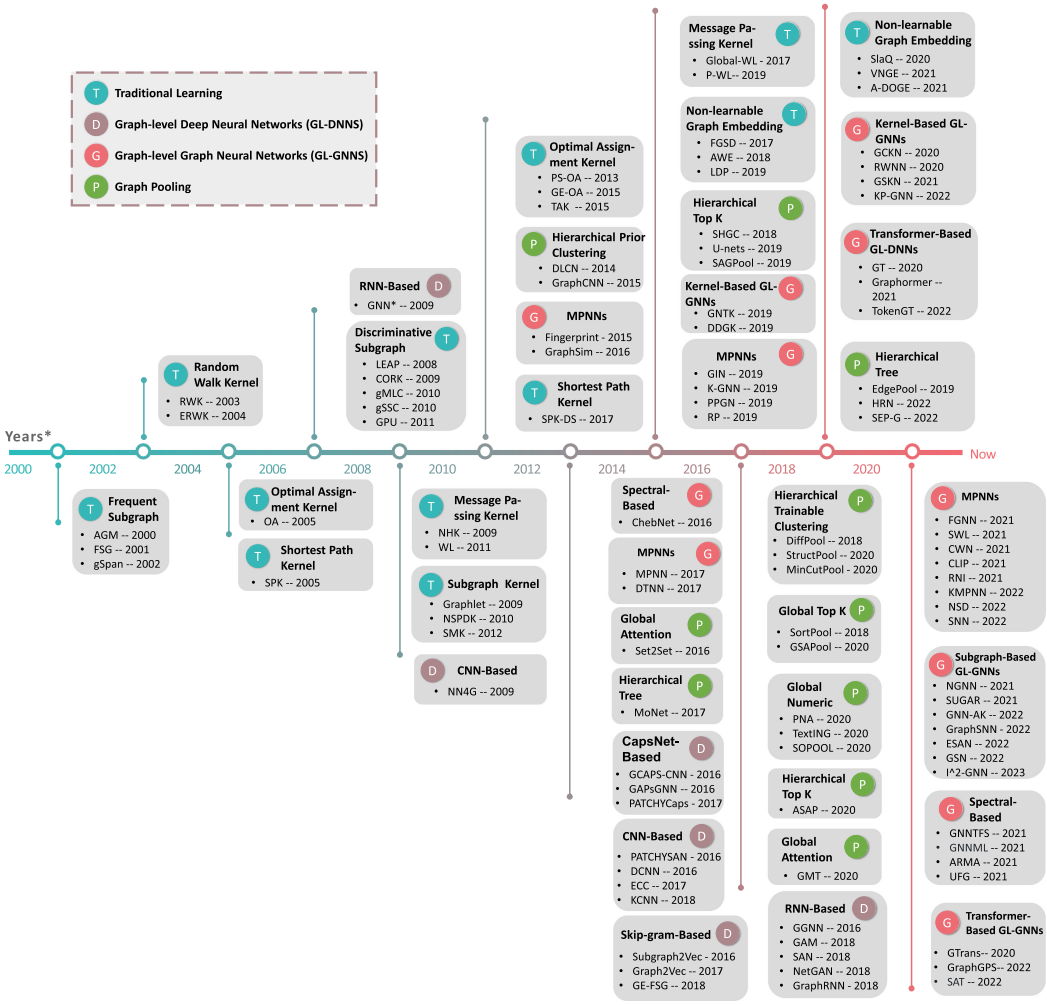


Fig. 2. The timeline of graph-level learning in terms of four mainstream techniques.

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-invariant.[2]

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 timeline of these four mainstream techniques is shown in Figure 2, and a related taxonomy tree can be found in Figure 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

[2]The 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.

features. Given well-designed features (e.g., random walk sequences [13], frequently occurring substructure [14]), 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 graph-structured data due to the computational bottlenecks, yet, it still provides great valuable insights, such as better interpretability and better ability to model irregular structures [24].

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 explorations range from skip-gram to transformer. These 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 them to learn graph-structured data that varies in size and has irregular neighborhood structures.

Graph-Level Graph Neural Networks (GL-GNNs). GL-GNNs are deep neural networks specific for graph-structured data. Most GL-GNNs employ the graph convolutional layers to aggregate and update neighborhood information layer-by-layer for graph-level learning. These methods are simple, easy to understand, and have linear complexity [25], condensing the most fruitful achievements of graph-level learning. Recently, there has been a trend that integrating GL-GNNs with other techniques, particularly traditional learning techniques, to promote 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 [20, 22]. 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 1 and compare the graph kernel architectures in Table 5 within Appendix B.

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 $S^{\mathcal{G}} = \{g_1, \ldots, g_I\}$, 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 [26] that obtains the kernel value $K_{R-conv}(\mathcal{G}, \mathcal{G}')$ of two graphs $\mathcal{G}$ and $\mathcal{G}'$ by:

$$K_{R-conv}(\mathcal{G}, \mathcal{G}') = \sum_{i=1}^{I} \sum_{j=1}^{J} K_{parts}\left(g_i, g_j'\right), \tag{1}$$

where $K_{parts}(g_i, g_j')$ is the kernel function that defines how to measure the similarity between g_i and g_j', and I and J count the size of bag-of-graphs for decomposing $\mathcal{G}$ and $\mathcal{G}'$, respectively. 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)**

Table 1. Summary of Traditional Graph-level Learning Methods

Technique	Architecture	Year	Method	Venue	Inputs	Complexity	Applications
Graph Kernels	Message -passing Kernels	2009	NHK [28]	ICDM	A, L	$O(hnbd)$	Molecule/Protein Classification
		2011	WL [27]	JMLR	A, L	$O(hm)$	Molecule/Protein Classification
		2016	PK [30]	ML	A, X	$O(hm)$	Molecule/Protein/Text/Image/3D Point Clouds Classification
		2017	Global-WL [29]	ICDM	A, X	$O(hm)$	Molecule/Protein/Social Network Classification
		2019	P-WL [31]	ICML	A, X	$O(hmlogm)$	Molecule/Protein/Social Network Classification
	ShortestPath Kernels	2005	SPK [32]	ICDM	A, X	$O(n^4)$	Molecule/Protein Classification
		2017	SPK-DS [33]	EMNLP	A, X	$O(n^4)$	Text Classification
	Random Walk Kernels	2003	RWK [13]	LNAI	A, X	$O(n^6)$	Molecule/Protein Classification
		2004	ERWK [34]	ICML	A, X	$O(n^6)$	Molecule/Protein Classification
		2010	SOMRWK [35]	JMLR	A, X	$O(n^3)$	Molecule/Protein Classification;Protein-protein Interaction Prediction
	Optimal Assignment Kernels	2005	OAK [36]	ICML	A, X	$O(n^4)$	Molecule/Protein Classification
		2013	PS-OAK [37]	NeurIPS	A, X	$O(n^3)$	Image Alignment
		2015	GE-OAK [38]	KDD	A, X	$O(n^3)$	Molecule/Protein Classification
		2015	TAK [39]	SIMBAD	A, X	$O(n^3)$	Molecule/Protein Classification;Shape Recognition
	Subgraph Kernels	2009	Graphlet [40]	AISTATS	A	$O(n^k)$	Molecule/Protein Classification
		2010	NSPDK [41]	ICML	A	$O(bnm)$	Molecule/Protein Classification
		2012	SMK [42]	ICML	A, X	$O(kn^{k+1})$	Molecule/Protein Classification
Subgraph Mining	Frequent Subgraph Mining	2000	AGM [14]	ECML PKDD	A, L	$O(nmn!m!)$	Molecule/Protein Classification
		2001	FSG [43]	ICDM	A, L	$O(nmn!m!)$	Molecule/Protein Classification
		2002	gSpan [44]	ICDM	A, L	$O(nmn!m!)$	Molecule/Protein Classification
	Discrimina -tive Subgraph Mining	2008	LEAP [45]	SIGMOD	A, L	$O(nmn!m!)$	Molecule/Protein Classification
		2009	CORK [46]	SDM	A, L	$O(nmn!m!)$	Molecule/Protein Classification
		2010	gMLC [47]	ICDM	A, L	$O(nmn!m!)$	Molecule/Protein Classification
		2010	gSSC [48]	KDD	A, L	$O(nmn!m!)$	Molecule/Protein Classification
		2011	gPU [49]	ICDM	A, L	$O(nmn!m!)$	Molecule/Protein Classification
		2014	gCGVFL [50]	ICDM	A, L	$O(nmn!m!)$	Image/Social Network Classification
Nonlearnable Graph Embedding		2017	FGSD [51]	NeurIPS	A	$O(n^2)$	Molecule/Protein/Social Network Classification
		2018	AWE [52]	ICML	A, X	$O(n^3)$	Molecule/Protein/Social Network Classification
		2019	LDP [53]	ICLR RLGM	A	$O(m)$	Molecule/Protein/Social Network Classification
		2020	SLAQ [54]	WWW	A	$O(m+n)$	Molecule/Protein/Social Network Classification
		2021	VNGE [55]	WWW	A	$O(n^3)$	Social Network Classification
		2021	A-DOGE [56]	ICDM	A, X	$O(n^2+hm)$	Molecule/Protein/Social Network Classification

*For the Inputs column, **A**, **L**, **X** refer to the adjacency matrix, node label matrix, and node attribute matrix, respectively. To be notified, node attributes could be node labels or even more fine-grained features (e.g., text information, RGB colors).

*The Complexity column depicts the time complexity of algorithms within the worst case. h is the iteration times and n, m refer to the number of nodes and edges, respectively. In addition, b involves the number of bits in the hash function, d means the average of node degrees, and k indicates the size of the largest mined substructure.

algorithm[3] [10, 27] is one of the most representative MPKs. 1-WL updates a node's label (or color) iteratively. The iteration of 1-WL is shown in Figure 3(d). At the hth 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-set[4] of labels $\{l^{(h-1)}(v), sort(l^{(h-1)}(u) : u \in \mathcal{N}(v))\}$. Subsequently, 1-WL employs an injective hash function $\phi(\cdot)$ to map the $\{l^{(h-1)}(v), sort(l^{(h-1)}(u) : u \in \mathcal{N}(v))\}$ into a new label $l^{(h)}(v)$. Formally:

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

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,

$$\phi_{wl}(\mathcal{G}) = \left[c^{(0)}(l_1^{(0)}), .., c^{(0)}(l_{I_0}^{(0)}); ...; c^{(H)}(l_1^{(H)}), \ldots, c^{(H)}(l_{I_H}^{(H)})\right], \tag{3}$$

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

$$K_{WL}(\mathcal{G}, \mathcal{G}') = < \phi_{wl}(\mathcal{G}), \phi_{wl}(\mathcal{G}') > . \tag{4}$$

The following works refine the 1-WL algorithm from two views: aggregation and relabeling. Hido and Kashima [28] replaced the hash function with a binary arithmetic giving rise to a faster

[3] 1-WL is also a well-known algorithm for graph isomorphism test.

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

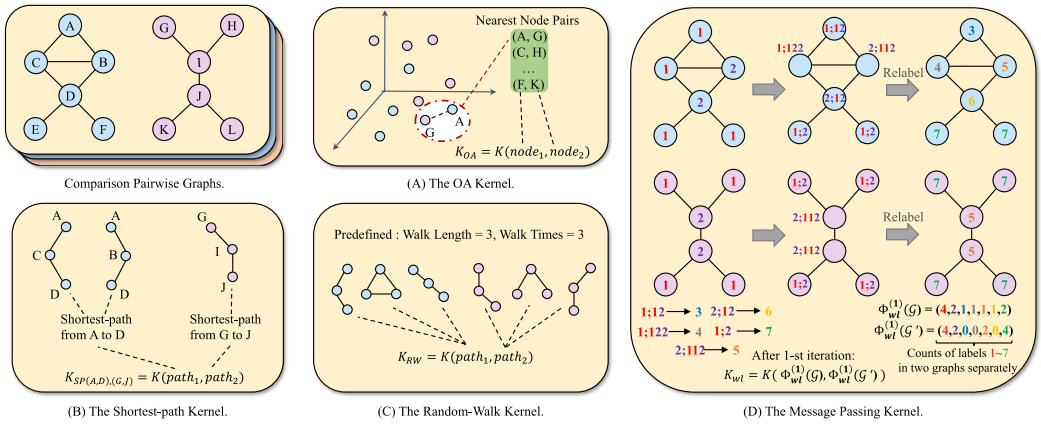


Fig. 3. Different mechanisms of four graph kernels in decomposing and comparing pairwise graphs.

$\phi(\cdot)$. Morris et al. [29] 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. [30] 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)** [31] 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 Figure 3(b)). For example, the shortest-path kernel [32] determines the shortest path between the vertices v and u via the Floyd-Warshall [57] or Dijkstra's [58] algorithms. The distance between the pairwise shortest paths from $\mathcal{G}$ and $\mathcal{G}'$ is defined as the kernel value between them. Formally,

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

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 [33] 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. [13] were 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

$$\begin{aligned} \mathcal{V}_\times &= \{(v,v') : v \in \mathcal{V} \wedge v' \in \mathcal{V}' \wedge l(v) \equiv l(v')\}, \\ \mathcal{E}_\times &= \{\{(v,v'),(u,u') \in \mathcal{V}_\times\} : \mathcal{E}_{v,u} \in \mathcal{E} \wedge \mathcal{E}'_{v',u'} \in \mathcal{E}'\}. \end{aligned} \tag{6}$$

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

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

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 Equation (7) assumes a uniform distribution for the beginning and ending probabilities of the walks across two graphs. However, Vishwanathan et al. [35] 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:

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

where μ_l is the convergence coefficient.

RWKs 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. [34] employed a second-order Markov random walk that considers the last two steps in the current random walk sequence when deciding the next step.

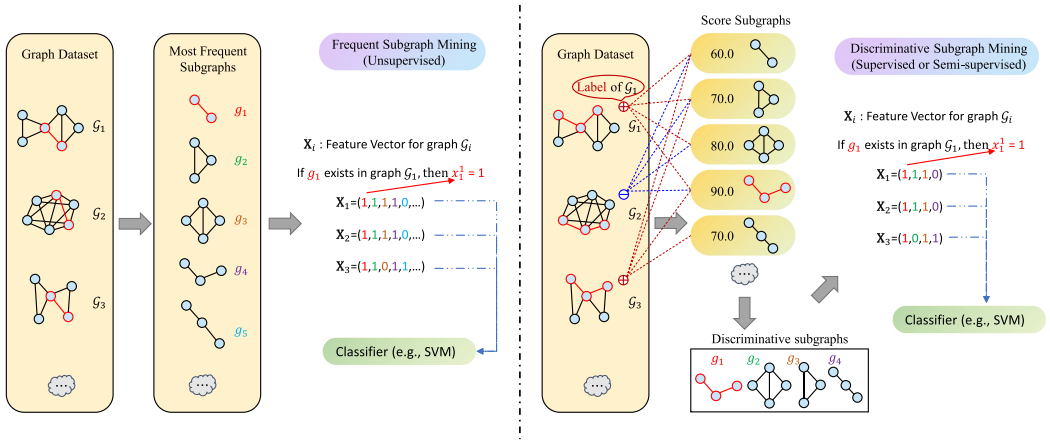
4.1.4 Optimal Assignment Kernels (OAKs). Fröhlich et al. [36] were the first to propose OAKs. OAKs consider nodes as a basic unit for measuring kernel values. Of all the GKs introduced in this article, OAKs are the only family of GKs that do not belong to R-Convolution paradigm. Specifically, given a fixed i in Equation (1), OAKs only add in the maximum similarity value between g_i and g'_j where $j \in \{1, \ldots, J\}$. Formally, OAKs are defined as:

$$K_{OA}(\mathcal{G}, \mathcal{G}') = \begin{cases} \max\limits_{\pi \in \prod_J} \sum_{i=1}^{I} K_{parts}\left(g_i, g'_{\pi[i]}\right), & \text{if } J \geq I, \\ \max\limits_{\pi \in \prod_I} \sum_{j=1}^{J} K_{parts}\left(g_{\pi[j]}, g'_j\right), & \text{otherwise}, \end{cases} \tag{9}$$

where $\prod_I$ represents all permutations of the indexes of a bag-of-graphs $\{1, \ldots, I\}$, 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. [38] 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)** [37, 39] are variants of OAKs. Unlike OAKs, which focus on comparing individual nodes between two graphs, TAKs assess pairs of nodes from each graph to measure graph-graph similarity. Additionally, TAKs take into account intermediary nodes between these pairs, allowing for the incorporation of more structural information. OAKs have been confined to node similarity measurement, although they can be extended to measure subgraph similarities to capture a graph's topological information [59]. 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 [40]—uses either **depth-first search (DFS)** or sampling to identify the subgraphs. With these subgraphs, the vector $\phi_{SG}(\mathcal{G}) = [c_{\mathcal{T}_1}^{(\mathcal{G})}, \ldots, c_{\mathcal{T}_N}^{(\mathcal{G})}]$ is then used to describe the graph $\mathcal{G}$, where $\mathcal{T}_i$ means the ith



(A) Frequent Subgraph Mining (FSM). (B) Discriminative Subgraph Mining (DSM).

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

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 $\phi_{SG}(\mathcal{G}')$:

$$K_{SG}(\mathcal{G}, \mathcal{G}') =< \phi_{SG}(\mathcal{G}), \phi_{SG}(\mathcal{G}') > . \tag{10}$$

There are several different implementations of SGKs kernel functions. For instance, Wale et al. [60] 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)** [42] 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)** [41] denote 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

Subgraph mining is similar to SGKs, where the vector $\mathbf{x}_i = [x_i^1, \ldots, x_i^M]^\top$ is taken as a graph-level representation of the graph $\mathcal{G}_i$. Here, $x_i^m \in \{0, 1\}$, $x_i^m = 1$ if $g_m \subseteq \mathcal{G}_i$, 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 $\mathcal{S}^* = \{g_1, \ldots g_T\}$ from the subgraph set $\{g_1, \ldots g_M\}$, where $g_1, \ldots, g_M$ denote all possible subgraphs of $\mathbb{G} = \{\mathcal{G}_1, \ldots, \mathcal{G}_N\}$. 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 Figure 4(a), while discriminative subgraph mining is the supervised or semi-supervised method (see Figure 4(b)).

4.2.1 Frequent Subgraph Mining (FSM). FSM identifies the subgraphs whose frequency of occurence in $\mathbb{G}$ sits over a predefined threshold δ. These subgraphs are then added to $\mathcal{S}^*$. Apriori-like algorithms, such as AGM [14] 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 $\mathcal{S}^*$. Other 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 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, CORK [46] proposes an evaluation criterion for the discriminative score of a subgraph set $\mathcal{S}$, $\mathcal{S} \subseteq \{g_1, \ldots, g_M\}$. Formally,

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

where $num(\cdot)$ 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:

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

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, then gSpan will no longer perform DFS along g_m. To speed up DSM based on discriminative scores and gSpan, LEAP [45] 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, gMLC [47] and gCGVFL [50] expand DSM to the multi-label[5] and multi-view[6] scenarios, respectively. Note, however, that all the DSM methods discussed are supervised methods. In terms of semi-supervised subgraph mining, gSSC [48] is proposed to map 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, gPU [49] utilizes the positively labeled graphs and unlabeled graphs to select $\mathcal{S}^*$ when performing binary graph classification tasks.

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) [53] summarizes the degree information of each node and its 1-hop neighbors as node features. LDP constructs graph representations by building an empirical distri-

[5]Each 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 (-).

[6]An 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.

bution 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] encode 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 [61], the spectrum of a graph is determined by its topology. Based on this theory, the **Family of Graph Spectral Distances (FGSD)** method [51] proposes that the distance between the spectrums of two graphs can be used to test whether the graphs are isomorphic. Thus, the histogram of the spectrum is used to construct a graph-level representation. Analogously, A-DOGE [56] 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 [54] uses stochastic approximations as a way of quickly calculating the distance between two graphs' spectral densities. More specifically, these works employ **Von Neumann Graph Entropy (VNGE)** [62, 63] 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. Another VNGE-based method [55] derives the error bound of the approximation estimation for VNGE.

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)**, **convolutional neural networks (CNNs)**, **capsule neural networks (CapsNets)**, and transformers to achieve Skip-gram-based (see Section 5.1), RNN-based (see Section 5.2), CNN-based (see Section 5.3), CapsNets-based (see Section 5.4) GL-DNNs, and Transformer-based GL-DNNs (see Section 5.5), respectively. The representative GL-DNNs mentioned in this section are summarized in Table 2.

5.1 Skip-gram-based GL-DNNs

Skip-gram [80] is a widely used unsupervised neural network model that predicts context words given a target word. Initially, the researchers built a skip-gram model based on the relationship between two adjacent subgraphs, namely, subgraph2vec [64]. Subgraph2vec first takes the (d-1)-, d-, (d+1)-hop neighborhoods of the vth selected node in the graph $\mathcal{G}_i$ as three subgraphs g^i_{v-1}, g^i_v, g^i_{v+1}, respectively, where $d \geq 1$ is a predefined value. $\{\mathbf{w}^1_{1-1}, \ldots, \mathbf{w}^1_{V+1}; \ldots; \mathbf{w}^N_{1-1}, \ldots, \mathbf{w}^N_{V+1}\}$ are the randomly initialized embeddings of all sampled subgraphs $\{g^1_{1-1}, \ldots, g^1_{V+1}; \ldots; g^N_{1-1}, \ldots, g^N_{V+1}\}$, 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 $\mathbf{w}^i_v$ as its input and predicts the context of $\mathbf{w}^i_v$ (i.e., $\mathbf{w}^i_{v-1}$ and $\mathbf{w}^i_{v+1}$). Then, the prediction results are back-propagated to update $\mathbf{w}^i_v$. To summarize, subgraph2vec's learning objective is to maximize the following probability Pr:

$$\sum_{i=1}^{N} \sum_{v=1}^{V} \log \Pr\left(\mathbf{w}^i_{v-1}, \ldots, \mathbf{w}^i_{v+1} \mid \mathbf{w}^i_v\right). \tag{13}$$

Another method, Graph2vec [65], was designed to tackle graph representation tasks. By establishing semantic associations between a graph and its sampled subgraphs, Graph2vec employs the

Table 2. Summary of Graph-Level Deep Neural Networks (GL-DNNs)

Architecture	Year	Method	Venue	Inputs	Complexity	Applications
Skipgram-based	2016	Subgraph2vec [64]	KDD MLG	**A, L**	$O(knm + hn)$	Molecule/Protein Classification; Malware detection
	2017	Graph2vec [65]	KDD MLG	**A, L**	$O(knm + hn)$	Molecule/Protein Classification; Malware detection
	2018	GE-FSG [66]	SDM	**A, L**	$O(nmn!m!)$	Molecule/Protein/Social Network Classification
RNN-based	2016	GGNN [67]	ICLR	**A, X**	$O(m)$	Code Retrieval
	2018	GAM [68]	KDD	**A, X**	$O(m)$	Molecule/Protein Classification
	2018	SAN [69]	AAAI	**A**	$O(nm)$	Molecule/Protein Classification
	2018	NetGAN [7]	ICML	**A, X**	$O(kmn)$	Social Network Generation
	2018	GraphRNN [8]	ICML	**A**	$O(nm)$	Protein/Social Network Generation
CNN-based	2016	PATCHYSAN [70]	ICML	**A, X, S**	$O(nkf)$	Molecule/Protein/Social Network Classification
	2016	DCNN [71]	NeurIPS	**A, X**	$O(n^2)$	Molecule/Protein Classification
	2017	ECC [72]	CVPR	**A, X, S**	$O(2m + n)$	Image/3D Point Clouds Classification
	2018	KCNN [73]	ICANN	**A, X**	$O(n^2)$	Molecule/Protein/Social Network Classification
CapsNet-based	2018	GCAPSCNN [74]	WCB	**A, X**	$O(n.c^2)$	Molecule/Protein/Social Network Classification
	2019	CapsGNN [75]	ICLR	**A, X**	$O(n.c^2)$	Molecule/Protein/Social Network Classification
	2019	PatchyCaps [76]	Arxiv	**A, X**	$O(n.c^2)$	Molecule/Protein Classification
Transformer-based	2020	GT [77]	AAAI DLG	**A, X, S**	$O(n^2 + m)$	Molecule Property Regression
	2021	Graphormer [78]	NeurIPS	**A, X, S**	$O(n^3 + m)$	Molecule/Protein Classification; Molecule Property Regression
	2022	TokenGT [79]	NeurIPS	**A, X, S**	$O((n + m)^2)$	Molecule Property Regression

*For the Inputs column, **A, L, X, S** refers to the adjacency matrix, node label matrix, node attribute matrix, and edge attribute matrix, respectively.

*The Complexity column depicts the time complexity of algorithms within the worst case. h is the iteration times, and n, m refer to the number of nodes and edges, respectively. In addition, f is the size of convolutional filters, k indicates the size of the largest substructure, and c means the number of statistical moments in capsules.

idea of Skip-gram to learn graph embeddings. Following this work, Dang et al. [66] 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 [81] and speech [82]. Two main types of algorithms 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 [68] 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. [69] 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 [7] 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 [8], creates various permutations of graphs, with various combinations of nodes and edges as sequential data for RNNs.

The second category of RNN-based GL-DNNs implements neural network versions of MPKs through RNN models. For instance, Scarselli et al. [83] recurrently updated node embeddings until reaching a stable situation, that is:

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

where 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. [67] 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:

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

where $\mathbf{h}_v^{(k)}$ represents the node representation of v at the kth 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:

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

where $f_t(\cdot)$ is a softmax function guided by an attention mechanism, which 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 [84, 85], are studied. Similar to RNN-based GL-DNNs, there are two main branches of CNN-based graph-level learning. In Appendix **??**, Figure **??** 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)** [73] 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 [86] 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:

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

where $f(\cdot)$ is a linear or sigmoidal function and $h_v^{(0)} = 0$. Another related work, ECC [72], 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 kth 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.

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

Moreover, **Diffusion CNN (DCNN)** [71] expands the diffusion process on graphs to derive the probability transition matrix $\mathbf{P} = [\mathbf{A}, \mathbf{A}^2, \ldots, \mathbf{A}^h] \in \mathbb{R}^{h \times n \times n}$, which presents the transition probability of information transfer from one node to other nodes during h diffusion steps. The representations of nodes within h steps is presented as $\mathbf{PX} \in \mathbb{R}^{h \times n \times f}$. For graph classification tasks, DCNN permutes the dimensions giving $\mathbf{PX} \in \mathbb{R}^{n \times h \times f}$, 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. Although it is similar to MPKs that aggregate multi-hop neighborhoods, DCNN leverages diffusion process to directly aggregate features over multiple steps instead of layer-by-layer. This unique characteristic allows DCNN to capture global topology and direct control over the scale of information aggregation through diffusion steps.

5.4 CapsNet-based GL-DNNs

CapsNets [87] 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) [74] iteratively aggregates neighbor information under different statistical moments (e.g., mean, standard deviation) to form a capsule representation of the central node, formulated as:

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

where $(\mathbf{W}_1, \mathbf{W}_2, \ldots)$ 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, then 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:

$$\mathbf{h}_{\mathcal{G}} = \frac{1}{n}(H^{(K)} - \mu)^\top (H^{(K)} - \mu). \tag{20}$$

CapsGNN [75] 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 to generate a graph capsule that can be plugged into a capsule network for graph classification. Mallea et al. [76] employ 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.

5.5 Transformer-based GL-DNNs

Recently, transformers [88] have achieved remarkable success in many fields, such as Natural Language Processing [88] and Computervision [89], attracting interest from the graph-level learning community. Transformers are initially designed for modeling relationships among all pair-wise entities (e.g., words, pixels); such relationships only consider semantic relationships while overlooking structure. Therefore, it still requires endeavors to generalize transformers on graph-level tasks. Dwivedi and Bresson [77] proposed an early Transformer-based GL-DNN, which limits transformers into the neighborhood range as the structure inductive bias, that is,

$$a_{vu}^{(k-1)} = \text{softmax}_u\left(\frac{\mathbf{q}_v^{(k-1)} \cdot \mathbf{k}_u^{(k-1)}}{\sqrt{d_z}}\right), \ \mathbf{h}_v^{(k)} = \sum_{u \in \mathcal{N}(v)} a_{vu}^{(k-1)} \mathbf{v}_u^{(k-1)}, \\ \text{where } \mathbf{q}_v^{(k-1)} = \mathbf{h}_v^{(k-1)} \mathbf{W}_Q, \ \mathbf{k}_v^{(k-1)} = \mathbf{h}_v^{(k-1)} \mathbf{W}_K, \text{ and } \mathbf{v}_v^{(k-1)} = \mathbf{h}_v^{(k-1)} \mathbf{W}_V, \tag{21}$$

where $\mathbf{h}_v^{(k)} \in R^{1*d}$ represents the node representation of v at the kth layers, $\mathbf{W}_Q \in R^{d*d_z}, \mathbf{W}_K \in R^{d*d_z}$, and $\mathbf{W}_V \in R^{d*d_z}$ are the projection matrix to map the node representations as query, key, value vectors, and $\mathcal{N}(v)$ means the set of nodes in the v's neighborhood including itself. Graphormer [78] frees transformers for all node pairs on the graph while encoding structure information (i.e., degree centrality and shortest path) for modeling relationships. In addition, TokenGT [79] treats all nodes and edges in the graph as inputs and augments them with orthonormal identifiers that allow transformers to recognize and exploit the connectivity structure of the graph. The graph-level representation can be obtained by inserting an extra trainable graph entity into TokenGT.

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)—is introduced first, followed by emerging methods in GL-GNNs, such as subgraph-based (see Section 6.2), graph kernel-based (see Section 6.3), and Transformer-based (see Section 6.5) approaches. Notably, subgraph- and graph kernel-based 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 3 for the GL-GNNs discussed in this section and Table 6 in Appendix D for comparisons of GL-GNNs/DNNs architectures.

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 Equation (15), but MPNNs set different weights

Table 3. Summary of Graph-level Graph Neural Networks (GL-GNNs)

Architecture	Year	Method	Venue	Inputs	Complexity	Applications
Message -passing Neural Networks	2015	Fingerprint [90]	NeurIPS	A, X	$O(m)$	Molecule/Protein Classification
	2016	GraphSim [91]	NeurIPS	A, X, S	$O(m)$	Physical State Prediction
	2017	MPNN [25]	ICML	A, X, S	$O(m)$	Molecule/Protein Property Regression
	2017	DTNN [92]	NC	A, X	$O(m)$	Molecule/Protein Classification
	2019	GIN [3]	ICLR	A, X	$O(m)$	Molecule/Protein/Social Network Classification
	2019	K-GNNs [93]	AAAI	A, X	$O(n^4)$	Molecule/Protein/Social Network Classification
	2019	PPGN [5]	NeurIPS	A, X	$O(n^3)$	Molecule/Protein/Social Network Classification; Molecule Property Regression
	2019	RP [94]	ICML	A, X, S	$O(n!)$	Molecule/Protein Classification
	2021	FGNN [95]	ICLR	A, X	$O(n^2)$	Molecule/Protein/Social Network Classification
	2021	SWL [96]	ICML	A, X, S	$O(n^k)$	Molecule/Protein/Social Network Classification
	2021	CWN [97]	NeurIPS	A, X, S	$O(n^k)$	Molecule/Protein/Social Network Classification; Molecule Property Regression
	2021	RNI [98]	IJCAL	A, X	$O(n^2)$	Molecule/Protein Classification
Subgraph -based	2020	SubGNN [99]	NeurIPS	A, X	$O(n^k)$	Molecule/Protein Classification; Clinical Diagnostic
	2021	SUGAR [100]	WWW	A, X	$O(knm)$	Molecule/Protein Classification
	2021	NGNN [101]	NeurIPS	A, X, S	$O(n^3)$	Molecule/Protein Classification; Molecule Property Regression
	2022	GNN-AK [102]	ICLR	A, X	$O(knm)$	Molecule/Protein Classification; Molecule Property Regression
	2022	GraphSNN [103]	ICLR	A, X	$O(m)$	Molecule/Protein Classification
	2022	ESAN [104]	ICLR	A, X	$O(n^4)$	Molecule/Protein/Social Network Classification
	2022	GSN [105]	TPAMI	A, X	$O(nd)$	Molecule/Protein/Social Network Classification
	2023	I^2-GNN [106]	ICLR	A, X, S	$O(nd^5)$	Molecule/Protein Classification; Cycles counting
Kernel -based	2019	GNTK [24]	NeurIPS	A, X	$O(nd^2)$	Molecule/Protein Classification
	2019	DDGK [107]	WWW	A, X, S	$O(hn)$	Molecule/Protein Classification
	2020	GCKN [108]	ICML	A, X	$O(nd^k)$	Molecule/Protein/Social Network Classification
	2020	RWNN [109]	NeurIPS	A, X	$O(n^3)$	Molecule/Protein/Social Network Classification
	2021	GSKN [110]	WWW	A	$O(nd^k)$	Molecule/Protein/Social Network Classification
	2022	KP-GNN [111]	NeurIPS	A, X, S	$O(n^2)$	Molecule/Protein/Social Network Classification
Spectral -based	2016	ChebNet [112]	NeurIPS	A, X	$O(m)$	Image/Text Classification
	2021	GNNTFS [113]	JMLR	A, X	$O(m)$	Molecule/Protein Classification
	2021	GNNMatlang [114]	ICML	A, X	$O(n^3)$	Molecule/Protein/Social Network Classification
	2021	ARMA [115]	TPAMI	A, X	$O(m)$	Molecule/Protein/Image/News Classification
	2021	UFG [116]	ICML	A, X	$O(m)$	Molecule/Protein Classification
Transformer -based	2020	GTransformer [6]	NeurIPS	A, X, S	$O(m+n^2)$	Molecule/Protein Classification; Molecule Property Regression
	2022	GraphGPS [117]	NeurIPS	A, X, S	$O(m+n)$	Molecule/Protein Classification; Molecule Property Regression
	2022	SAT [116]	ICML	A, X	$O(m+n^2)$	Molecule/Protein Classification; Molecule Property Regression

*For the Inputs column, **A**, **X**, **S** refer to the adjacency matrix, node attribute matrix, and edge attribute matrix, respectively.

*The Complexity column depicts the time complexity of algorithms within the worst case. h is the iteration times and n, m refers to the number of nodes and edges, respectively. In addition, d means the large value of node degrees, k indicates the size of the largest sampled substructure, and p is the degree of the polynomial.

in separate layers rather than sharing weights in all layers. Gilmer et al. [25] summarize a collection of MPNNs [90–92] and further propose a unified framework for this branch of techniques, as shown in Figure 5(a) and denoted as:

$$\mathbf{h}_v^{(k)} = U^{(k)}\left(\mathbf{h}_v^{(k-1)}, \sum_{u \in \mathcal{N}(v)} M^{(k)}\left(\mathbf{h}_v^{(k-1)}, \mathbf{h}_u^{(k-1)}, \mathcal{E}_{v,u}\right)\right), \tag{22}$$

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:

$$\mathbf{h}_{\mathcal{G}} = \text{readout}\left(\mathbf{h}_v^{(k)} : v \in \mathcal{V}\right). \tag{23}$$

MPNNs have become the mainstream of graph-level studies [25]. They are also representative of spatial-based GL-GNNs, since they are easy to use through matrix operations. Last, 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).

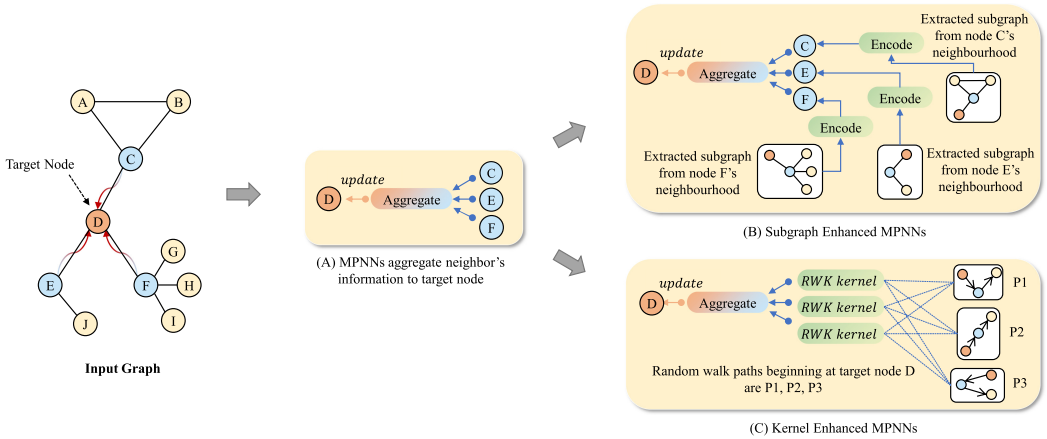


Fig. 5. Different mechanisms of MPNNs, Subgraph Enhanced MPNNs, and Kernel Enhanced MPNNs. In Subgraph Enhanced MPNN, we used 1-hop neighborhoods as the subgraph for easy understanding, but the specific subgraph extraction is up to the article.

6.2 Subgraph-based GL-GNNs

The widespread of MPNNs attracts researchers' interest in exploring their upper-bound capability for distinguishing structures. Recently, some studies [111, 118] have identified flaws in MPNNs, noting that they fail to perform in certain cases, such as regular graphs [94, 106, 119, 120]. The reason for these flaws is that MPNNs, relying on aggregating neighborhood information, are restricted by local topology. Thus, investigating GL-GNNs capable of capturing more topological information has been a crucial stream of study. As yet, 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 Figure 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)** [105], for example, counts the number of occurrences of a predefined subgraph pattern $g_1, \ldots, g_M$ (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:

$$\begin{cases} x_v^{g_m} = |\{g_s \simeq g_m : v \in \mathcal{V}, v \in \mathcal{V}_{g_s}, g_s \subseteq \mathcal{G}\}|, \quad \mathbf{x}_v^g = [x_v^{g_1}, \ldots, x_v^{g_M}]^\top \text{ (Node)}, \\ S_{v,u}^{g_m} = |\{g_s \simeq g_m : \mathcal{E}_{v,u} \in \mathcal{E}, \mathcal{E}_{v,u} \in \mathcal{E}_{g_s}, g_s \subseteq \mathcal{G}\}|, \quad \mathbf{S}_{v,u}^g = [S_{v,u}^{g_1}, \ldots, S_{v,u}^{g_M}]^\top \text{ (Edge)}, \end{cases} \tag{24}$$

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:

$$\mathbf{h}_v^{(k)} = U^{(k)}\left(\mathbf{h}_v^{(k-1)}, \mathbf{m}_v^{(k)}\right), \quad \mathbf{m}_v^{(k)} = \begin{cases} \sum\limits_{u \in \mathcal{N}(v)} M^{(k)}\left(\mathbf{h}_v^{(k-1)}, \mathbf{h}_u^{(k-1)}, \mathbf{x}_v^g, \mathbf{x}_u^g, \mathcal{E}_{v,u}\right) \text{(Node)}, \\ \sum\limits_{u \in \mathcal{N}(v)} M^{(k)}\left(\mathbf{h}_v^{(k-1)}, \mathbf{h}_u^{(k-1)}, \mathbf{S}_{v,u}^g, \mathcal{E}_{v,u}\right) \text{(Edge)}. \end{cases} \tag{25}$$

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 [102] samples neighborhood subgraphs as supplementary information for MPNN learning. 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.:

$$\mathbf{x}_v^g = \left[Emb\left(v|g_v\right) \mid \sum_{u \in \mathcal{V} \wedge u \neq v} Emb\left(u|g_v\right) \mid \sum_{u \in \mathcal{V} \wedge u \neq v} Emb\left(v|g_u\right) \right], \tag{26}$$

where $Emb\left(v|g_v\right)$ is the embedding of node v produced by running the base MPNN on subgraph g_v, $Emb\left(v|g_u\right) == 0$ 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)** [101] extracts nodes (i.e., $\mathcal{N}(v) \cup v$) and edges (i.e., $\mathcal{E}_{v_1,v_2} \in \mathcal{E} \quad \& \quad v_1, v_2 \in \mathcal{N}(v) \cup v$) 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 [103] 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 [100], 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)** [99] 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)** [104], 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 [121] compresses all the subgraph embeddings into one graph-level representation. In addition, I^2-GNN [106] extracts k-hop neighborhood subgraphs from each root node while assigning distinct identifiers for the root node and its neighbors within

each subgraph. By aggregating all subgraph embeddings, I^2-GNN obtains powerful graph-level representations that are proven capable of distinguishing cycles of length 3 to 6.

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 Figure 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. Another branch involves designing differentiable and parameterized kernels by plugging kernels into neural networks.

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) to capture more structure information.

For example, **Graph Convolutional Kernel Networks (GCKN)** [108] and **Graph Structured Kernel Networks (GSKN)** [110] 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 kth 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 $K\left(R(P_i), R(P_j)\right)$ (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:

$$h_v^{(k)} = \sum_{1 \le i \le q} \left[K\left(R(P_1), R(P_i)\right), \ldots, K\left(R(P_q), R(P_i)\right)\right]^\top. \tag{27}$$

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. [24] combined a **Neural Tangent Kernel (NTK)** [122] 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 topology information to encode into the node embeddings. Recently, Feng et al. [111] proposed to leverage the shortest path kernel and graph diffusion kernel to filter duplicate nodes in K-hop neighborhoods, pursuing higher expressivity of K-hop MPNNs (i.e., MPNNs aggregate and update node embeddings via k-hop neighborhood information).

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. [123] 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) [107] take M base graphs $\{\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_M\}$ to represent a target graph $\mathcal{G}_t$ as M-dimensional vectors $\mathbf{h}_{\mathcal{G}_t} = [K_D(\mathcal{G}_1, \mathcal{G}_t), \ldots, K_D(\mathcal{G}_M, \mathcal{G}_t)]^\top$, 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 $\{\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_M\}$ to train an encoder $\{\mathcal{Z}_1, \mathcal{Z}_2, \ldots, \mathcal{Z}_M\}$. The encoder $\mathcal{Z}_m$ takes the one-hot encoding of nodes (e.g., the first node's encoding is $[1, 0, 0, \cdots]^\top$) 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, then the correct output should be $[0, 1, 1, 0, \cdots]^\top$). 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:

$$K_D(\mathcal{G}_m, \mathcal{G}_t) = \sum_{v_i, v_j \in \mathcal{V}_t, \mathcal{E}_{i,j} \in \mathcal{E}_t} -\log\left(v_j | v_i, \mathcal{Z}_m\right). \tag{28}$$

Random Walk graph Neural Networks (RWNN) [109] also derive a trainable random walk kernel $K_{RW}(\cdot, \cdot)$ through a series of learnable graph patterns $\{\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_M\}$. 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 $\mathbf{h}_{\mathcal{G}_t} = [K_{RW}(\mathcal{G}_1, \mathcal{G}_t), \ldots, K_{RW}(\mathcal{G}_M, \mathcal{G}_t)]^\top$ of the target graph $\mathcal{G}_t$ for graph classification tasks. Correspondingly, the classification results are back-propagated 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 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. [124], which designed graph convolutions via the spectral graph theory [61]. Recently, Balcilar et al. [125] 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 [126]. However, with graph-level tasks, graph signals beyond the low frequency may be critical, since they can highlight the differences between different graphs [127], 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 [61], 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 [128, 129]. Specifically, $\{\lambda_1, \ldots, \lambda_n\}$ where $0 \leq \lambda_1 \leq \ldots \leq \lambda_n \leq 2$, and $\mathbf{U} = (\mathbf{u}_1, \ldots, \mathbf{u}_n)$ 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 $\hat{\mathbf{x}} = \mathbf{U}^\mathrm{T}\mathbf{x}$, 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 [112] defines the spectral graph convolution as $\mathbf{U}\mathrm{diag}(\Phi(\mathbf{\Lambda}))\mathbf{U}^\mathrm{T}\mathbf{x}$, where $\Lambda = \mathrm{diag}(\{\lambda_i\}_{i=1}^{i=n})$, $\Phi(\mathbf{\Lambda}) = \sum_{k=0}^{K} \theta_k \mathbf{\Lambda}^k$ 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 related to downstream tasks [125]. However, only a few practitioners have designed graph-level neural networks from the perspective of spectral graph convolution [114, 115]. 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. [113]

theoretically proved the transferability of spectral filters on multigraphs. Balcilar et al. [114] 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. [115] 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. Zheng et al. [116] 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. In addition, Wang et al. [130] analyzed various spectral GL-GNNs to find that an orthogonal basis whose weight function is related to the density of graph signals can maximize the convergence speed. Thus, they designed a novel spectral graph convolution via the Jacobi basis, which is orthogonal and flexible to adapt to a wide range of weight functions.

6.5 Transformer-based GL-GNNs

Transformers [88] have attracted growing interest from the graph-level learning community. In addition to generalizing it directly on graph-level tasks (see Section 5.5), a more efficient way is to leverage transformer architecture for raising up GL-GNNs. The addition of transformers into GL-GNNs offers numerous advantages, such as capturing long-range interactions and relieving strict structural inductive bias. GTransformer [6] welds GL-GNNs and transformer as a hybrid layer, passing the node representations learned by GL-GNNs through a subsequent transformer. Thus, the final node representations contain both local and long-range topology information to capture precise graph embeddings. GraphGPS [117] constructs the framework that ensembles various GL-GNNs and transformers, achieving state-of-the-art results on multiple graph-level tasks. In addition, SAT [131] employs GL-GNNs and transformers to generate and update subgraph embeddings, respectively, and sums subgraph embeddings as the graph representation.

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 Sections 7.1 and 7.2). We summarize all discussed pooling approaches in Table 7 in Appendix E and compare different pooling architectures in Table 8 in Appendix E.

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 [3, 90], 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 Figure 6(a). It is common to see practitioners aggregating node embeddings via summation, maximization, minimization, mean, and concatenation functions. For example:

$$\mathbf{h}_{\mathcal{G}} = \sum_{v \in \mathcal{V}} \mathbf{h}_v \Big\| \max_{v \in \mathcal{V}} / \min_{v \in \mathcal{V}} (\mathbf{h}_v) \Big\| \frac{1}{|\mathcal{V}|} \sum_{v \in \mathcal{V}} \mathbf{h}_v \Big\| [\mathbf{h}_{v_1} | \ldots | \mathbf{h}_{v_{|\mathcal{V}|}}]. \tag{29}$$

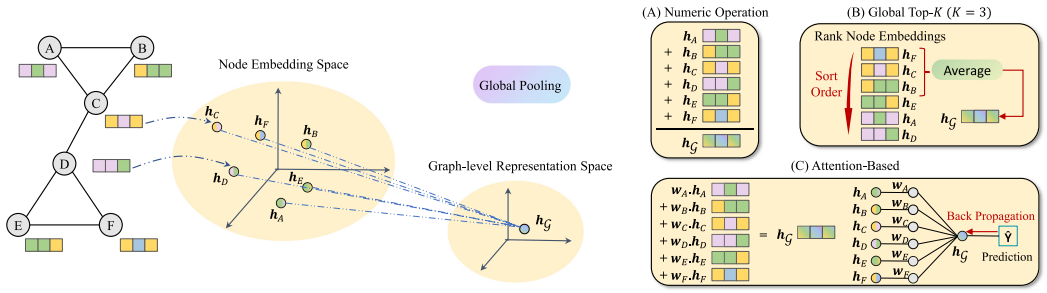


Fig. 6. Illustrative examples of Global Pooling methods.

Duvenaud et al. [90] empirically proved that, in graph-level learning, summation has no weaker an outcome than a hash function. Similarly, GIN [3] 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)** [132] employs all three of these functions to pool the node embeddings, while TextING [133] 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)** [92] applies a neural layer that processes the node embeddings before the summation function and **second-order pooling (SOPOOL)** [134] is executed as $\mathbf{h}_{\mathcal{G}} = [\mathbf{h}_{v_1}^T \mathbf{h}_{v_1} | \ldots | \mathbf{h}_{v_{|\mathcal{V}|}}^T \mathbf{h}_{v_{|\mathcal{V}|}}]$.

7.1.2 Attention-based. The contributions of node embeddings to graph-level representations could not be equal, as some contain 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 Figure 6(c). Li et al. [67] and Duvenaud et al. [90], for example, both employ a softmax function as an attention-based global pooling for aggregation. This can be written as:

$$\mathbf{h}_{\mathcal{G}} = \sum_{v,k} \text{softmax}\left(w_v^k, \mathbf{h}_v^k\right), \tag{30}$$

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 [135] 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 Global Top-K. Global top-K graph pooling sorts all nodes and selects the first K node embeddings for aggregation, as shown in Figure 6(b). In this way, the pooling layer only preserves K significant vertices and drops out others. SortPool [4] regards the last layer of GL-GNNs containing fine-grained topology information, thus, for ranking vertices. Subsequently, the highest-scored K nodes are embedded by contacting all layers' outputs in GL-GNNs, which come together to form the graph-level representation. **Graph Self-adaptive Pooling (GSAPool)** [136] 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 taken the same as SortPool, while the feature scores are learned by feeding the node features into an MLP.

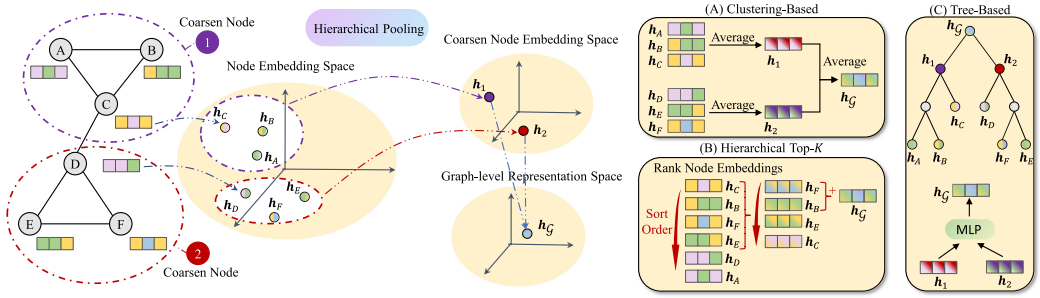


Fig. 7. Illustrative examples of hierarchical pooling methods.

7.2 Hierarchical Graph Pooling

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.

7.2.1 Prior Clustering. Clustering methods were originally designed to capture the hidden hierarchical structures in graphs, which can be incorporated into the pooling process. Figure 7(a) demonstrates clustering-based graph pooling, which has been the focus of many studies. The prior clustering indicates that the graph pooling is served by off-the-shelf clustering methods, which partition graphs based on inherent properties rather than model performance. For instance, Henaff et al. [137] implemented multi-resolution spectral clustering [138] that assigns each node to a matched cluster. Subsequently, the clusters in the input graph are treated as coarsened nodes in the coarsened graph. The embedding of the coarsened 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, EigenPool [139] involves a spectral clustering method that coarsens graphs and pools node embeddings into cluster-level embeddings by converting spectral-domain signals, and Bruna et al. [140] adopted hierarchical agglomerative clustering [141] to coarsen graphs. The prior clustering methods are implemented conveniently and efficiently, however, they cannot optimize the clustering process via results on downstream tasks, falling into sub-optimal performance.

7.2.2 Trainable Clustering. To overcome the limitations of prior clustering methods, researchers build the end-to-end architecture for clustering and coarsening graphs, that is, trainable clustering. **Graph Multiset Transformer (GMT)** [142] uses a multi-head self-attention mechanism to cluster nodes into different sets according to the final task, and the graph-level representation is therefore derived through these sets. MinCutPool [143] 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. The above trainable clustering methods follow the fashion of hard assignment, yet, DiffPool [144] 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 kth 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 kth 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:

$$\mathbf{S}^{(k)} = \text{softmax}\left(\text{Conv}^{(k)}\left(\mathbf{A}^{(k)}, \mathbf{H}^{(k)}, \mathbf{W}^{(k)}\right)\right), \tag{31}$$

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 kth layer, respectively. $\mathbf{W}^{(k)} \in \mathbb{R}^{f \times n^{(k+1)}}$ is the trainable weight matrix, and softmax $(\cdot)$ function is applied to each row. In addition, StructPool [145] supports both hard and soft assignment clustering by training assignment matrix via softmax and argmax functions, respectively.

7.2.3 Hierarchical Top-k. The high complexity of the clustering process exacerbates the computational cost burden of graph pooling. For example, the DiffPool [144] 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 Figure 7(b). Graph U-Nets [146], for example, project each node feature into a 1-dimensional vector $\mathbf{Y}$ as the rank score. Subsequently, the K nodes with the highest score are selected to form the new coarsened graph, which is defined as:

$$\mathbf{Y} = \frac{\mathbf{Z}^{(l)}\mathbf{P}^{(l)}}{||\mathbf{P}^{(l)}||}, \quad \text{idx} = \text{Top}\, K\,(\mathbf{Y}), \quad \mathbf{Z}^{(l+1)} = \left(\mathbf{Z}^{(l)} \odot \left(\text{sigmoid}\,(\mathbf{Y}_{\text{idx}})\,\mathbf{1}_Z^\top\right)\right), \quad \mathbf{A}^{(l+1)} = \mathbf{A}^{(l)}_{\text{idx,idx}}, \tag{32}$$

where $Z^{(l)}$ is the input node features at the layer l, $\mathbf{P}^{(l)}$ is a learnable projection matrix, 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. [147] 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 [148] 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. [149] 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 [150] that scores each cluster based on the node embeddings within it.

7.2.4 CNN-based. CNN-based hierarchical graph pooling tends to partition a graph into several small subgraphs within close sizes, where each subgraph is coarsened by performing a regular 1-D pooling. For instance, ChebNet [112] and MoNet [151] use the Graclus [152] algorithm to cluster pair-wise nodes in the graph, then perform a max pooling between the clustered node pairs to obtain the coarsened nodes. Similarly, NDP [153] partitions graphs based on the MAXCUT objective and pools in the partitioned subgraphs. These methods are equivalent to CNNs that compute local characteristics for pooling.

7.2.5 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 Figure 7(c). Wu et al. [154] use a structure encoding tree [155] 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,

Table 4. Summary of Selected Benchmark Datasets

Domain	Dataset	Size	#Graphs	Average #Nodes	Average #Edges	Node Attr.	Edge Attr.	#Classes	Source
Biology	ENZYMES	Small	600	32.6	62.1	✓	—	6	[160, 166]
	PROTEINS	Small	1,113	39.1	72.8	✓	—	2	[160, 166]
	D&D	Small	1,178	284.3	715.7	✓	—	2	[160, 167]
	BACE	Small	1,513	34.1	36.9	✓	✓	2	[2, 168]
	MUV	Medium	93,087	24.2	26.3	✓	✓	2	[2, 169]
	ppa	Medium	158,100	243.4	2,266.1	—	✓	37	[161, 170]
Chemistry	MUTAG	Small	188	17.9	19.8	✓	✓	2	[42, 160]
	SIDER	Small	1,427	33.6	35.4	✓	✓	2	[2, 171]
	ClinTox	Small	1,477	26.2	27.9	✓	✓	2	[2, 172]
	BBBP	Small	2,039	24.1	26.0	✓	✓	2	[2, 173]
	Tox21	Small	7,831	18.6	19.3	✓	✓	2	[2, 174]
	ToxCast	Small	8,576	18.8	19.3	✓	✓	2	[2, 175]
	MolHIV	Small	41,127	25.5	27.5	✓	✓	2	[2, 161]
	MolPCBA	Medium	437,929	26.0	28.1	✓	✓	2	[2, 161]
	FreeSolv	Small	642	8.7	8.4	✓	✓	—	[2, 176]
	ESOL	Small	1,128	13.3	13.7	✓	✓	—	[2, 177]
	Lipophilicity	Small	4,200	27.0	29.5	✓	✓	—	[2, 178]
	AQSOL	Small	9,823	17.6	35.8	✓	✓	—	[163, 179]
	ZINC	Small	12,000	23.2	49.8	✓	✓	—	[163, 180]
	QM9	Medium	129,433	18.0	18.6	✓	✓	—	[2, 160]
Social Networks	IMDB-BINARY	Small	1000	19.8	96.5	—	—	2	[160, 181]
	IMDB-MULTI	Small	1,500	13.0	65.9	—	—	3	[160, 181]
	DBLP_v1	Small	19,456	10.5	19.7	✓	✓	2	[160]
	COLLAB	Medium	5,000	74.5	2,457.8	—	—	3	[160, 181]
	REDDIT-BINARY	Small	2,000	429.6	497.8	—	—	2	[160, 181]
	REDDIT-MULTI-5K	Medium	4,999	508.5	594.9	—	—	5	[160, 181]
	REDDIT-MULTI-12K	Medium	11,929	11.0	391.4	—	—	11	[160, 181]
Computer Vision	CIFAR10	Medium	60,000	117.63	941.1	✓	—	10	[163, 182]
	MNIST	Medium	70,000	70.57	564.53	✓	—	10	[163, 183]
	MODELNET40	Medium	12,311	—	—	✓	—	40	[165]
Code Retrieval	Code2	Medium	452,741	125.2	124.2	✓	✓	—	[161, 184]
Cybersecurity	MALNET	Large	1,262,024	15,378	35,167	—	—	696	[162]

*Node Attr. and Edge Attr. indicate the labels or features of nodes and edges, respectively.
*The size of datasets follows the setting of OGB [161], medium datasets have more than 1 million nodes or more than 10 million edges, and large datasets own over 100 million nodes or 1 billion edges.

Wu et al. [156] empirically verified that the hierarchical tree pooling guided by structure entropy [157, 158] can preserve higher-quality structural information than U-Nets and MinCutPool. Alternatively, EdgePool [159] scores edges based on the features of two endpoint nodes, eliminating the highest-ranked edge by merging its two end nodes while maintaining all neighbors. The features of the newly generated node are obtained by summing two node features for passing a tanh gating function. EdgePool merges two child nodes in a tree into a father node.

8 Benchmarks

8.1 Datasets

Table 4 summarizes a selection of benchmark graph-level datasets, including TUDataset [160], **Open Graph Benchmark (OGB)** [161], MOLECULENET [2], MALNET [162], and others [163]. TUDataset [160], consisting of molecules, proteins, images, social networks, synthetic graphs, and data from many other domains, has been widely used to evaluate graph-level learning approaches. However, despite its wide use, it has attracted criticism from some practitioners. For example, Ivanov et al. [164] contend 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. [163] 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 for their property prediction. Dwivedi et al. [163] transformed images into graphs for classification, in which a group of pixels is clustered as a node. In addition, 3D point clouds are naturally represented as graph datasets. For instance, MODELNET40 [165] consists of 12,311 meshed models, and each model can be treated as a single graph by sampling 1,000 points as nodes and constructing edges between all neighbors [72]. Based on real-world cybersecurity scenarios, Freitas et al. [162] proposed a large-scale graph dataset of over 1.2 million graphs with imbalanced labels. Recently, OGB [161] 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. [185] 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 [186]. 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. [163] built a benchmark framework based on PyTorch and DGL[7] that evaluates models on graph classification and graph regression tasks with an 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 [161] 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. [187] provided a benchmark framework for graph contrastive learning.

9 Downstream Tasks and Applications

This section introduces seven common downstream tasks with corresponding applications and metrics of graph-level learning, including **Graph Classification**, which learns the mapping relationship between graphs and corresponding class labels and predicts the discrete labels of unseen graphs; **Graph Regression**, which predicts the continuous properties of graphs. Taking molecules as examples, graph regression can predict different molecular properties related to the tightness of chemical bounds, fundamental vibrations, the state of electrons in molecules, the spatial distribution of electrons in molecules, and others; **Graph Generation**, aiming to generate new graphs that have specific properties based on a series of graphs; **Graph Comparison**, which evaluates the distance or similarity between graphs; **Subgraph Discovery** for detecting discriminative substructures in a graph dataset; **Graph Ranking**, sorting a set of graphs into a particular order based on the importance and relevance to the specific tasks, namely, graph ranking; and **Applying Complex Scenarios**, referring to graph-level learning extended to complex scenarios, such as

[7]DEEP GRAPH LIBRARY: https://www.dgl.ai

multi-graph-view [188, 189], semi-supervised setting [48], and so on. Details of related applications and metrics are shown in Appendix **??**.

10 Future Directions

In this section, we spotlight 12 future directions of graph-level learning, involving technical challenges, application issues, and potential opportunities, for readers to refer to.

10.1 Neural Architecture Search (NAS) for GL-GNNs

Challenges: Existing GL-GNNs often have a complex architecture, consisting of a number of different components, e.g., multiple graph convolutions and graph pooling layers. Developing effective NAS methods to free researchers from repeatedly searching for good architectures manually and, in turn, tuning the parameters is an urgent goal. However, the multitude of optional components with numerous parameters results in an exceedingly vast parameter search space for GL-GNNs. Additionally, diverse graphs could desire distinct optimal parameters, raising the demand for a globally searched architecture that fits all graphs.

Opportunities: Although NAS has attracted a bulk of interest from the deep learning community, there is limited exploration for NAS on GL-GNNs. To fill this gap, one strategy could be leveraging reinforcement learning or policy gradients to restrict the entire architecture space to an optimal subspace by maximizing expected results (e.g., accuracy). Another strategy could involve estimating optimal parameters based on intrinsic graph properties. This approach would enable the consideration of diverse graph properties, thereby providing a globally applicable architecture for entire graph datasets. For instance, Yang et al. [190] estimated the embedding dimensionality for each graph via their entropy and sought a few centers of all individual estimated dimensionalities as final embedding dimensions for encoding the graph dataset.

10.2 Geometrically Equivariant GL-GNNs

Challenges: In geometric graphs [191], 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 that together comprise the geometric vector. A key challenge for GL-GNNs working on geometric graphs is the equivariant, i.e., when inputting a geometric graph with a specific rotation into a GL-GNN, the subsequent output should reflect the same rotation.

Opportunities: Geometrically Equivariant GL-GNNs have been a newly raised field in recent years; as yet, there is limited research [192, 193]. A straightforward direction could be to extend the current GL-GNN architecture on both node features and geometric vectors to construct the GL-GNNs on geometric graphs. For instance, **Equivariant Graph Neural Networks (EGNN)** [192] expand MPNNs aggregating both feature vectors and geometric vectors, and AMP [194] designs a novel message-passing scheme considering the directional effects from an external field or a set of particles unreachable in a graph. In addition, we argue that presenting geometric graphs as hypergraphs or multi-view graphs can also help us better learn complex geometric features.

10.3 Global Explainable GL-GNNs

Challenges: Providing global explanations for GL-GNNs' prediction could offer high-level insight into model behavior and task mechanism. For example, we can learn why models diagnose brain networks as normal or disordered without case-by-case explanations. However, building global explainable GL-GNNs is non-trivial, since it requires an excellent abstraction ability to explain generic patterns of a set of graphs, rather than only recognizing the significant part in a single graph.

Opportunities: Most algorithms for explaining the predictions of GL-GNNs focus on instances (e.g., PGExplainer [195], GSAT [196]); globally explainable GL-GNNs is less studied. A promising direction in building globally interpretable GL-GNN could be summarizing rules from all instance-level explanations via statistical inference methods, e.g., logic combination [197], Bayes theorem, and so on. Furthermore, the graph generation techniques could also be a significant tool, generating generic graph patterns without reference to instance-level explanations [198].

10.4 Informative Graph Pooling

Challenges: Existing graph pooling techniques leverage all nodes collectively or seek the most representative nodes for presenting graphs, but neither can ensure a low redundancy between the selected nodes. An informative graph pooling should select the set of nodes within the high representative and low redundancy. Yet, there are no existing criteria to evaluate the redundancy of a set of nodes, and optimizing the node set with two criteria could be a nonconvex problem.

Opportunities: As far as we know, informative graph pooling is first identified in this article. We highlight the informative node set selection for pooling with two criteria, high representative and low redundancy. On the one hand, the criteria of representative nodes could be a score function via neural networks [4, 136] or the policy gradient's feedback. On the other hand, the redundancy of a set of nodes can be evaluated by the similarity of their features or even the graph kernel value between their neighborhoods. The nonconvex optimization problem could be alleviated by reinforcement learning or evolutionary algorithms.

10.5 Graph-level Federated Learning

Challenges: Most state-of-the-art graph-level learning models are data-hungry, especially GL-GNNs. In the industrial scenario, however, the size of the graph dataset sourced from a single institution is limited for training models. There is a practical demand to develop graph-level federated learning, that is, joint training graph-level learning models via diverse institutions. A major obstacle to graph-level federated learning is the divergence of graph structure and feature distribution across graph datasets from various institutions. This incongruity leads to complications in both training and convergence of federated models.

Opportunities: Federated learning has attracted a surge of attention in promoting graph learning, as both are valuable in real-world applications. Xie et al. [199] proposed a graph-level federated learning model, which clusters graph datasets from various institutions and trains an exclusive GNN for each cluster. This could be a promising strategy that divides the multi-sourced data into homogeneous clusters to smooth training. Another strategy could involve enhancing the generalizability of graph-level learning models, which enable models trained via common patterns between graphs sourced from distinct institutions.

10.6 Graph-level Imbalance Learning

Challenges: Graph-level tasks could face an imbalanced state of class labels, such as anomaly detection [200] and long-tail event detection [201], confronting challenges on model learning. Specifically, the 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.

Opportunities: Although imbalanced learning has been a long-standing issue in deep learning, graph-level imbalance learning, especially with deep models, is underexplored. Adopting some simple but effective solutions for imbalanced learning is an easy-implemented way, such as Wang et al. [202] over-sampled graphs in the minority class to relieve imbalance distributions between the majority and minority classes. However, this approach may be criticized for its shortcomings,

such as over-fitting and changing the original distribution of the dataset. We argue that a more reasonable direction is leveraging graph generation techniques to expand the subspace of the minor classes [203–205]. Another possible direction could be to capture the special factors of minority graphs, e.g., key substructures existing in minority graphs but not appearing in the majority. Towards this, strengthening the structural awareness of the current graph-level learning tools is feasible.

10.7 Graph-level Learning on Complex Graphs

Challenges: 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). However, realistic graphs are usually complex graphs, such as heterogeneous graphs involving multiple types of nodes or edges, dynamic graphs whose structure or attributes evolve over time, and so on. Graph-level learning on complex graphs confronts the challenges of tuning graph-level models for the specific type of graph according to its structure and attributes.

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 but also loses partial semantic relationships [206, 207]. Thus, developing the automatic mechanism for exploring comprehensive meta-paths could be a feasible direction. Dynamic graphs could be conceptualized as a collection of static graphs sourcing from a series of temporal scales. One potential strategy for dealing with dynamic graphs is learning from distinct static graphs at each scale, culminating in the fusion of all outcomes [208]. In addition, building a cross-time graph linking all static graphs together represents a promising direction, e.g., tdGraphEmbed [209] adopts a temporal random walk across all time-scale static graphs.

10.8 Graph-level Interaction Learning

Challenges: 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 mixing multiple drugs. Graph-level interaction learning raises the requirement to explore the graph-graph relationships, which is tough, since there are no explicit metrics to compare two graphs regarding a tremendous number of irregular components (e.g., nodes, substructures).

Opportunities: Although this topic has strong practical implications for graph-level learning applications in biochemistry, it is still understudied. A few attempts [104, 210] revamp GL-GNNs for this task, but we argue that GL-GNNs are not the most suitable tool. Graph kernels and attention mechanisms could be the better candidate for this task, due to their advantages in measuring pairwise similarities. In addition, capturing crucial substructures that lead to chemical reactions can be a powerful assistance, which requires the addition of interpretable learning.

10.9 Graph-level Anomaly Detection

Challenges: Anomaly detection aims to identify objects that significantly deviate from the majority of other objects. However, when it comes to the anomalous as entire graphs, the inducement could be node features or substructures in graphs. Generally anomaly detectors mostly only pay attention to outliers in the feature space, but the challenge of graph-level anomaly detectors is identifying outlier points in terms of both node feature and structure.

Opportunities: Graph-level anomaly detection that identifies anomalous graphs in a graph dataset is a research topic of great value application-wise but lacks exploration. Some pioneering studies [211, 212] combine state-of-the-art GL-GNNs with traditional anomaly-detection methods (e.g., one-class classification [213]) to detect anomalous graphs in a graph dataset. This is a feasible way but still needs to solve some derivative issues, e.g., performance flip, and hypersphere collapse [212]. Another possible direction could be designing specific graph convolution layers to highlight anomalous. This is because current graph convolution works like a low-pass filter [125] that smooths the anomalous information in a graph [214].

10.10 Out-of-distribution Generalization

Challenges: **Out-of-distribution (OOD)** learning indicates the scenarios in which the test data own deviation distribution as the training data. 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 test data have a scaffold that never appears in training data, graph-level learning methods models will not perform nearly as well.

Opportunities: The graph-level learning community has recently noticed the OOD generalization issue and desired related research in response. Invariant learning could be a direction for this task, for instance, Wu et al. [215] identified the causal subgraphs that are invariant across different distributions to improve the OOD generalization ability of GL-GNNs. Similarly, Bevilacqua et al. [216] 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. For more details on this topic, we refer readers to the graph OOD learning benchmark [217].

10.11 Brain Network Analytics

Challenges: 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. 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, distinct brain networks have the same ROIs, and node identities and ROIs are one-to-one correspondence [218].

Opportunities: Brain network analytics pertains to a vast of valuable realistic applications, such as distinguishing brains with neurological disorders from normal individuals and identifying those regions of the brain that are the cause of brain disease. We argue that graph-level learning with brain networks requires lightweight models to prevent the model from overfitting on realistic small datasets. In addition, the model should add a module for identifying corresponding nodes between different graphs.

10.12 Multi-graph-level Learning

Challenges: 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. Two significant challenges arise when multiple graphs are bagged as an object: how to glean insights from multi-graph level labels instead of individual instance-level labels and how to filter redundant information across multi-graphs.

Opportunities: To the best of our knowledge, the current multi-graph-level learning algorithms are all subgraph mining methods [219, 220]. However, these endeavors cannot leverage label information to guide the learning process. We argue that deep learning models will revolutionize this task; one possible direction is to build a new GL-GNN framework enabling multiple graph embeddings used to predict a single label. Another promising way could be informative graph pooling, which avails of selecting the node set that contains highly representative and low-redundancy information across multiple graphs.

11 Conclusions

This survey article 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 article, 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. Last, we identified 12 future directions based on currently open issues that would make valuable contributions to this field.

References

[1] Leonhard Euler. 1741. Solutio problematis ad geometriam situs pertinentis. *Comment. Acad. Sci. Imp. Petropol.* (1741), 128–140.
[2] Zhenqin Wu, Bharath Ramsundar, Evan N. Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S. Pappu, Karl Leswing, and Vijay Pande. 2018. MoleculeNet: A benchmark for molecular machine learning. *Chem. Sci.* 9, 2 (2018), 513–530.
[3] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How powerful are graph neural networks? In *ICLR*. 1–17.
[4] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. 2018. An end-to-end deep learning architecture for graph classification. In *AAAI*. 1–8.
[5] Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. 2019. Provably powerful graph networks. In *NeurIPS*. 1–12.
[6] Yu Rong, Yatao Bian, Tingyang Xu, Weiyang Xie, Ying Wei, Wenbing Huang, and Junzhou Huang. 2020. Self-supervised graph transformer on large-scale molecular data. In *NeurIPS*, Vol. 33. 12559–12571.
[7] Aleksandar Bojchevski, Oleksandr Shchur, Daniel Zügner, and Stephan Günnemann. 2018. NetGAN: Generating graphs via random walks. In *ICML*. 610–619.
[8] Jiaxuan You, Rex Ying, Xiang Ren, William Hamilton, and Jure Leskovec. 2018. GraphRNN: Generating realistic graphs with deep auto-regressive models. In *ICML*. 5708–5717.
[9] Frank Harary. 1969. The four color conjecture and other graphical diseases. *Proof Techniques in Graph Theory* 1, 1 (1969), 1–9.
[10] Boris Weisfeiler and Andrei Leman. 1968. The reduction of a graph to canonical form and the algebra which appears therein. *NTI, Series* 2, 9 (1968), 12–16.
[11] Brendan D. McKay and Adolfo Piperno. 2014. Practical graph isomorphism, II. *J. Symb. Comput.* 60 (2014), 94–112.
[12] László Babai. 2016. Graph isomorphism in quasipolynomial time. In *STOC*. 684–697.
[13] Thomas Gärtner, Peter Flach, and Stefan Wrobel. 2003. On graph kernels: Hardness results and efficient alternatives. In *LNAI*. 129–143.
[14] Akihiro Inokuchi, Takashi Washio, and Hiroshi Motoda. 2000. An apriori-based algorithm for mining frequent substructures from graph data. In *ECML-PKDD*. 13–23.

[15] Yuyang Wang, Jianren Wang, Zhonglin Cao, and Amir Barati Farimani. 2022. Molecular contrastive learning of representations via graph neural networks. *Nat. Mach. Intell.* 4, 3 (2022), 279–287.
[16] Jessica Vamathevan, Dominic Clark, Paul Czodrowski, Ian Dunham, Edgardo Ferran, George Lee, Bin Li, Anant Madabhushi, Parantu Shah, Michaela Spitzer, and Shanrong Zhao. 2019. Applications of machine learning in drug discovery and development. *Nat. Rev. Drug Discov* 18, 6 (2019), 463–477.
[17] Tommaso Lanciano, Francesco Bonchi, and Aristides Gionis. 2020. Explainable classification of brain networks via contrast subgraphs. In *KDD*. 3308–3318.
[18] Xuexiong Luo, Jia Wu, Jian Yang, Hongyang Chen, Zhao Li, Hao Peng, and Chuan Zhou. 2024. Knowledge distillation guided interpretable brain subgraph neural networks for brain disorder exploration. *IEEE Trans. Neural Netw. Learn. Syst.* 1, 1 (2024), 1–14.
[19] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. 2020. A comprehensive survey on graph neural networks. *IEEE Trans. Neural Netw. Learn. Syst.* 32, 1 (2020), 4–24.
[20] Ziwei Zhang, Peng Cui, and Wenwu Zhu. 2020. Deep learning on graphs: A survey. *IEEE Trans. Knowl. Data Eng.* 34, 1 (2020), 249–270.
[21] Nils M. Kriege, Fredrik D. Johansson, and Christopher Morris. 2020. A survey on graph kernels. *Appl. Netw. Sci.* 5, 1 (2020), 1–42.
[22] Chuang Liu, Yibing Zhan, Chang Li, Bo Du, Jia Wu, Wenbin Hu, Tongliang Liu, and Dacheng Tao. 2023. Graph pooling for graph neural networks: Progress, challenges, and opportunities. In *IJCAI*.
[23] Daniele Grattarola, Daniele Zambon, Filippo Maria Bianchi, and Cesare Alippi. 2022. Understanding pooling in graph neural networks. *IEEE Trans. Neural Netw. Learn. Syst.* 35, 2 (2022), 2708–2718.
[24] Simon S. Du, Kangcheng Hou, Ruslan Salakhutdinov, Barnabás Póczos, Ruosong Wang, and Keyulu Xu. 2019. Graph neural tangent kernel: Fusing graph neural networks with graph kernels. In *NeurIPS*. 5724–5734.
[25] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. 2017. Neural message passing for quantum chemistry. In *ICML*. 1263–1272.
[26] David Haussler. 1999. Convolution kernels on discrete structures. Citeseer.
[27] Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. 2011. Weisfeiler-Lehman graph kernels. *J. Mach. Learn. Res.* 12, 9 (2011).
[28] Shohei Hido and Hisashi Kashima. 2009. A linear-time graph kernel. In *ICDM*. 179–188.
[29] Christopher Morris, Kristian Kersting, and Petra Mutzel. 2017. Glocalized Weisfeiler-Lehman graph kernels: Global-local feature maps of graphs. In *ICDM*. 327–336.
[30] Marion Neumann, Roman Garnett, Christian Bauckhage, and Kristian Kersting. 2016. Propagation kernels: Efficient graph kernels from propagated information. *Mach. Learn.* 102, 2 (2016), 209–245.
[31] Bastian Rieck, Christian Bock, and Karsten Borgwardt. 2019. A persistent Weisfeiler-Lehman procedure for graph classification. In *ICML*. 5448–5458.
[32] Karsten M. Borgwardt and Hans-Peter Kriegel. 2005. Shortest-path kernels on graphs. In *ICDM*. 1–8.
[33] Giannis Nikolentzos, Polykarpos Meladianos, François Rousseau, Yannis Stavrakas, and Michalis Vazirgiannis. 2017. Shortest-path graph kernels for document similarity. In *EMNLP*. 1890–1900.
[34] Pierre Mahé, Nobuhisa Ueda, Tatsuya Akutsu, Jean-Luc Perret, and Jean-Philippe Vert. 2004. Extensions of marginalized graph kernels. In *ICML*. 70–78.
[35] S. Vichy N. Vishwanathan, Nicol N. Schraudolph, Risi Kondor, and Karsten M. Borgwardt. 2010. Graph kernels. *J. Mach. Learn. Res.* 11 (2010), 1201–1242.
[36] Holger Fröhlich, Jörg K. Wegner, Florian Sieker, and Andreas Zell. 2005. Optimal assignment kernels for attributed molecular graphs. In *ICML*. 225–232.
[37] Deepti Pachauri, Risi Kondor, and Vikas Singh. 2013. Solving the multi-way matching problem by permutation synchronization. In *NeurIPS*. 1860–1868.
[38] Fredrik D. Johansson and Devdatt Dubhashi. 2015. Learning with similarity functions on graphs using matchings of geometric embeddings. In *KDD*. 467–476.
[39] Michele Schiavinato, Andrea Gasparetto, and Andrea Torsello. 2015. Transitive assignment kernels for structural classification. In *SIMBAD*. 146–159.
[40] Nino Shervashidze, S. V. N. Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten Borgwardt. 2009. Efficient graphlet kernels for large graph comparison. In *AISTATS*. 488–495.
[41] Fabrizio Costa and Kurt De Grave. 2010. Fast neighborhood subgraph pairwise distance kernel. In *ICML*. 255–262.
[42] Nils Kriege and Petra Mutzel. 2012. Subgraph matching kernels for attributed graphs. In *ICML*. 291–298.
[43] Michihiro Kuramochi and George Karypis. 2001. Frequent subgraph discovery. In *ICDM*. 313–320.
[44] Xifeng Yan and Jiawei Han. 2002. gSpan: Graph-based substructure pattern mining. In *ICDM*. 721–724.
[45] Xifeng Yan, Hong Cheng, Jiawei Han, and Philip S. Yu. 2008. Mining significant graph patterns by leap search. In *SIGMOD*. 433–444.

[46] Marisa Thoma, Hong Cheng, Arthur Gretton, Jiawei Han, Hans-Peter Kriegel, Alex Smola, Le Song, Philip S. Yu, Xifeng Yan, and Karsten Borgwardt. 2009. Near-optimal supervised feature selection among frequent subgraphs. In *SDM*. 1076–1087.
[47] Xiangnan Kong and Philip S. Yu. 2010. Multi-label feature selection for graph classification. In *ICDM*. 274–283.
[48] Xiangnan Kong and Philip S. Yu. 2010. Semi-supervised feature selection for graph classification. In *KDD*. 793–802.
[49] Yuchen Zhao, Xiangnan Kong, and Philip S. Yu. 2011. Positive and unlabeled learning for graph classification. In *ICDM*. 962–971.
[50] Jia Wu, Zhibin Hong, Shirui Pan, Xingquan Zhu, Zhihua Cai, and Chengqi Zhang. 2014. Multi-graph-view learning for graph classification. In *ICDM*. 590–599.
[51] Saurabh Verma and Zhi-Li Zhang. 2017. Hunt for the unique, stable, sparse and fast feature learning on graphs. In *NeurIPS*. 87–97.
[52] Sergey Ivanov and Evgeny Burnaev. 2018. Anonymous walk embeddings. In *ICML*. 2186–2195.
[53] Chen Cai and Yusu Wang. 2018. A simple yet effective baseline for non-attribute graph classification. *arXiv preprint arXiv:1811.03508* (2018).
[54] Anton Tsitsulin, Marina Munkhoeva, and Bryan Perozzi. 2020. Just SLaQ when you approximate: Accurate spectral distances for web-scale graphs. In *WWW*. 2697–2703.
[55] Xuecheng Liu, Luoyi Fu, and Xinbing Wang. 2021. Bridging the gap between von Neumann graph entropy and structural information: Theory and applications. In *WWW*. 3699–3710.
[56] Saurabh Sawlani, Lingxiao Zhao, and Leman Akoglu. 2021. Fast attributed graph embedding via density of states. In *ICDM*. 559–568.
[57] Robert W. Floyd. 1962. Algorithm 97: Shortest path. *Commun. ACM* 5, 6 (1962), 345–350.
[58] Edsger W. Dijkstra. 1959. A note on two problems in connexion with graphs. *Numer. Math.* 1, 1 (1959), 269–271.
[59] Adam Woźnica, Alexandros Kalousis, and Melanie Hilario. 2010. Adaptive matching based kernels for labelled graphs. In *PAKDD*. 374–385.
[60] Nikil Wale, Ian A. Watson, and George Karypis. 2008. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowl. Inf. Syst.* 14, 3 (2008), 347–375.
[61] Fan R. K. Chung and Fan Chung Graham. 1997. *Spectral Graph Theory*. Vol. 92. American Mathematical Society.
[62] Samuel L. Braunstein, Sibasish Ghosh, and Simone Severini. 2006. The Laplacian of a graph as a density matrix: A basic combinatorial approach to separability of mixed states. *Ann. Comb.* 10, 3 (2006), 291–317.
[63] Pin-Yu Chen, Lingfei Wu, Sijia Liu, and Indika Rajapakse. 2019. Fast incremental von Neumann graph entropy computation: Theory, algorithm, and applications. In *ICML*. 1091–1101.
[64] Annamalai Narayanan, Mahinthan Chandramohan, Lihui Chen, Yang Liu, and Santhoshkumar Saminathan. 2016. subgraph2vec: Learning distributed representations of rooted sub-graphs from large graphs. In *KDD Workshop on Mining and Learning with Graphs*.
[65] Annamalai Narayanan, Mahinthan Chandramohan, Rajasekar Venkatesan, Lihui Chen, Yang Liu, and Shantanu Jaiswal. 2017. graph2vec: Learning distributed representations of graphs. In *KDD Workshop on Mining and Learning with Graphs*.
[66] Dang Nguyen, Wei Luo, Tu Dinh Nguyen, Svetha Venkatesh, and Dinh Phung. 2018. Learning graph representation via frequent subgraphs. In *SDM*. 306–314.
[67] Yujia Li, Richard Zemel, Marc Brockschmidt, and Daniel Tarlow. 2016. Gated graph sequence neural networks. In *ICLR*. 1–20.
[68] John Boaz Lee, Ryan Rossi, and Xiangnan Kong. 2018. Graph classification using structural attention. In *KDD*. 1666–1674.
[69] Xiaohan Zhao, Bo Zong, Ziyu Guan, Kai Zhang, and Wei Zhao. 2018. Substructure assembling network for graph classification. In *AAAI*, Vol. 32.
[70] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. 2016. Learning convolutional neural networks for graphs. In *ICML*. 2014–2023.
[71] James Atwood and Don Towsley. 2016. Diffusion-convolutional neural networks. In *NeurIPS*. 1993–2001.
[72] Martin Simonovsky and Nikos Komodakis. 2017. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *CVPR*. 1–10.
[73] Giannis Nikolentzos, Polykarpos Meladianos, Antoine Jean-Pierre Tixier, Konstantinos Skianis, and Michalis Vazirgiannis. 2018. Kernel graph convolutional neural networks. In *ICANN*. 22–32.
[74] Saurabh Verma and Zhi-Li Zhang. 2018. Graph capsule convolutional neural networks. In *ICML-IJCAL Workshop on Computational Biology*. 1–12.
[75] Zhang Xinyi and Lihui Chen. 2018. Capsule graph neural network. In *ICLR*. 1–16.
[76] Marcelo Daniel Gutierrez Mallea, Peter Meltzer, and Peter J. Bentley. 2019. Capsule neural networks for graph classification using explicit tensorial graph representations. *arXiv preprint arXiv:1902.08399* (2019).

[77] Vijay Prakash Dwivedi and Xavier Bresson. 2020. A generalization of transformer networks to graphs. In *AAAI Workshop on Deep Learning on Graphs: Methods and Applications*. 1–8.
[78] Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. 2021. Do transformers really perform badly for graph representation? In *NeurIPS*, Vol. 34. 28877–28888.
[79] Jinwoo Kim, Dat Nguyen, Seonwoo Min, Sungjun Cho, Moontae Lee, Honglak Lee, and Seunghoon Hong. 2022. Pure transformers are powerful graph learners. In *NeurIPS*, Vol. 35. 14582–14595.
[80] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. In *ICLR*. 1–12.
[81] Pengfei Liu, Xipeng Qiu, and Xuanjing Huang. 2016. Recurrent neural network for text classification with multi-task learning. In *IJCAI*. 2873–2879.
[82] Geoffrey Hinton, Li Deng, Dong Yu, George E. Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N. Sainath, and Brian Kingsbury. 2012. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Process. Mag.* 29, 6 (2012), 82–97.
[83] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2009. The graph neural network model. *IEEE Trans. Neural Netw.* 20, 1 (2009), 61–80.
[84] Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. 2017. Geometric deep learning: Going beyond euclidean data. *IEEE Signal Process. Mag.* 34, 4 (2017), 18–42.
[85] Wanqi Ma, Hong Chen, Wenkang Zhang, Han Huang, Jian Wu, Xu Peng, and Qingqing Sun. 2024. DSYOLO-trash: An attention mechanism-integrated and object tracking algorithm for solid waste detection. *Waste Manag.* 178 (2024), 46–56.
[86] Alessio Micheli. 2009. Neural network for graphs: A contextual constructive approach. *IEEE Trans. Neural Netw.* 20, 3 (2009), 498–511.
[87] Geoffrey E. Hinton, Alex Krizhevsky, and Sida D. Wang. 2011. Transforming auto-encoders. In *ICANN*. 44–51.
[88] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NeurIPS*, Vol. 30. Curran Associates, Inc., 1–11.
[89] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*.
[90] David K. Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P. Adams. 2015. Convolutional networks on graphs for learning molecular fingerprints. In *NeurIPS*. 2224–2232.
[91] Peter W. Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, and Koray Kavukcuoglu. 2016. Interaction networks for learning about objects, relations and physics. In *NeurIPS*. 4502–4510.
[92] Kristof T. Schütt, Farhad Arbabzadah, Stefan Chmiela, Klaus R. Müller, and Alexandre Tkatchenko. 2017. Quantum-chemical insights from deep tensor neural networks. *Nat. Commun.* 8, 1 (2017), 1–8.
[93] Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. 2019. Weisfeiler and Leman go neural: Higher-order graph neural networks. In *AAAI*. 4602–4609.
[94] Ryan Murphy, Balasubramaniam Srinivasan, Vinayak Rao, and Bruno Ribeiro. 2019. Relational pooling for graph representations. In *ICML*. 4663–4673.
[95] Waiss Azizian and Marc Lelarge. 2021. Expressive power of invariant and equivariant graph neural networks. In *ICLR*. 1–39.
[96] Cristian Bodnar, Fabrizio Frasca, Yuguang Wang, Nina Otter, Guido F. Montufar, Pietro Lio, and Michael Bronstein. 2021. Weisfeiler and Lehman go topological: Message passing simplicial networks. In *ICML*. 1026–1037.
[97] Cristian Bodnar, Fabrizio Frasca, Nina Otter, Yuguang Wang, Pietro Lio, Guido F. Montufar, and Michael Bronstein. 2021. Weisfeiler and Lehman go cellular: CW networks. In *NeurIPS*. 2625–2640.
[98] Ralph Abboud, Ismail Ilkan Ceylan, Martin Grohe, and Thomas Lukasiewicz. 2021. The surprising power of graph neural networks with random node initialization. In *IJCAI*. 2112–2118.
[99] Emily Alsentzer, Samuel G. Finlayson, Michelle M. Li, and Marinka Zitnik. 2020. Subgraph neural networks. In *NeurIPS*. 1–13.
[100] Qingyun Sun, Jianxin Li, Hao Peng, Jia Wu, Yuanxing Ning, Philip S. Yu, and Lifang He. 2021. SUGAR: Subgraph neural network with reinforcement pooling and self-supervised mutual information mechanism. In *WWW*. 2081–2091.
[101] Muhan Zhang and Pan Li. 2021. Nested Graph neural networks. In *NeurIPS*. 1–14.
[102] Lingxiao Zhao, Wei Jin, Leman Akoglu, and Neil Shah. 2022. From stars to subgraphs: Uplifting any GNN with local structure awareness. In *ICLR*. 1–22.

[103] Asiri Wijesinghe and Qing Wang. 2022. A new perspective on "how graph neural networks go beyond Weisfeiler-Lehman?" In *ICLR*. 1–23.
[104] Beatrice Bevilacqua, Fabrizio Frasca, Derek Lim, Balasubramaniam Srinivasan, Chen Cai, Gopinath Balamurugan, Michael M. Bronstein, and Haggai Maron. 2022. Equivariant subgraph aggregation networks. In *ICLR*. 1–46.
[105] Giorgos Bouritsas, Fabrizio Frasca, Stefanos P. Zafeiriou, and Michael Bronstein. 2022. Improving graph neural network expressivity via subgraph isomorphism counting. *IEEE Trans. Pattern Anal. Mach. Intell.* 45, 1 (2022), 657–668.
[106] Yinan Huang, Xingang Peng, Jianzhu Ma, and Muhan Zhang. 2023. Boosting the cycle counting power of graph neural networks with I^2-GNNs. In *ICLR*.
[107] Rami Al-Rfou, Bryan Perozzi, and Dustin Zelle. 2019. DDGK: Learning graph representations for deep divergence graph kernels. In *WWW*. 37–48.
[108] Dexiong Chen, Laurent Jacob, and Julien Mairal. 2020. Convolutional kernel networks for graph-structured data. In *ICML*. 1576–1586.
[109] Giannis Nikolentzos and Michalis Vazirgiannis. 2020. Random walk graph neural networks. In *NeurIPS*. 16211–16222.
[110] Qingqing Long, Yilun Jin, Yi Wu, and Guojie Song. 2021. Theoretically improving graph neural networks via anonymous walk graph kernels. In *WWW*. 1204–1214.
[111] Jiarui Feng, Yixin Chen, Fuhai Li, Anindya Sarkar, and Muhan Zhang. 2022. How powerful are k-hop message passing graph neural networks. In *NeurIPS*, Vol. 35. 4776–4790.
[112] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *NeurIPS*. 3837–3845.
[113] Ron Levie, Wei Huang, Lorenzo Bucci, Michael M. Bronstein, and Gitta Kutyniok. 2021. Transferability of spectral graph convolutional neural networks. *J. Mach. Learn. Res.* 22 (2021), 272–331.
[114] Muhammet Balcilar, Pierre Héroux, Benoit Gauzere, Pascal Vasseur, Sébastien Adam, and Paul Honeine. 2021. Breaking the limits of message passing graph neural networks. In *ICML*. 599–608.
[115] Filippo Maria Bianchi, Daniele Grattarola, Lorenzo Livi, and Cesare Alippi. 2021. Graph neural networks with convolutional arma filters. *IEEE Trans. Pattern Anal. Mach. Intell.* 44, 7 (2021), 3496–3507.
[116] Xuebin Zheng, Bingxin Zhou, Junbin Gao, Yuguang Wang, Pietro Lió, Ming Li, and Guido Montufar. 2021. How framelets enhance graph neural networks. In *ICML*. 1–10.
[117] Ladislav Rampášek, Michael Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. 2022. Recipe for a general, powerful, scalable graph transformer. In *NeurIPS*, Vol. 35. 14501–14515.
[118] Keyulu Xu, Mozhi Zhang, Jingling Li, Simon Shaolei Du, Ken-Ichi Kawarabayashi, and Stefanie Jegelka. 2021. How neural networks extrapolate: From feedforward to graph neural networks. In *ICLR*. 1–52.
[119] Vikraman Arvind, Frank Fuhlbrück, Johannes Köbler, and Oleg Verbitsky. 2020. On Weisfeiler-Leman invariance: Subgraph counts and related graph properties. *J. Comput. Syst. Sci.* 113 (2020), 42–59.
[120] Clément Vignac, Andreas Loukas, and Pascal Frossard. 2020. Building powerful and equivariant graph neural networks with structural message-passing. In *NeurIPS*. 14143–14155.
[121] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. 2017. PointNet: Deep learning on point sets for 3D classification and segmentation. In *CVPR*. 652–660.
[122] Arthur Jacot, Clément Hongler, and Franck Gabriel. 2018. Neural tangent kernel: Convergence and generalization in neural networks. In *NeurIPS*. 1–10.
[123] Tao Lei, Wengong Jin, Regina Barzilay, and Tommi Jaakkola. 2017. Deriving neural architectures from sequence and graph kernels. In *ICML*. 2024–2033.
[124] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann Lecun. 2014. Spectral networks and locally connected networks on graphs. In *ICML*. 1–14.
[125] Muhammet Balcilar, Guillaume Renton, Pierre Héroux, Benoit Gaüzère, Sébastien Adam, and Paul Honeine. 2020. Analyzing the expressive power of graph neural networks in a spectral perspective. In *ICLR*.
[126] Deyu Bo, Xiao Wang, Chuan Shi, and Huawei Shen. 2021. Beyond low-frequency information in graph convolutional networks. In *AAAI*. 3950–3957.
[127] Muhammet Balcilar, Guillaume Renton, Pierre Héroux, Benoit Gauzere, Sebastien Adam, and Paul Honeine. 2020. Bridging the gap between spectral and spatial domains in graph neural networks. *arXiv preprint arXiv:2003.11702* (2020).
[128] Bingbing Xu, Huawei Shen, Qi Cao, Yunqi Qiu, and Xueqi Cheng. 2019. Graph wavelet neural network. In *ICLR*. 1–13.
[129] David K. Hammond, Pierre Vandergheynst, and Rémi Gribonval. 2011. Wavelets on graphs via spectral graph theory. *Appl. Comput. Harmon. Anal.* 30, 2 (2011), 129–150.
[130] Xiyuan Wang and Muhan Zhang. 2022. How powerful are spectral graph neural networks. In *ICML*. 23341–23362.
[131] Dexiong Chen, Leslie O'Bray, and Karsten Borgwardt. 2022. Structure-aware transformer for graph representation learning. In *ICML*. PMLR, 3469–3489.

[132] Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. 2020. Principal neighbourhood aggregation for graph nets. In *NeurIPS*. 1–11.
[133] Yufeng Zhang, Xueli Yu, Zeyu Cui, Shu Wu, Zhongzhen Wen, and Liang Wang. 2020. Every document owns its structure: Inductive text classification via graph neural networks. In *ACL*. 334–339.
[134] Zhengyang Wang and Shuiwang Ji. 2020. Second-order pooling for graph neural networks. *IEEE Trans. Pattern Anal. Mach. Intell.* 45, 6 (2020), 1–12.
[135] Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. 2016. Order matters: Sequence to sequence for sets. In *ICLR*. 1–11.
[136] Liang Zhang, Xudong Wang, Hongsheng Li, Guangming Zhu, Peiyi Shen, Ping Li, Xiaoyuan Lu, Syed Afaq Ali Shah, and Mohammed Bennamoun. 2020. Structure-feature based graph self-adaptive pooling. In *WWW*. 3098–3104.
[137] Mikael Henaff, Joan Bruna, and Yann LeCun. 2015. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163* (2015).
[138] Ulrike Von Luxburg. 2007. A tutorial on spectral clustering. *Stat. Comput.* 17, 4 (2007), 395–416.
[139] Yao Ma, Suhang Wang, Charu C. Aggarwal, and Jiliang Tang. 2019. Graph convolutional networks with eigenpooling. In *KDD*. 723–731.
[140] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2014. Spectral networks and deep locally connected networks on graphs. In *ICLR*. 1–14.
[141] Trevor Hastie, Robert Tibshirani, and Jerome H. Friedman. 2009. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Vol. 2. Springer.
[142] Jinheon Baek, Minki Kang, and Sung Ju Hwang. 2021. Accurate learning of graph representations with graph multiset pooling. In *ICLR*. 1–22.
[143] Filippo Maria Bianchi, Daniele Grattarola, and Cesare Alippi. 2020. Spectral clustering with graph neural networks for graph pooling. In *ICML*. 874–883.
[144] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. 2018. Hierarchical graph representation learning with differentiable pooling. In *NeurIPS*. 4805–4815.
[145] Hao Yuan and Shuiwang Ji. 2020. StructPool: Structured graph pooling via conditional random fields. In *ICLR*. 1–12.
[146] Hongyang Gao and Shuiwang Ji. 2019. Graph U-Nets. In *ICML*. 2083–2092.
[147] Cătălina Cangea, Petar Veličković, Nikola Jovanović, Thomas Kipf, and Pietro Liò. 2018. Towards sparse hierarchical graph classifiers. *arXiv preprint arXiv:1811.01287* (2018).
[148] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. 2019. Self-attention graph pooling. In *ICML*. 3734–3743.
[149] Ekagra Ranjan, Soumya Sanyal, and Partha Talukdar. 2020. ASAP: Adaptive structure aware pooling for learning hierarchical graph representations. In *AAAI*. 5470–5477.
[150] Tao Shen, Tianyi Zhou, Guodong Long, Jing Jiang, Shirui Pan, and Chengqi Zhang. 2018. DiSAN: Directional self-attention network for RNN/CNN-free language understanding. In *AAAI*, Vol. 32.
[151] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M. Bronstein. 2017. Geometric deep learning on graphs and manifolds using mixture model CNNs. In *CVPR*. 5115–5124.
[152] Inderjit S. Dhillon, Yuqiang Guan, and Brian Kulis. 2007. Weighted graph cuts without eigenvectors a multilevel approach. *IEEE Trans. Pattern Anal. Mach. Intell.* 29, 11 (2007), 1944–1957.
[153] Filippo Maria Bianchi, Daniele Grattarola, Lorenzo Livi, and Cesare Alippi. 2020. Hierarchical representation learning in graph neural networks with node decimation pooling. *IEEE Trans. Neural Netw. Learn. Syst.* 33, 5 (2020), 2195–2207.
[154] Junran Wu, Shangzhe Li, Jianhao Li, Yicheng Pan, and Ke Xu. 2022. A simple yet effective method for graph classification. In *IJCAI*. 1–7.
[155] Angsheng Li and Yicheng Pan. 2016. Structural information and dynamical complexity of networks. *IEEE Trans. Inf. Theor.* 62, 6 (2016), 3290–3339.
[156] Junran Wu, Xueyuan Chen, Ke Xu, and Shangzhe Li. 2022. Structural entropy guided graph hierarchical pooling. In *ICML*. 24017–24030.
[157] Runze Yang, Hao Peng, Chunyang Liu, and Angsheng Li. 2024. Incremental measurement of structural entropy for dynamic graphs. *Artif. Intell.* 334 (2024), 104175.
[158] Li Sun, Zhenhao Huang, Hao Peng, YuJie Wang, Chunyang Liu, and Philip S. Yu. 2024. LSEnet: Lorentz structural entropy neural network for deep graph clustering. In *ICML*.
[159] Frederik Diehl. 2019. Edge contraction pooling for graph neural networks. *arXiv preprint arXiv:1905.10990* (2019).
[160] Christopher Morris, Nils M. Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. 2020. TUDataset: A collection of benchmark datasets for learning with graphs. *arXiv preprint arXiv:2007.08663* (2020).
[161] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open graph benchmark: Datasets for machine learning on graphs. In *NeurIPS*. 22118–22133.

[162] Scott Freitas, Yuxiao Dong, Joshua Neil, and Duen Horng Chau. 2021. A large-scale database for graph representation learning. In *NeurIPS Track Datasets and Benchmarks*. 1–13.
[163] Vijay Prakash Dwivedi, Chaitanya K. Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. 2020. Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982* (2020).
[164] Sergei Ivanov, Sergei Sviridov, and Evgeny Burnaev. 2019. Understanding isomorphism bias in graph data sets. *arXiv preprint arXiv:1910.12091* (2019).
[165] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 2015. 3D ShapeNets: A deep representation for volumetric shapes. In *CVPR*. 1912–1920.
[166] Karsten M. Borgwardt, Cheng Soon Ong, Stefan Schönauer, S. V. N. Vishwanathan, Alex J. Smola, and Hans-Peter Kriegel. 2005. Protein function prediction via graph kernels. *Bioinformatics* 21, 1 (2005), 47–56.
[167] Paul D. Dobson and Andrew J. Doig. 2003. Distinguishing enzyme structures from non-enzymes without alignments. *J. Mol. Biol.* 330, 4 (2003), 771–783.
[168] Govindan Subramanian, Bharath Ramsundar, Vijay Pande, and Rajiah Aldrin Denny. 2016. Computational modeling of β-secretase 1 (BACE-1) inhibitors using ligand based approaches. *J. Chem. Inf. Model* 56, 10 (2016), 1936–1949.
[169] Sebastian G. Rohrer and Knut Baumann. 2009. Maximum unbiased validation (MUV) data sets for virtual screening based on PubChem bioactivity data. *J. Chem. Inf. Model* 49, 2 (2009), 169–184.
[170] Marinka Zitnik, Rok Sosič, Marcus W. Feldman, and Jure Leskovec. 2019. Evolution of resilience in protein interactomes across the tree of life. *Proc. Natl. Acad. Sci. U. S. A.* 116, 10 (2019), 4426–4433.
[171] Han Altae-Tran, Bharath Ramsundar, Aneesh S. Pappu, and Vijay Pande. 2017. Low data drug discovery with one-shot learning. *ACS Cent. Sci.* 3, 4 (2017), 283–293.
[172] Kaitlyn M. Gayvert, Neel S. Madhukar, and Olivier Elemento. 2016. A data-driven approach to predicting successes and failures of clinical trials. *Cell Chem. Biol.* 23, 10 (2016), 1294–1301.
[173] Ines Filipa Martins, Ana L. Teixeira, Luis Pinheiro, and Andre O. Falcao. 2012. A Bayesian approach to in silico blood-brain barrier penetration modeling. *J. Chem. Inf. Model* 52, 6 (2012), 1686–1697.
[174] TD Challenge. 2014. Tox21 data challenge 2014. Retrieved from: https://tripod.nih.gov/tox/challenge
[175] Ann M. Richard, Richard S. Judson, Keith A. Houck, Christopher M. Grulke, Patra Volarath, Inthirany Thillainadarajah, Chihae Yang, James F. Rathman, Matthew Thomas Martin, John F. Wambaugh, Thomas B. Knudsen, Jayaram Kancherla, Kamel Mansouri, Grace Y. Patlewicz, Antony John Williams, Stephen B. Little, Kevin M. Crofton, and Russell S. Thomas. 2016. ToxCast chemical landscape: paving the road to 21st century toxicology. *Chem. Res. Toxicol.* 29, 8 (2016), 1225–1251.
[176] David L. Mobley and J. Peter Guthrie. 2014. FreeSolv: A database of experimental and calculated hydration free energies, with input files. *J. Comput.-aid. Molec. Des.* 28, 7 (2014), 711–720.
[177] John S. Delaney. 2004. ESOL: Estimating aqueous solubility directly from molecular structure. *J. Chem. Inf. Comput.* 44, 3 (2004), 1000–1005.
[178] M. Wenlock and N. Tomkinson. 2015. Experimental in vitro DMPK and physicochemical data on a set of publicly disclosed compounds.
[179] Murat Cihan Sorkun, Abhishek Khetan, and Süleyman Er. 2019. AqSolDB, a curated reference set of aqueous solubility and 2D descriptors for a diverse set of compounds. *Sci. Data* 6, 1 (2019), 1–8.
[180] John J. Irwin, Teague Sterling, Michael M. Mysinger, Erin S. Bolstad, and Ryan G. Coleman. 2012. ZINC: A free tool to discover chemistry for biology. *J. Chem. Inf. Model* 52, 7 (2012), 1757–1768.
[181] Pinar Yanardag and S. V. N. Vishwanathan. 2015. Deep graph kernels. In *KDD*. 1365–1374.
[182] Alex Krizhevsky and Geoffrey Hinton. 2009. Learning multiple layers of features from tiny images. Citeseer.
[183] Yann LeCun. 1998. The MNIST database of handwritten digits. Retrieved from: http://yann.lecun.com/exdb/mnist/
[184] Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. 2019. CodeSearchNet challenge: Evaluating the state of semantic code search. *arXiv preprint arXiv:1909.09436* (2019).
[185] Daniel Ruffinelli, Samuel Broscheit, and Rainer Gemulla. 2019. You can teach an old dog new tricks! On training knowledge graph embeddings. In *ICLR*. 1–20.
[186] Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. 2020. A fair comparison of graph neural networks for graph classification. In *ICLR*. 1–16.
[187] Yanqiao Zhu, Yichen Xu, Qiang Liu, and Shu Wu. 2021. An empirical study of graph contrastive learning. In *NeurIPS Track Datasets and Benchmarks*. 1–25.
[188] Jia Wu, Zhibin Hong, Shirui Pan, Xingquan Zhu, Zhihua Cai, and Chengqi Zhang. 2016. Multi-graph-view subgraph mining for graph classification. *Knowl. Inf. Syst.* 48, 1 (2016), 29–54.
[189] Jia Wu, Shirui Pan, Xingquan Zhu, Chengqi Zhang, and Philip S. Yu. 2017. Multiple structure-view learning for graph classification. *IEEE Trans. Neural Netw. Learn. Syst.* 29, 7 (2017), 3236–3251.
[190] Zhenyu Yang, Ge Zhang, Jia Wu, Jian Yang, Quan Z. Sheng, Hao Peng, Angsheng Li, Shan Xue, and Jianlin Su. 2023. Minimum entropy principle guided graph neural networks. In *WSDM*. 114–122.

[191] Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. 2021. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478* (2021).
[192] Vıctor Garcia Satorras, Emiel Hoogeboom, and Max Welling. 2021. E(n) equivariant graph neural networks. In *ICML*. 9323–9332.
[193] Johannes Gasteiger, Florian Becker, and Stephan Günnemann. 2021. GemNet: Universal directional graph neural networks for molecules. In *NeurIPS*. 1–13.
[194] Moritz Thürlemann and Sereina Riniker. 2022. Anisotropic message passing: Graph neural networks with directional and long-range interactions. In *ICLR*.
[195] Dongsheng Luo, Wei Cheng, Dongkuan Xu, Wenchao Yu, Bo Zong, Haifeng Chen, and Xiang Zhang. 2020. Parameterized explainer for graph neural network. In *NeurIPS*. 19620–19631.
[196] Siqi Miao, Mia Liu, and Pan Li. 2022. Interpretable and generalizable graph learning via stochastic attention mechanism. In *ICML*. 1–20.
[197] Steve Azzolin, Antonio Longa, Pietro Barbiero, Pietro Lio, and Andrea Passerini. 2023. Global explainability of GNNs via logic combination of learned concepts. In *ICLR*. 1–16.
[198] Hao Yuan, Jiliang Tang, Xia Hu, and Shuiwang Ji. 2020. XGNN: Towards model-level explanations of graph neural networks. In *KDD*. 430–438.
[199] Han Xie, Jing Ma, Li Xiong, and Carl Yang. 2021. Federated graph classification over non-IID graphs. In *NeurIPS*. 18839–18852.
[200] Ge Zhang, Zhenyu Yang, Jia Wu, Jian Yang, Xue Shan, Hao Peng, Jianlin Su, Chuan Zhou, Quan Z. Sheng, Leman Akoglu, and Charu C. Aggarwal. 2022. Dual-discriminative graph neural network for imbalanced graph-level anomaly detection. In *NeurIPS*. 1–12.
[201] Puneet Agarwal, Rajgopal Vaithiyanathan, Saurabh Sharma, and Gautam Shroff. 2012. Catching the long-tail: Extracting local news events from Twitter. In *AAAI*. 379–382.
[202] Yu Wang, Yuying Zhao, Neil Shah, and Tyler Derr. 2021. Imbalanced graph classification via graph-of-graph neural networks. *arXiv preprint arXiv:2112.00238* (2021).
[203] Wen-Zhi Li, Chang-Dong Wang, Hui Xiong, and Jian-Huang Lai. 2023. GraphSHA: Synthesizing harder samples for class-imbalanced node classification. In *KDD*. 1328–1340.
[204] Xiaoxiao Ma, Ruikun Li, Fanzhen Liu, Kaize Ding, Jian Yang, and Jia Wu. 2024. Graph anomaly detection with few labels: A data-centric approach. In *KDD*. 1–12.
[205] Fanzhen Liu, Xiaoxiao Ma, Jia Wu, Jian Yang, Shan Xue, Amin Beheshti, Chuan Zhou, Hao Peng, Quan Z. Sheng, and Charu C. Aggarwal. 2022. DAGAD: Data augmentation for graph anomaly detection. In *ICDM*. 259–268.
[206] Rana Hussein, Dingqi Yang, and Philippe Cudré-Mauroux. 2018. Are meta-paths necessary? Revisiting heterogeneous graph embeddings. In *CIKM*. 437–446.
[207] Yaming Yang, Ziyu Guan, Jianxin Li, Wei Zhao, Jiangtao Cui, and Quan Wang. 2021. Interpretable and efficient heterogeneous graph convolutional network. *IEEE Trans. Knowl. Data Eng.* (2021).
[208] Yunling Ma, Qianqian Wang, Liang Cao, Long Li, Chaojun Zhang, Lishan Qiao, and Mingxia Liu. 2023. Multi-scale dynamic graph learning for brain disorder detection with functional MRI. *IEEE Trans. Neural Syst. Rehabil. Eng.* 31, 1 (2023).
[209] Moran Beladev, Lior Rokach, Gilad Katz, Ido Guy, and Kira Radinsky. 2020. tdGraphEmbed: Temporal dynamic graph-level embedding. In *CIKM*. 55–64.
[210] Hanchen Wang, Defu Lian, Ying Zhang, Lu Qin, and Xuemin Lin. 2021. GoGNN: Graph of graphs neural network for predicting structured entity interactions. In *IJCAI*. 1–7.
[211] Chen Qiu, Marius Kloft, Stephan Mandt, and Maja Rudolph. 2022. Raising the bar in graph-level anomaly detection. In *IJCAI*. 1–8.
[212] Lingxiao Zhao and Leman Akoglu. 2021. On using classification datasets to evaluate graph outlier detection: Peculiar observations and new insights. *Big Data* 11, 3 (2021), 1–30.
[213] Lukas Ruff, Robert Vandermeulen, Nico Goernitz, Lucas Deecke, Shoaib Ahmed Siddiqui, Alexander Binder, Emmanuel Müller, and Marius Kloft. 2018. Deep one-class classification. In *ICML*. 4393–4402.
[214] Jianheng Tang, Jiajin Li, Ziqi Gao, and Jia Li. 2022. Rethinking graph neural networks for anomaly detection. In *ICML*. 21076–21089.
[215] Ying-Xin Wu, Xiang Wang, An Zhang, Xiangnan He, and Tat-Seng Chua. 2022. Discovering invariant rationales for graph neural networks. In *ICLR*. 1–22.
[216] Beatrice Bevilacqua, Yangze Zhou, and Bruno Ribeiro. 2021. Size-invariant graph representations for graph classification extrapolations. In *ICML*. 1–14.
[217] Shurui Gui, Xiner Li, Limei Wang, and Shuiwang Ji. 2022. GOOD: A graph out-of-distribution benchmark. *arXiv preprint arXiv:2206.08452* (2022).
[218] Olaf Sporns. 2022. Graph theory methods: Applications in brain networks. *Dialogues Clin. Neurosci.* 20, 2 (2022).

[219] Jia Wu, Shirui Pan, Xingquan Zhu, and Zhihua Cai. 2014. Boosting for multi-graph classification. *IEEE Trans. Cybern.* 45, 3 (2014), 416–429.

[220] Jia Wu, Shirui Pan, Xingquan Zhu, Chengqi Zhang, and Xindong Wu. 2018. Multi-instance learning with discriminative bag mapping. *IEEE Trans. Knowl. Data Eng.* 30, 6 (2018), 1065–1080.

Received 15 May 2023; revised 7 June 2024; accepted 22 August 2024