Graph Structure and Isomorphism Learning: A Survey

Tuyen Ho Thi Thanh[1,2,3]

[1] Faculty of Information Technology, University of Science, Ho Chi Minh City, Vietnam
[2] Vietnam National University, Ho Chi Minh City, Vietnam
[3] University of Economics, Ho Chi Minh City, Vietnam

Correspondence: Tuyen Ho Thi Thanh (tuyenhtt@ueh.edu.vn)
Communication: received xx, revised xx, accepted xx
Digital Object Identifier: 10.32913/mic-ict-research.v2022.n1.1028

Abstract: With the great success of artificial intelligence in recent years, graph learning is gaining attention from both academia and industry [1, 2]. The power of graph data is its capacity to represent numerous complicated structures in a broad spectrum of application domains including protein networks, social networks, food webs, molecular structures, knowledge graphs, sentence dependency trees, and scene graphs of images. However, designing an effective graph learning architecture on arbitrary graphs is still an ongoing research topic because of two challenges of learning complex topological structures of graphs and their nature of isomorphism. In this work, we aim to summarize and discuss the latest methods in graph learning, with special attention to two aspects of structure learning and permutation invariance learning. The survey starts by reviewing basic concepts on graph theory and graph signal processing. Next, we provide systematic categorization of graph learning methods to address two aspects above respectively. Finally, we conclude our paper with discussions and open issues in research and practice.

Keywords: *Graph learning, graph structure learning, graph isomorphism, permutation invariance, graph neural networks.*

I. INTRODUCTION

Graphs are powerful data structures which can be considered as *a general language for describing and modeling complex systems* [3] in practice. Unlike many other structures such as images, texts and audios, which only represent the information of entities, graphs are able to capture both entities and their interactions via nodes and edges of graphs. Social networks are typical examples of graphs whose nodes are users and edges show the existence of the relationships between two users. As a consequence of their power, graphs not only play an important role in computer science but also are ubiquitous in related fields (e.g., physics, biology, chemistry) with a wide range of applications: classifying proteins in biological interaction networks, predicting the influence of an individual in social networks and discovering new drug molecules.

Basically, a graph consists of two information types: a feature matrix, denoted by $\boldsymbol{X}$, encoding entities and their properties; and a structural matrix $\mathcal{A}$ (such as adjacency matrix $\boldsymbol{A}$, Laplacian matrix $\boldsymbol{L}$ and its variants), describing the relationship between entities. Between such two types, the structural information is more crucial since object-object interaction representation is a special characteristic of graph data and different graphs are distinguishable via their topological configurations. As a result, learning graph structure becomes a central problem in graph learning. However, converting complicated raw structure information of graphs into compact fixed-size vectors in a continuous embedding space is non-trivial because of two challenges: 1) graph sizes vary dramatically in practice from several units and interactions in molecular networks to millions of objects and connections in social networks, hindering the invention of effective algorithms to extract compact representations but not losing important information; and 2) graph isomorphism, where two graphs look different but are actually the same (Fig 1), causing the requirement of permutation invariance in designing graph learning frameworks. The success of handling these challenges is the key to proceed graph learning field.

To tackle these challenges, tremendous approaches have been proposed in graph learning, ranging from graph kernel methods to graph neural networks. In this survey, we provide an overview of recent advanced learning techniques on graphs in a comprehensive manner. Our focus is effective methods that can either learn graph structures or deal with graph isomorphism.

There exists several previous comprehensive reviews related to our survey. [5] and [6] provide a thorough overview of *geometric deep learning*, a broader area, which attempts

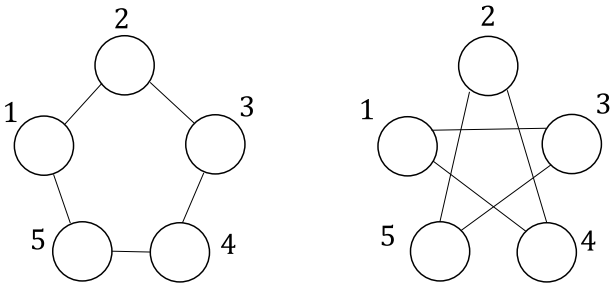

Figure 1. Two identical graphs with different looking structures (adopted from [4]).

to extend machine learning techniques for non-Euclidean data including graphs and manifolds. The early work encoding the relationships of *nodes as embedding vectors* in continuous space refers to node/graph embedding methods with detailed summary in [7, 8]. For *graph learning*, some notable surveys are [1, 9]. Mital Kinderkhedia [9] reviews over existing graph learning methods and presents them into kernel methods and graph neural networks. Meanwhile, Feng Xia et al. [1] systematically categorize the field into four approaches of graph signal processing, matrix factorization, random walks and graph neural networks. With the success of deep learning in images, there have been many methods employing deep learning techniques on graphs (aka *graph neural networks*) in recent years and the most up-to-date surveys [2, 10–12] mainly focus on this subfield of graph learning. Zhou Jie et al. [10] introduce the design pipeline of graph neural networks and provide a review on different implementations for each module in the pipeline. [2] and [11] are based on network architectures to divide graph neural network methods into groups of graph recurrent neural networks, graph convolution networks, graph autoencoders, graph reinforcement learning and graph adversarial networks. There have also been several surveys paying attention to other subfields or specific topics of graph learning such as *kernel methods* [13, 14] or constructing a unified framework for *graph convolutional networks* [12]. Although, many reviews on graph learning have been emerged, they mainly provide *technical views* with taxonomies of algorithm designs and training strategies rather than focusing on the *problem-driven aspect* as our work, i.e., solving two key problems of learning the topology and isomorphism properties of graphs. To the best of our knowledge, there is little effort to systematically summarize the field in these directions. One survey closely related our work is [15] but it only examines a limited number of works and the paper is equivalent to our Subsecs. III.2 and III.3.

In this paper, we introduce a different overview of recent advancements towards solving essential problems in graph learning. To summarize, our contributions are as follows:

- We introduce two new taxonomies of graph learning, corresponding to graph structure and permutation invariance learning. For the *graph structure learning*, existing methods are categorized into six main approaches of *graph coloring-based methods, matrix factorization, node embedding, aggregation operators* and *motif-based methods* while, to solve the latter of *graph isomorphism*, recent works are focusing on four directions of *aggregation function, ordering, histogram and permutation sampling*.
- Comprehensive survey: we provide a detailed review on numerous methods in the literature with descriptions and discussions on their advantages and disadvantages.
- Finally, we discuss on the limitations of existing methods and propose open problems for future research.

The rest of the paper is structured as follows. Sec. II presents notations, basic definitions and concepts in graph learning research, brief introduction of graph signal processing, which is the backbone of spectral graph learning methods, and Weisfeiler-Lehman test, a baseline for graph isomorphism test. We start to introduce graph structure learning and its taxonomy in Sec. III. Sec. IV outlines the approach in graph isomorphism learning and systematically summarizes different existing methods. Finally, we conclude the survey with a discussion on open challenges and potential future directions in graph learning in Sec. V and VI.

II. BACKGROUND

1. Notations

In this paper, we use bold uppercase characters to denote matrices and bold lowercase ones for vectors. For an arbitrary matrix $\boldsymbol{M}$, we use the notations of $\boldsymbol{M}(i,j)$, $\boldsymbol{M}(i,\cdot)$ and $\boldsymbol{M}(\cdot,j)$ to present a matrix element, the i^{th} row and the j^{th} column of $\boldsymbol{M}$ respectively. Table II.1 lists our commonly used notations. Unless particularly specified, all notations are following this table. Table II.1 also shows the abbreviations of methods mentioned in the next sections.

2. Graph

An undirected graph $G = (V, E)$ of N_v vertices/nodes and N_e edges/links consists of two sets: a set of vertices $V = \{i | 1 \le i \le N_v\}$ and a set of edges $E = \left\{e_{i,j} = (i,j) \,|1 \le i, j \le N_v\right\}$ [11]. Each vertex is associated with a N_d-dimensional feature vector $\boldsymbol{x}_i \in \mathbb{R}^{1\times N_d}$ and all vertices form a feature matrix $\boldsymbol{X} = \left[\boldsymbol{x}_1, \boldsymbol{x}_2, .., \boldsymbol{x}_{N_v}\right]^{\mathsf{T}}$. Basically, a graph G can be described by two components: a

Table I
NOTATIONS AND ABBREVIATIONS

Notations	Descriptions	Abbreviations	Descriptions
$\boldsymbol{G}$	Graph	1-WL	1-dimensional Weisfeiler-Lehman
$\boldsymbol{V}$	The set of nodes	CNN	Convolutional Neural Network
$\boldsymbol{E}$	The set of edges	LSTM	Long-Sort Term Memory
$\boldsymbol{A}$	Adjacency matrix	MLP	Multi-layer Perceptron
$\boldsymbol{D}$	Degree matrix	GAE	Graph Auto-encoder
$\boldsymbol{L}$	Laplacian matrix	GAT	Graph Attention Network
$\tilde{\boldsymbol{L}}$	Normalized Laplacian matrix	GCN	Graph Convolutional Network
$\boldsymbol{H}$	Hidden layer	GNN	Graph Neural Network
N_v	The number of nodes	GraphSAGE	Graph SAmple and aggreGatE
N_e	The number of edges	NMF	Non-negative Matrix Factorization
N_l	The number of network layers	RNN	Recurrent Neural Network
$\boldsymbol{X}$	Feature matrix		
$\mathcal{A}$	Structural matrix		
$\mathcal{N}(\cdot)$	Neighborhood of a node		
$\boldsymbol{s}$	Graph signal vector		
$\boldsymbol{P}$	Permutation matrix		
$\mathbb{R}_{\geq 0}$	Non-negative real numbers		
$\|\cdot\|_F$	Frobenius norm		
$\simeq$	Isomorphism		

feature matrix $\boldsymbol{X}$ and a graph structure matrix $\mathcal{A}$. Between them, graph structure is most important but difficult to obtain an effective representation form. In the following, we summarize some typical structural matrices for undirected graphs in literature.

- **Adjacency matrix**: An adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{N_v \times N_v}$ of an undirected graph is a symmetric matrix whose element at the i^{th} row and j^{th} column, denoted $\boldsymbol{A}(i,j)$, is equal to 1 if there exists an edge between two vertices i and j, otherwise $\boldsymbol{A}(i,j) = 0$.

$$\boldsymbol{A}(i,j) = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$$

- **Degree matrix:** A degree matrix $\boldsymbol{D} = \text{diag}\left(d_1, d_2, .., d_{N_v}\right) \in \mathbb{R}^{N_v \times N_v}$ is a diagonal matrix whose the i^{th} element on the diagonal $d_i = |\{j|\,(i,j) \in E\}|$ is the degree at the vertex i or the number of edges containing that vertex.

$$\boldsymbol{D}(i,j) = \begin{cases} d_i & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$

- **Laplacian matrix**: This matrix is defined as $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{A}$. It is symmetric and diagonal and its elements are from degree matrix whilst off-diagonal elements are -1 if connected.

$$\boldsymbol{L}(i,j) = \begin{cases} -1 & \text{if } (i,j) \in E \\ d_i & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

- **Normalized Laplacian matrix**: The matrix $\boldsymbol{L}$ can be normalized into $[-1, 1]$ by dividing its elements by corresponding geometric mean as:

$$\tilde{\boldsymbol{L}}(i,j) = \begin{cases} -1/\sqrt{d_i d_j} & \text{if } (i,j) \in E \text{ vÃ } i \neq j \\ 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

or rewrite it in a matrix multiplication form:

$$\begin{aligned} \tilde{\boldsymbol{L}} &= \boldsymbol{D}^{-1/2}\boldsymbol{L}\boldsymbol{D}^{-1/2} \\ &= \boldsymbol{D}^{-1/2}(\boldsymbol{D} - \boldsymbol{A})\boldsymbol{D}^{-1/2} = \mathbf{I} - \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2} \end{aligned}$$

- **Positive point-wise mutual information (PPMI)** PPMI is a probabilistic co-occurrence matrix, $\mathcal{A}_{\text{P}} \in \mathbb{R}^{N_v \times N_v}$ originated from information theory. In graph learning, PPMI is applied to encode the semantic information of entire graphs [16–18] and it is estimated via two steps: 1) compute the frequency of co-occurrence of nodes over graphs and store these values in a frequency matrix $\boldsymbol{F}$, e.g., an entry $\boldsymbol{F}(i,j)$ counts the number of random walks between two nodes i and j within a predefined window [16]; and then 2) estimate the probability $\mathcal{A}_{\text{P}}(i,j)$ that the node i occurs within a window around the node j:

$$\mathcal{A}_{\text{P}}(i,j) = \max\left(\log\left(\frac{\boldsymbol{F}(i,j)}{\left(\sum_i \boldsymbol{F}(i,j)\right)\left(\sum_j \boldsymbol{F}(i,j)\right)}\right), 0\right)$$

In addition to describe the graph structural information (vertices and their connections), the aforementioned matrices (except for PPMI matrix) can be

used as graph operators. For example, multiplying an adjacency matrix $\boldsymbol{A}$ by a graph signal vector $\boldsymbol{s} = \left[s_1, s_2, .., s_{N_v}\right] \in \mathbb{R}^{1\times N_v}$, where s_i is an observed signal/feature at the i^{th} vertex, results in a new signal $\boldsymbol{As} \in \mathbb{R}^{1\times N_v}$ whose entry is the sum of all signals of vertices j connecting to i. Meanwhile, $\boldsymbol{Ds} \in \mathbb{R}^{1\times N_v}$ is multiplying s_i by the corresponding degree value d_i at i and $\boldsymbol{Ls} \in \mathbb{R}^{1\times N_v}$ (or normalized version $\tilde{\boldsymbol{L}}\boldsymbol{s} \in \mathbb{R}^{1\times N_v}$) expresses the sum of difference between signal at i and its neighbor's signals.

3. Graph signal processing

From the perspective of graph signal processing, $\boldsymbol{A}$, $\boldsymbol{D}$,$\boldsymbol{L}$ and $\tilde{\boldsymbol{L}}$ are considered as graph representations in spatial domain. Another way of graph analysis is to convert them into frequency domain via graph Fourier transforms. The transformed graph signals show some interesting and meaningful information, e.g., connection strength between connected components, which is not obviously exposed in spatial domain, and therefore, graph signal processing is a powerful tool to learn the representation of graphs. This part provides the basic concepts in graph signal processing as an underlying theory for the approach of spectral graph learning methods.

Spectral graph analysis: Given a graph G and its normalized Laplacian matrix $\tilde{\boldsymbol{L}}$, spectral graph analysis [19] finds the spectral characteristics of G by eigen-decomposing $\tilde{\boldsymbol{L}}$ into eigen-values, described by a diagonal matrix $\boldsymbol{\Lambda}$, and a matrix $\boldsymbol{U}$ of eigen-vectors. These matrices own some important properties and therefore are considered as indicators of graph topology, for example, the number of zero eigen-values is equal to the number of connected components of G.

Graph Fourier transform: Fourier transform implies to analyze an input signal/voice/image/graph as a combination of basic components, e.g., wavelet, graphlet. Similarly, Fourier transform on graph converts a signal vector $\boldsymbol{s}$ into spectral domain encoded by eigen-vectors $\boldsymbol{U}$ of $\tilde{\boldsymbol{L}}$. Particularly, if $\tilde{\boldsymbol{L}} = \boldsymbol{U\Lambda U}^{\mathsf{T}}$ then Fourier transform of $\boldsymbol{s}$ is $\tilde{\boldsymbol{s}} = \mathcal{F}(\boldsymbol{s}) = \boldsymbol{U}^{\mathsf{T}}\boldsymbol{s}$, where $\tilde{\boldsymbol{s}}$ is the spectral representation of the input signal $\boldsymbol{s}$. The inverse graph Fourier transform projecting $\tilde{\boldsymbol{s}}$ back to the input space is given as: $\tilde{\tilde{\boldsymbol{s}}} = \mathcal{F}^{-1}(\tilde{\boldsymbol{s}}) = \boldsymbol{U}\tilde{\boldsymbol{s}} = \boldsymbol{s}$.

Graph convolution: A convolution between a signal vector $\boldsymbol{s}$ and a filter $\boldsymbol{g} \in \mathbb{R}^{N_v}$ is defined as:

$$\begin{aligned} \boldsymbol{s}*\boldsymbol{g} &= \mathcal{F}^{-1}\left(\mathcal{F}(\boldsymbol{s}) \odot \mathcal{F}(\boldsymbol{g})\right) \\ &= \boldsymbol{U}\left(\boldsymbol{U}^{\mathsf{T}}\boldsymbol{s} \odot \boldsymbol{U}^{\mathsf{T}}\boldsymbol{g}\right) = \boldsymbol{U}\left(\boldsymbol{U}^{\mathsf{T}}\boldsymbol{g} \odot \boldsymbol{U}^{\mathsf{T}}\boldsymbol{s}\right) \end{aligned}$$

where $*$ is convolution operator and $\odot$ is Hadamard product. Let $\boldsymbol{g}_\theta$ be diag$\left(\boldsymbol{U}^{\mathsf{T}}\boldsymbol{g}\right)$, resulting in $\boldsymbol{U}^{\mathsf{T}}\boldsymbol{g} \odot \boldsymbol{U}^{\mathsf{T}}\boldsymbol{s} =$ diag$\left(\boldsymbol{U}^{\mathsf{T}}\boldsymbol{g}\right) \odot \boldsymbol{U}^{\mathsf{T}}\boldsymbol{s}$ and the convolution can be simplified as $\boldsymbol{s}*\boldsymbol{g}_\theta = \boldsymbol{U}\boldsymbol{g}_\theta\boldsymbol{U}^{\mathsf{T}}\boldsymbol{s}$. Basically, spectral graph learning methods can be distinguished based on different filters $\boldsymbol{g}_\theta$.

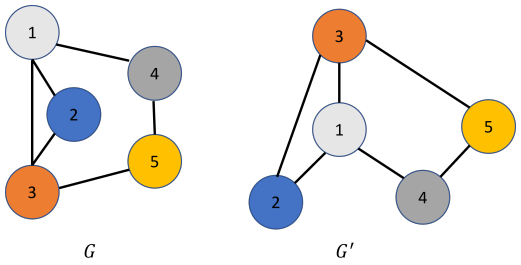

Figure 2. An example of two isomorphic graphs. The corresponding vertices share the same color (adopted from [21]).

4. Graph isomorphism

Two graphs $G = (V, E)$ and $G' = (V', E')$ are *isomorphic* $G \simeq G'$ if and only if there exists a bijection preserving the adjacency relationship of vertices. Formally, a mapping $f : V \rightarrow V'$ satisfies $\forall (u, v) \in E$ if and only if $(f(u), f(v)) \in E'$. The mapping f is called *isomorphism*. If G and $G'$are identical then $f : V \rightarrow V$ are a bijection from V to V and f is an automorphism. Fig. 2 illustrates two isomorphic graphs. Verifying whether two graphs are isomorphic is still an open problem in computer science. The recent results show that this is NP but has not been known to be NP-complete or be solved in polynomial time [20].

One of the most widely-used and effective algorithm to test isomorphism in polynomial time is Weisfeiler-Lehman test. Its most well-known is the 1-dimensional version (1-WL) [22]. Suppose that G and G' are two graphs with the size of N_v, 1-WL algorithm employs an iterative procedure (up to N_v iterations). For each iteration, 1-WL goes through all vertices of two graphs and reassigns their labels/refines their colors based on their current labels/colors and ones of their neighbors.

Let $c_{i,j}$ be the color of the j^{th} vertex of G at the i^{th} iteration and $\mathcal{N}_G(j)$ is the neighborhood of the node j of G. Colors of vertices can be sortable integers. Two nodes with the same/different color value indicate the same/different local structures. We use a multi-set $S_{i,j}$ to describe colors of that node and its neighbors at the i^{th} iteration. A multi-set is an extension of set but accepts replicated elements. 1-WL algorithm is summarized in Alg. 1.

1) Set color value to be 1 for all vertices of G and G' (lines 1-3).
2) For each iteration, enumerate all vertices of two graphs and update their colors (line 5).

Algorithm 1: 1-WL

1 **Inputs**: $G = \{V, E\}, G' = \{V', E'\}, |V| = |V'|, N_{\text{WL}}$
2 **Outputs**: True/False
3 **begin**
4 **for** $m \leftarrow 1, \ldots, N_v$ **do**
5 $c_{0,i} \leftarrow 1$;
6 $c'_{0,i} \leftarrow 1$;
7 **end**
8 **for** $i \leftarrow 1, \ldots, N_{WL}$ **do**
9 **for** $j \leftarrow 1, \ldots, N_v$ **do**
10 $S_{i,j} \leftarrow \text{sort}\left(\left\{c_{i-1,u} | u \in \mathcal{N}_G(j)\right\}\right)$;
11 $S'_{i,j} \leftarrow \text{sort}\left(\left\{c'_{i-1,u} | u \in \mathcal{N}_{G'}(j)\right\}\right)$;
12 $c_{i,j} \leftarrow \text{hash}\left(c_{i-1,j}, S_{i,j}\right)$;
13 $c'_{i,j} \leftarrow \text{hash}\left(c'_{i-1,j}, S'_{i,j}\right)$;
14 **end**
15 **end**
16 **if** *partitions of G and G' are the same over iterations* **then**
17 return True;
18 **else**
19 return False;
20 **end**
21 **end**

a) At each node j, gather all node's colors in its neighborhood, form a multi-set $S_{i,j}$ and sort them in ascending order (lines 6-7).
b) Concatenate the color of the node j and $S_{i,j}$ to produce a tuple $\left(j, S_{i,j}\right)$ and label this tuple with a new color $c_{i,j}$ so that two nodes with the same tuple have the same color value (lines 8-9).

3) At the i^{th} iteration, if the algorithm divides two graphs G and G' into different partitions then two graphs are non-isomorphic and we can stop the algorithm. Otherwise, if all N_v iterations are passed and the partitions of two graphs are consistent over iterations, two graphs have high probability to be isomorphic (lines 10-13).

Basically, 1-WL produces two different colors for two graphs, they are certainly non-isomorphic. In the case of the same coloring results, we are unable to conclude there is an isomorphism between two graphs because there still exists some non-isomorphic graphs with the same 1-WL test, e.g., Fig. 3. However, WL is still powerful enough to distinguish almost all pairs of non-isomorphic graphs [23]. Fig. 4 demonstrates typical steps of 1-WL. The algorithm ends at the 3$^{\text{rd}}$ iterations because of no changes in colors of two graphs. The complexity of 1-WL is $O\left(N_{\text{WL}} N_v\right)$, which is proportional to graph size and the number of iterations.

1-WL is a crucial algorithm in graph learning. It not only deals with the problem of testing the isomorphism of two graphs but also has been a fundamental framework to develop most models invariant to isomorphism recently.

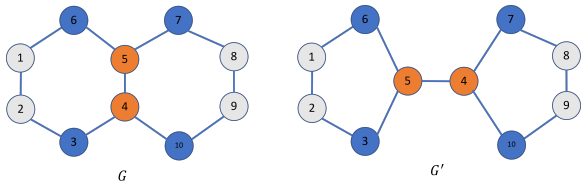

Figure 3. Non-isomorphic graphs with the same 1-WL test result (adopted from [21]).

III. GRAPH STRUCTURE LEARNING

Learning graph topology is a core topic in graph learning, whose goal is to encode entire graph structures or nodes' k-hop connections into a compact vector, which can be processed efficiently and effectively by machine learning algorithms in downstream tasks. Studies on graph structure learning are fairly diverse and their distinction lies in which part of structural information is captured and how to extract it. Overall, graph structure learning can be divided into six approaches of graph coloring, matrix factorization, node embedding, structure encoding as aggregation operator, spectral domain-based methods and motif-based methods.

1. Graph coloring based methods

Graph coloring or graph labeling is a function from a vertex set V to an ordered set (e.g., integers, real numbers, strings) and assigns a value, named color, of this target set to each node. Two nodes with similar local structures (based on predefined criteria) share the same color and vice versa, nodes with different neighborhood topologies are labeled with different colors. As a result, a color value becomes a compact embedding code describing a node's local structure.

PATCHY-SAN [24] introduces a method to extract local regions. First, based on the result of graph coloring (e.g., Weisfeiler-Lehman test [22]), the method produces a sorted list of nodes using their colors and then traverses through this list with a given stride to obtain a node sequence. Next, a breadth-first search procedure is performed to gather the neighbors of each node v with the increasing distance from v until reaching the expected number of K neighboring nodes. The graph is normalized using an extension of coloring algorithms for similar graphs, which is learned to find an optimal labeling from a collection of graphs.

[25] extracts tensorial representations of graphs using a coloring procedure twice across two dimensions. The first coloring process is performed to select a sequence of N_{sel} nodes, similar to [24]. Then, for each node, K closest neighbors are gathered and sorted based on the colors provided by another coloring procedure. Finally, we obtain a final tensor of shape $N_{\text{sel}} \times K \times N_d$, where N_d is the length of categorical one-hot vectors of nodes.

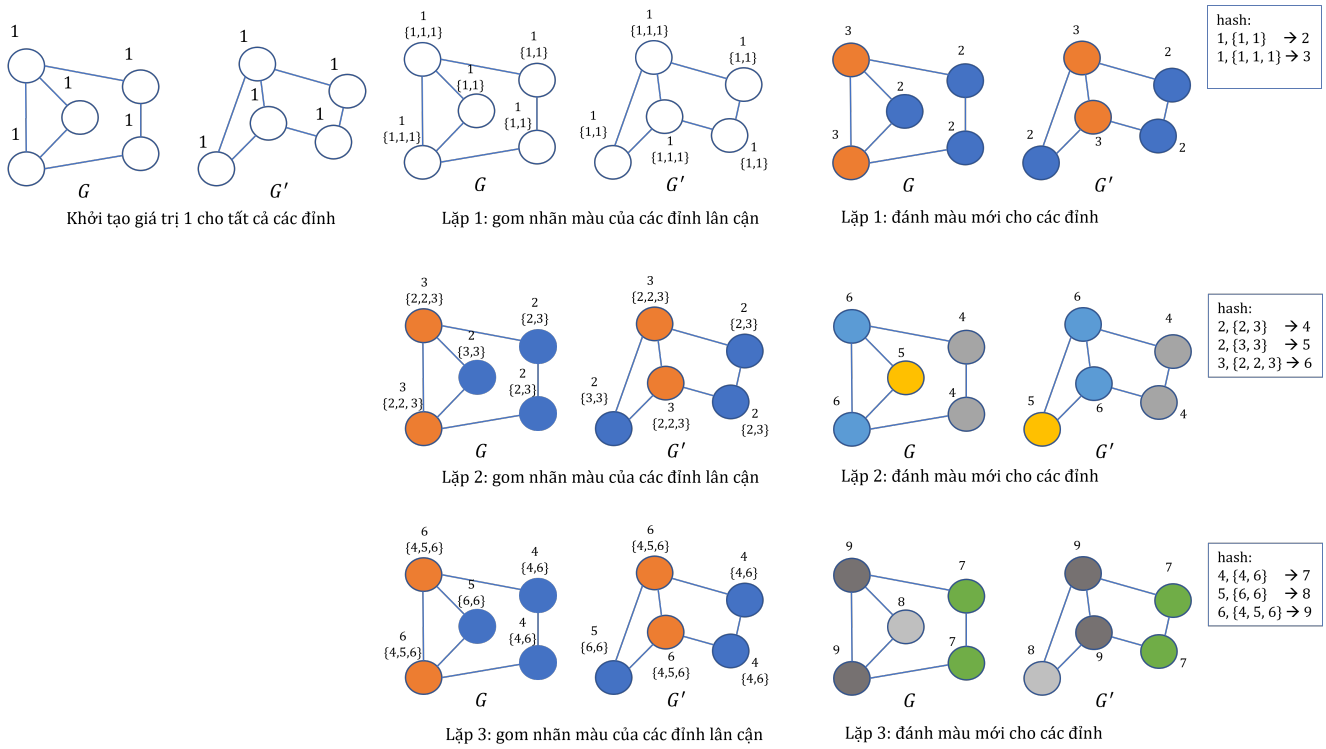

Figure 4. 1-WL procedure (adopted from [21]).

Graph coloring-based methods allow to represent complex topology of graphs into simple color codes, which are convenient for both processing and storage but still distinguishable. Moreover, the codes are designed to be orderable, rendering no requirement to explicitly store in representations, e.g., codes can be implied as tensor indices in [24, 25]. Since this structure encoding approach is completely based on graph coloring algorithms, its representation power mainly depends on the accuracy and speed of the chosen algorithms.

2. Structural matrix factorization

The topology of graphs is initially described in structural matrices such as adjacency matrix or Laplacian matrix. However, such matrices usually only provide basic information, i.e., two vertices are connected or not. For specific applications, matrix factorization can be employed to discover hidden factors by factorizing these matrices into the product of compact matrices. Resulting elementary matrices are problem-driven and help more to improve the performance of systems than the original structural matrices.

GNMTF (Graph regularized non-negative matrix tri-factorization) [26] uses non-negative matrix factorization to extract the intrinsic geometric information of networks for overlapping community detection. Particularly, given the adjacency matrix $\boldsymbol{A}$ of a graph, GNMTF aims to tri-factorize $\boldsymbol{A} \approx \boldsymbol{M}\boldsymbol{\Lambda}\boldsymbol{M}^{\mathsf{T}}$, where $\boldsymbol{M}(i,j)$ indicates the strength of the i^{th} node belonging to the j^{th} community, while the matrix $\boldsymbol{\Lambda}$ represents the relationship among communities. To find $\boldsymbol{M}$ and $\boldsymbol{\Lambda}$, GNMTF minimizes the square loss function: $\left\|\boldsymbol{A} - \boldsymbol{M}\boldsymbol{\Lambda}\boldsymbol{M}^{\mathsf{T}}\right\|_F^2$ or generalized KL-divergence $\sum\left[\boldsymbol{A}(i,j)\log\frac{\boldsymbol{A}(i,j)}{\left(\boldsymbol{M}\boldsymbol{\Lambda}\boldsymbol{M}^{\mathsf{T}}\right)(i,j)} - \boldsymbol{A}(i,j) + \left(\boldsymbol{M}\boldsymbol{\Lambda}\boldsymbol{M}^{\mathsf{T}}\right)(i,j)\right]$.

ESNMF (Ego-Splitting networks using symmetric Non-negative Matrix Factorization) [27] follows the similar idea of optimization-based factorization for overlapping community detection. Instead of factorizing the adjacency matrix of the large-scale graph, ESNMF partitions it into connected subgraphs via an ego-splitting process and incorporates prior information to obtain a subgraph matrix $\hat{\boldsymbol{A}}_{\text{sub}}$. The factorization is done on this matrix: $\left\|\hat{\boldsymbol{A}}_{\text{sub}} - \boldsymbol{M}\boldsymbol{M}^{\mathsf{T}}\right\|_F^2$, where $\boldsymbol{M}(i,j)$ indicates the i^{th} node belongs to the j^{th} community, $\boldsymbol{M}(i,j) = 1$, or not $\boldsymbol{M}(i,j) = 0$.

Gaia Ceddia et al. [28] extend Non-negative Matrix Tri-Factorization to predict the interactions between drugs and proteins. An input network contains multiple types of drugs and proteins, which can be encoded as an association matrix $R \in \mathbb{R}_{\geq 0}^{|V_1|\times|V_2|}$ between two sets of nodes V_1 and V_2, whose element $R(v_1, v_2) > 0$ if $v_1 \in V_1$ and $v_2 \in V_2$ are

connected; and $R(v_1, v_2) = 0$ otherwise. In [28], after introducing an enhanced association matrix $R' \in [0,1]^{|V_1|\times|V_2|}$, where $R'(v_1, v_2)$ is related to the shortest path between v_1 and v_2, the authors propose to convert R' into three non-negative factors by minimizing the following Frobenius norm: $\left\|R' - M_1 Q M_2^{\mathsf{T}}\right\|_F^2$ using an iterative update procedure. The approximate matrix $\hat{R}' = M_1 Q M_2^{\mathsf{T}}$ is used to infer novel associations (i.e., drug-protein-disease interactions) between elements in V_1 and V_2.

Non-negative Matrix Factorization is also adopted to learn clusterable representations of nodes in a semi-supervised manner. In the proposed Unified Semi-Supervised NMF framework (USS-NMF) [29], node representations $\boldsymbol{M}$ are learned by jointly factorizing Pointwise Mutual Information matrix, label matrix, inferred cluster assignment matrix. The factorization allows connected nodes to have similar representations, rendering the capacity of local invariance node encoding. These learned representations can be used as input for any traditional classifiers such as Logistic Regression.

The above NMF-based community detection is shallow methods, which directly map from input space to community membership space. Deep Autoencoder-like Non-negative Matrix Factorization (DANMF) [30] develops a hierarchical mapping to capture the complex and diverse structures of real-world graphs. Formally, the adjacency matrix is approximated by $\boldsymbol{A} \approx \boldsymbol{M_1 M_2} ... \boldsymbol{M}_{N_l} \boldsymbol{Q}$, wherein, $\boldsymbol{M}_l$ and $\boldsymbol{Q}$ are non-negative matrices. Each column of $\boldsymbol{Q}$ reveals the possibility that a node is a member of a community whilst the series of $\boldsymbol{M}_l$ is mapping functions between the input space and the community membership space. To model this hierarchical factorization, DANMF trains a Deep Autoencoder with parameter matrices $\boldsymbol{M_l}$ and $\boldsymbol{Q}$ to minimize the difference between the original structural matrix $\boldsymbol{A}$ and its approximation. DANMF inherits both the intrinsic representation learning of deep models and the power of NMF for community detection and hence it outperforms many shallow NMF-based methods.

Node feature representations can be extracted with low rank matrix factorization. Text-Associated DeepWalk (TADW) proposes a way to incorporate text information, encoded in matrix $\boldsymbol{X}_{\text{text}}$, into a factorization framework. Let $\boldsymbol{T}$ be a transition matrix in PageRank, TADW attempts to approximate $\left(\boldsymbol{T} + \boldsymbol{T}^2\right)/2 \approx \boldsymbol{M}_1^{\mathsf{T}} \boldsymbol{Q} \boldsymbol{X}_{\text{text}}$, wherein $\boldsymbol{M}_1$ and $\boldsymbol{Q}$ are low-rank matrices. Similar to [26, 27, 29], the approximation is casted to minimization problem $\min_{\boldsymbol{M}_1, \boldsymbol{Q}} \left\|\left(\boldsymbol{T} + \boldsymbol{T}^2\right)/2 - \boldsymbol{M}_1^{\mathsf{T}} \boldsymbol{Q} \boldsymbol{X}_{\text{text}}\right\|_F^2 + \alpha\left(\|\boldsymbol{M}_1\|_F^2 + \|\boldsymbol{Q}\|_F^2\right)$, which can be solved by alternatively updating $\boldsymbol{M}_1$ and $\boldsymbol{Q}$. Finally, $\boldsymbol{M}_1$ and $\boldsymbol{Q}\boldsymbol{X}_{\text{text}}$ are treated as the low-dimensional representations of nodes.

Designed for scalability, NetSMF (large-scale network embedding as sparse matrix factorization) [31] extracts the sparse version $\hat{\boldsymbol{L}}$ of Laplacian matrix $\boldsymbol{L}$ and constructs a NetMF matrix sparsifier with the controllable number of non-zero elements. By applying randomized SVD, which is working efficiently on sparse matrices, the authors obtain three matrix factors $\boldsymbol{U}$,Λ and $\boldsymbol{V}$ from the matrix sparsifier, and use the $\boldsymbol{U}\boldsymbol{\Lambda}^{-1/2}$ as network embeddings.

In the scenario of defending adversarial attacks, adjacency matrices are perturbed and GNNs can easily be fooled to make wrong predictions. Pro-GNN [32] attempts to learn a clean adjacency matrix $\boldsymbol{S}$ from the given perturbed matrix $\boldsymbol{A}$. The idea is to find a low-rank and sparse matrix close to $\boldsymbol{A}$ by minimizing the loss $\min_S \|\boldsymbol{A} - \boldsymbol{S}\|_F^2 + \alpha \|\boldsymbol{S}\|_1 + \beta \|\boldsymbol{S}\|_*$, where $\|\cdot\|_1$ and $\|\cdot\|_*$ are L_1 and nuclear norm respectively to force $\boldsymbol{S}$ to be a sparse and low rank matrix. Learning $\boldsymbol{S}$ is done simultaneously with training GNN parameters to improve node classification task. Similarly, LDS (Learning Discrete Structures) [33] is based on an optimization framework to iteratively learn the parameters $\boldsymbol{\theta}$ of a graph generative model, which can draw edges from Bernoulli random variables $\boldsymbol{A} = \text{Ber}(\boldsymbol{\theta})$, and find the optimal parameters of a GCN given generated graphs.

For graph structure learning, matrix factorization approach enables to disentangle hidden factors, which are usually not clearly observed in original structural matrices, and therefore training machine learning methods on extracted meaningful factors yields more benefits. The main drawback of factorization-based structure learning is that it requires much expert knowledge to choose proper factorization methods for target applications, limiting the applicability of this approach. In addition, matrix factorization is also an expensive operation and infeasible to work on large graphs such as social networks or collaboration networks.

3. Node embedding

Node embedding is a group of techniques to encode nodes into low-dimensional vectors, preserving local neighborhood of nodes. More specifically, the geometric distance of two embeddings in a latent space reflects their relationship in original networks. According to [34], node embedding methods share the same 2-stage framework of encoding and decoding. An encoder is a function f^{enc} mapping from a node $v \in V$ to an embedding vector $\boldsymbol{z} \in \mathbb{R}^d$ in a d-dimensional latent space: $\boldsymbol{z} = f^{\text{enc}}(v)$. Meanwhile, a decoder f^{dec} is a mapping from the latent space to the input space, which predicts the relationship of a pair of nodes: $r = f^{\text{dec}}(\boldsymbol{z}_v, \boldsymbol{z}_u)$. Given a structural matrix $\mathcal{A}$, node embedding methods attempt to optimize the following loss: $\mathcal{L} = \sum_{u,v\in V} f^{\text{dis}}\left(f^{\text{dec}}(\boldsymbol{z}_u, \boldsymbol{z}_v), \mathcal{A}(u, v)\right)$. Existing

embedding methods can be distinguished via their choice of the encoder f^{enc}, the decoder f^{dec} and the distance function f^{dis}.

High-Order Proximity preserved Embedding (HOPE) [35] defines the decoding function as inner product $f^{\text{dec}}(z_u, z_v) = z_u^{\top} z_v$ to capture high-order proximity and the distance function as Euclidean distance $\mathcal{L} = \sum_{u,v \in V} \left\| z_u^{\top} z_v - \mathcal{A}(u, v) \right\|$, where $\mathcal{A}$ is a proximity matrix, to ensure that the embeddings of nodes correlate with their observed connections.

Unlike HOPE which tries to reconstruct the structural matrix directly, probabilistic methods such as node2vec are based on random walks to investigate graph topology and attempt to reconstruct the probabilities of random walks. In node2vec [36], the inner product of two latent representations is connected to the conditional probability of two corresponding nodes via random walks $p(v|f^{\text{enc}}(u)) = \frac{\exp(z_u^{\top} z_v)}{\sum_{i,j} z_i^{\top} z_j}$. The entire node2vec framework reconstructs the maximum probability of linked nodes from their embeddings as follows:

$$\begin{aligned} \mathcal{L} &= \sum_{u \in V} \log \left[\prod_{v \in \mathcal{N}(u)} p(v|f^{\text{enc}}(u)) \right] \\ &= \sum_{u \in V} \left[-\log \mathcal{Z}_u + \sum_{v \in \mathcal{N}(u)} z_u^{\top} z_v \right], \end{aligned} \tag{1}$$

wherein $\mathcal{Z}_u = \sum_{v \in V} \exp\left(z_u^{\top} z_v\right)$. Similarly, DeepWalk [37] also follows the same strategy of using truncated random walks to extract the structural information and learn the topology via maximizing the probability of neighborhood.

LINE (Large-scale Information Network Embedding) [38] proposes the idea of learning both the local and global structures. Each structure type is learned via two independent models to produce different embedding vectors $z_{\text{local},u}$ and $z_{\text{global},u}$. Two embeddings are concatenated to create a final representation vector for each vertex u. The local structure is captured by the first-order proximity between nodes. In other words, it is the direct connection between nodes. The decoder as a sigmoid function of inner product of two embedding vectors is defined as the joint probability of two nodes.

$$f_{\text{local}}^{\text{dec}}(v, u) = p_1(v, u) = \frac{1}{1 + \exp\left(-z_{\text{local},v}^{\top} z_{\text{local},u}\right)}$$

Meanwhile, the global structure or the second-order proximity is determined as the common neighborhoods of two nodes. The decoder represents the conditional probability of a context u given a node v:

$$f_{\text{global}}^{\text{dec}}(v, u) = p_2(u|v) = \frac{\exp\left(z_{\text{global},v}^{\top} z_{\text{global},u}\right)}{\sum_{v_1 \in V} z_{\text{global},v}^{\top} z_{\text{global},v_1}}$$

Basically, LINE maps both the local and global structures into continuous embedding vectors that represent the probabilities over vertices. The object functions are trained to minimize the KL-divergence between the joint and conditional probabilities described by two decoders and the corresponding empirical probabilities.

For heterogeneous graphs, Heterogeneous Graph Structure Learning (HGSL) [39] first defines a weighted cosine similarity function:

$$f_{\text{r}}^{\text{dec}}(\boldsymbol{x}_v, \boldsymbol{x}_u) = \frac{1}{K} \sum_{i=1}^{K} \cos\left(\boldsymbol{w}_{k,r} \odot \boldsymbol{x}_v, \boldsymbol{w}_{k,r} \odot \boldsymbol{x}_u\right)$$

where $\boldsymbol{w}_{k,r}$ are learnable parameters, and then HGSL applies it to generate the feature similarity graph $\boldsymbol{A}_r^{\text{SF}}$, which predicts the connection probability of two nodes based on their feature vectors. By propagating feature similarity graph along topological structure $\boldsymbol{A}_r$ of graph, HGSL obtains the feature propagation graphs for head nodes $\boldsymbol{A}_r^{\text{SFH}} = \boldsymbol{A}_r^{\text{SF}} \boldsymbol{A}_r$ and tail nodes $\boldsymbol{A}_r^{\text{SFT}} = \boldsymbol{A}_r \boldsymbol{A}_r^{\text{SF}}$, which represent nodes with similar features and neighbours. These matrices are combined to obtain the feature graph $\boldsymbol{A}_r^{\text{Feat}}$. The cosine similarity function above is also used on embedding vectors of metapaths to construct a semantic graph $\boldsymbol{A}_r^{\text{Sem}}$. Finally, learned structure graph is a combination of the feature graph $\boldsymbol{A}_r^{\text{Feat}}$, the semantic graph $\boldsymbol{A}_r^{\text{Sem}}$ and the original graph $\boldsymbol{A}_r$ via an attention layer. The parameters of the graph structure learning network are jointly trained with 2-layer GCN to predict node labels.

Node embedding-based methods essentially convert discrete graph data into a richer continuous embedding space but still preserve the structural information of nodes. However, these methods suffer the following disadvantages [34]: 1) no parameters sharing between encoders cause computational inefficiency because the number of parameters grows linearly with respect to the graph size and 2) with transductive nature, these methods are unable to extract the embedding vectors of new unseen nodes without retraining embeddings. These main problems hinder the further development of this direction in recent years.

4. Structure encoding as aggregation operators

Structural matrices $\mathcal{A}$ can be viewed as 1) data objects that store the structural information of graphs (which nodes connect to which nodes) and 2) operators that are used to aggregate the information within neighborhood. Aggregation-based structure learning grounds on the latter

characteristic of structural matrices. Since most convolutional graph neural networks (ConvGNN), which are gaining attention from research communities recently, are following this direction to learn graph structures, we focus on ConvGNNs in this part.

Inheriting the principles of deep learning, ConvGNNs consist of many layers of neurons, which connect to those on the next layer. Eq. 2 shows fundamental transformation in a layer, which contains a matrix multiplication between input $\boldsymbol{H}^{(l-1)}$ and learnable parameters $\Theta^{(l-1)}$ of the layer $l-1$, the neighbor aggregation via a matrix multiplication with structural matrix $\mathcal{A}$ and finally a non-linear mapping f^{act}:

$$\boldsymbol{H}^{(l)} = f^{\text{act}}\left(\mathcal{A}\boldsymbol{H}^{(l-1)}\boldsymbol{\Theta}^{(l-1)}\right) \tag{2}$$

wherein, $\boldsymbol{H}^{(0)} = \boldsymbol{X}$ and f^{act} can be any non-linear function such as sigmoid or ReLU. Generally speaking, the input signal of each layer is transformed via a linear mapping controlled by $\Theta^{(l-1)}$ before these transformed vectors are aggregated along edges. To deeply uncover the role of $\mathcal{A}$, let's rewrite Eq. 2 into the form of neuron-wise transformation:

$$\boldsymbol{h}_v^{(l)} = f^{\text{act}}\left(\sum_{u\in\mathcal{N}(v)} \boldsymbol{h}_u^{(l-1)}\boldsymbol{\Theta}^{(l-1)}\right)$$

It can be seen that the structural matrix is only used to provide the connection information to aggregate nodes within a neighborhood, rendering an underuse of graph structures.

The early work on ConvGNNs is NN4G (Neural Networks for Graphs) [40] which directly employs adjacency matrix $\mathcal{A} = \boldsymbol{A}$ to gather the hidden representations of neighbors. Unlike Eq. 2, NN4G takes the feature matrix $\boldsymbol{X}$ of graph as well as the hidden representations of all preceding layers $\boldsymbol{H}^{(i)}$ into account.

$$\boldsymbol{H}^{(l)} = f^{\text{act}}\left(\boldsymbol{X}\boldsymbol{W}^{(l-1)} + \sum_{i=1}^{l-1}\boldsymbol{A}\boldsymbol{H}^{(i)}\boldsymbol{\Theta}^{(i,l-1)}\right)$$

where, $\boldsymbol{W}^{(l-1)}$ and $\Theta^{(i,l-1)}$ are parameters corresponding to feature and hidden representations respectively.

Graph Convolutional Networks (GCN) [41] is one of seminal works on applying deep learning to graphs. In GCN, the structural matrix is chosen as a deterministic function of adjacency matrix $\mathcal{A} = \boldsymbol{D}^{-\frac{1}{2}}(\boldsymbol{A}+\mathbf{I})\boldsymbol{D}^{-\frac{1}{2}}$, wherein $\boldsymbol{A}+\mathbf{I}$ implies self-connections added and $\boldsymbol{D}^{-\frac{1}{2}}$ works as a normalizing factor.

$$\boldsymbol{H}^{(l)} = f^{\text{act}}\left(\boldsymbol{D}^{-\frac{1}{2}}(\boldsymbol{A}+\mathbf{I})\boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{H}^{(l-1)}\boldsymbol{\Theta}^{(l-1)}\right) \tag{3}$$

Deep Graph Convolutional Neural Network (DGCNN) [42] is also based on adjacency matrices to incorporate structural information but performs a different way to compute the structural matrix. For particular, by setting $\tilde{\boldsymbol{A}} = \boldsymbol{A}+\mathbf{I}$ and $\tilde{\boldsymbol{D}}(i,i) = \sum_j \tilde{\boldsymbol{A}}(i,j)$, we obtain a hidden layer update equation, similar to GCN's, but different in normalization factor:

$$\boldsymbol{H}^{(l+1)} = f^{\text{act}}\left(\tilde{\boldsymbol{D}}^{-1}(\boldsymbol{A}+\mathbf{I})\boldsymbol{H}^{(l-1)}\boldsymbol{\Theta}^{(l-1)}\right)$$

Bo Jiang et al. extend GCN to semi-supervised graph learning, where the structural matrix is not provided explicitly. The core idea of Graph Learning-Convolution Network (GLCN) [43] is to construct the structural matrix $\mathcal{A}$ by leveraging node embedding techniques (Subsec. III.3). The first layer of GLCN, parameterized by a vector $\boldsymbol{w}$ that learns the similarity of two nodes v and u using the following equation:

$$\mathcal{A}(v,u) = \frac{\boldsymbol{A}(v,u)\exp\left(\text{ReLU}\left(\boldsymbol{w}^{\mathsf{T}}|x_v - x_u|\right)\right)}{\sum_{j=1}^{n}\boldsymbol{A}(v,j)\exp\left(\text{ReLU}\left(\boldsymbol{w}^{\mathsf{T}}|x_v - x_j|\right)\right)} \tag{4}$$

In some cases, when $\boldsymbol{A}$ is not available, we drop the $\boldsymbol{A}(\cdot,\cdot)$ to yield the corresponding equation without $\boldsymbol{A}$. The next layers are typical convolutional layers, similar to GCN, but work on computed structural matrix $\mathcal{A}$ and its corresponding degree matrix $\boldsymbol{D}_{\mathcal{A}} = \text{diag}(d_1, d_2, ..., d_{N_v})$,where $d_i = \sum_{j=1}^{N_v}\mathcal{A}(i,j)$:

$$\boldsymbol{H}^{(l)} = f^{\text{act}}\left(\boldsymbol{D}_{\mathcal{A}}{}^{-\frac{1}{2}}(\mathcal{A}+\mathbf{I})\boldsymbol{D}_{\mathcal{A}}^{-\frac{1}{2}}\boldsymbol{H}^{(l-1)}\boldsymbol{\Theta}^{(l-1)}\right)$$

Finally, a perceptron layer is stacked on top to predict the label of the input graph.

The aggregation above can be viewed as a message passing process which propagates the message from a neighbor u to a node v. GAT (Graph Attention Networks) [44] assumes that neighbors send messages with different contributions to the central node v and therefore GAT introduces an attention-based mechanism to re-compute the weights of messages. The hidden state of a neuron is updated as follows:

$$\boldsymbol{h}_v^{(l)} = \sigma\left(\sum_{u\in\mathcal{N}(v)\cup\{v\}}\alpha_{vu}^{(l)}\boldsymbol{\Theta}^{(l)}\boldsymbol{h}_u^{(l-1)}\right)$$

The attention weight $\alpha_{vu}^{(l)}$ shows the contribution of a node u to a node v and is updated as $\alpha_{vu}^{(l)} = \text{softmax}\left(\text{LeakyReLU}\left(\boldsymbol{w}^{\mathsf{T}}\left[\boldsymbol{\Theta}^{(l)}\boldsymbol{h}_v^{(l-1)}, \boldsymbol{\Theta}^{(l)}\boldsymbol{h}_u^{(l-1)}\right]\right)\right)$, where $\boldsymbol{w}$ and $\boldsymbol{\Theta}$ are learnable parameters.

Information aggregation using structural matrices is widely implemented in a lot of state-of-the-art graph neural network systems nowadays. But richer and higher-level structural information encoded in $\mathcal{A}$ has not been discovered yet except for some basic connections within k-hops neighborhood. There is a question about how well graph

neural networks can preserve graph structures [45]. This is confirmed in [46], whose experiments show that structure-agnostic methods outperform graph neural networks on structure-driven problems (i.e., chemical datasets). Hence, more studies should be conducted to explore and extend the capacity of aggregating graph structural information.

5. Spectral domain-based methods

Spectral graph learning consists of methods that are based on graph Fourier transform and have a strong connection to the theory of graph signal processing [47–49]. Given an undirected graph whose structure can be represented by a structural matrix $\mathcal{A}$ (adjacent matrix or Laplacian matrix). Since $\mathcal{A}$ owns special properties such as real symmetric positive semi-definite, it can be eigen-decomposed into $\mathcal{A} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\top$, wherein $\boldsymbol{U} = \left[\boldsymbol{u}_1, \boldsymbol{u}_2, ..., \boldsymbol{u}_{N_v}\right] \in \mathbb{R}^{N_v \times N_v}$ is orthonormal eigen-vectors ($\boldsymbol{U}^\top\boldsymbol{U} = \mathbf{I}$) and $\boldsymbol{\Lambda}(i,i) = \lambda_i$ are eigen-values. A graph signal is defined as a vector $\boldsymbol{s} \in \mathbb{R}^{N_v}$, whose i^{th} element is an observed signal at the i^{th} vertex. Graph Fourier transform applied to $\boldsymbol{s}$ is demonstrated as $\tilde{\boldsymbol{s}} = \mathcal{F}(\boldsymbol{s}) = \boldsymbol{U}^\top\boldsymbol{s}$ and its inverse transform is $\mathcal{F}^{-1}(\tilde{\boldsymbol{s}}) = \boldsymbol{U}\tilde{\boldsymbol{s}}$. Basically, graph Fourier transform maps an observed signal in an input space to $\tilde{\boldsymbol{s}}$ in a spectral space, which is a space of columns of $\boldsymbol{U}$. In other words, the i^{th} element of $\tilde{\boldsymbol{s}}$ is a new coordinate of $\boldsymbol{s}$ in the spectral space or $\boldsymbol{s} = \sum_i \tilde{s}_i \boldsymbol{u}_i$. A convolution between a graph signal $\boldsymbol{s}$ and a filter $\boldsymbol{g} \in \mathbb{R}^{N_v}$ is defined as:

$$\boldsymbol{s} * \boldsymbol{g} = \mathcal{F}^{-1}\left(\mathcal{F}(\boldsymbol{s}) \odot \mathcal{F}(\boldsymbol{g})\right) = \boldsymbol{U}\left(\boldsymbol{U}^\top\boldsymbol{s} \odot \boldsymbol{U}^\top\boldsymbol{g}\right)$$

where $\odot$ is Hadamard product and $*$ is convolution operator. If $\boldsymbol{g}_\theta = \text{diag}\left(\boldsymbol{U}^\top\boldsymbol{g}\right)$ then we can simplify $\boldsymbol{s} * \boldsymbol{g}_\theta = \boldsymbol{U}\boldsymbol{g}_\theta\boldsymbol{U}^\top\boldsymbol{s}$.

As a branch of ConvGNNs, spectral methods are strongly related to spatial-based ones or aggregation-based structure learning mentioned in Subsec. III.4. The basic idea is to convert the structural matrix $\mathcal{A}$ into a spectral domain, apply a filter and project back to the spatial domain. Let us revisit Eq. 2 and view the resulting multiplication $\boldsymbol{H}^{(l-1)}\boldsymbol{\Theta}^{(l-1)}$ as multi-dimensional graph signals on a graph described by $\mathcal{A}$. If we employ a filter $\boldsymbol{g}_\theta$ in a spectral space, whose basis is $\boldsymbol{U}$, the layer-wise transformation of spectral methods can be derived as:

$$\boldsymbol{H}^{(l)} = f^{\text{act}}\left(\boldsymbol{U}\boldsymbol{g}_\theta\boldsymbol{U}^\top\boldsymbol{H}^{(l-1)}\boldsymbol{\Theta}^{(l-1)}\right) \tag{5}$$

Spectral CNN [50] is one of the first studies on this direction. The transformation is done by choosing a structural matrix as Laplacian matrix and decomposing it. The transformation equation of Spectral CNN is as follows:

$$\boldsymbol{H}^{(l)}_{:,j} = f^{\text{act}}\left(\sum_{i=1}^{N_c{}^{(l-1)}} \boldsymbol{U}\boldsymbol{\Phi}^{(l)}_{ij}\boldsymbol{U}^\top\boldsymbol{H}^{(l-1)}_{:,i}\right)\left(j = 1, 2, ..., N_c{}^{(l)}\right)$$

where $N_c{}^{(l)}$ is the number of channels at the layer l. The filter $\boldsymbol{g}_\theta = \boldsymbol{\Phi}^{(l)}_{ij}$ is a diagonal matrix, whose diagonal entries can be updated during training. Eigen-decomposition is expensive and causes $O\left(N_v^3\right)$ in computational complexity. To tackle this problem, Chebyshev Spectral CNN [51] and GCN [41] are introduced to reduce the complexity using polynomial approximation.

ChebNet (Chebyshev Spectral CNN) [51] assumes that the filter $\boldsymbol{g}_\theta$ can be approximated by Chebyshev polynomial with respect to $\boldsymbol{\Lambda}$ as $\boldsymbol{g}_\theta = \sum_{i=0}^{K-1}\theta_i T_i\left(\bar{\boldsymbol{\Lambda}}\right)$, where $\bar{\boldsymbol{\Lambda}} = 2\boldsymbol{\Lambda}/\lambda_{\max} - \mathbf{I}_{N_v} \in [-1, 1]$. The advantage of Chebyshev polynomial is that each term can be recursively computed from its preceding terms $T_i(\boldsymbol{x}) = 2\boldsymbol{x}T_{i-1}(\boldsymbol{x}) - T_{i-2}(\boldsymbol{x})$ where we define $T_0(\boldsymbol{x}) = 1$ and $T_1(\boldsymbol{x}) = \boldsymbol{x}$. As a result, the convolution between a graph signal $\boldsymbol{s}$ and a filter $\boldsymbol{g}_\theta$ is obtained as:

$$\boldsymbol{s} * \boldsymbol{g}_\theta = \boldsymbol{U}\left(\sum_{i=0}^{K-1}\theta_i T_i\left(\bar{\boldsymbol{\Lambda}}\right)\right)\boldsymbol{U}^\top\boldsymbol{s}$$

Let us denote $\bar{\boldsymbol{L}} = 2\boldsymbol{L}/\lambda_{\max} - \mathbf{I}$ which can be considered as a normalizing form of Laplacian matrix $\boldsymbol{L}$ and we have $\bar{\boldsymbol{L}} = 2\boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\top/\lambda_{\max} - \mathbf{I} = \boldsymbol{U}\bar{\boldsymbol{\Lambda}}\boldsymbol{U}^\top$ and then $T_i\left(\bar{\boldsymbol{L}}\right) = \boldsymbol{U}T_i\left(\bar{\boldsymbol{\Lambda}}\right)\boldsymbol{U}^\top$. The convolution $\boldsymbol{s} * \boldsymbol{g}_\theta$ can be rewritten as:

$$\boldsymbol{s} * \boldsymbol{g}_\theta = \sum_{i=0}^{K-1}\theta_i T_i\left(\bar{\boldsymbol{L}}\right)\boldsymbol{s}$$

The new form indicates that the convolution can be approximated by recursively computing $T_i\left(\bar{\boldsymbol{L}}\right)$ in spatial domain and therefore the graph Fourier transform is completely removed and the computational cost is $O(KN_e)$. Another approximation approach is CayleyNet [52] which replaces Chebyshev polynomial with Cayley polynomial. It is proven that ChebNet is an instance of CayleyNet.

GCN (Graph Convolutional Networks) is a seminal work proposed by Kipf et al. in [41]. As a bridge between spectral and spatial approaches in ConvGNNs, GCN inherits the strength of both. From the spectral view, GCN is the first order approximation of ChebNet with $K = 1$ and $\lambda_{\max} = 2$ and the convolution equation in spatial domain is $\boldsymbol{s} * \boldsymbol{g}_\theta = \theta_0\boldsymbol{s} - \theta_1\boldsymbol{D}_{\mathcal{A}}^{-\frac{1}{2}}\mathcal{A}\boldsymbol{D}_{\mathcal{A}}^{-\frac{1}{2}}\boldsymbol{s}$. To avoid over-fitting, GCN assumes $\theta = \theta_0 = -\theta_1$. Furthermore, it uses a normalized version of adjacency matrix, i.e., $\mathcal{A} = \boldsymbol{A} + \mathbf{I}_{N_v}$ and $\boldsymbol{D}_{\mathcal{A}}(i,i) = \sum_j \mathcal{A}(i,j)$, to stabilize training and prevent gradients from exploding or vanishing. Its convolution equation is given as:

$$\boldsymbol{s} * \boldsymbol{g}_\theta = \theta\left(\mathbf{I}_{N_v} + \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{-\frac{1}{2}}\right)\boldsymbol{s} = \theta\boldsymbol{D}_{\mathcal{A}}^{-\frac{1}{2}}\mathcal{A}\boldsymbol{D}_{\mathcal{A}}^{-\frac{1}{2}}\boldsymbol{s}$$

DGCN (Dual Graph Convolutional Network) [16] employs two networks with the same architectures of GCN. These networks are different in input structural matrices. The first network (named $\text{Conv}_{\boldsymbol{A}}$) is based on the normalized adjacency matrix in [41] to extract local structures around

nodes. The second network takes PPMI (Positive Pointwise Mutual Information) (described in Sec. II) as input structural matrix. Two networks contain their own softmax layers on top to individually predict output classes. DGCN is trained with a loss function to minimize the inconsistency between these two outputs and labels.

Among graph structure learning approaches, spectral domain-based methods have a strong theoretical background of graph signal processing, rendering its notable capacity of explainability. Particularly, it has been proven that eigen-values and eigen-vectors are associated with graph connected components, showing that spectral domain-based methods can learn the topology of graphs. The main drawback of these methods are high computational cost required to convert signals from spatial to spectral domain but it has not been a big problem due to excellent improvement [41, 51, 52] in computation time recently.

6. Motif-based methods

Motifs are *"patterns of interconnections that occur with significantly higher frequencies in complex networks than those in random networks"* [53]. Generally speaking, network motifs are popular subgraph structures frequently occurring in real graphs. Methods in this approach are mainly based on computing the frequency of motifs in input networks. Elementary motif counting methods are not only able to represent the structures of any graphs but also are effective to measure the structural similarity between two graphs even with different number of nodes and edges. Therefore, they are able to process a lot of real-world network types varying in size and topology.

Graphlets are one of popular motifs in graph learning which were introduced in 2004 for protein-protein interaction networks [54] and are developed in graph kernel by Shervashidze et al. [55]. Since the number of graphlets exponentially increases with respect to the number of vertices, the graphlet set is limited to small graphs whose size is ≤ 5 vertices. Let $\boldsymbol{g} = \left[g_1, g_2, ..., g_{N_{\text{graphlet}}}\right]^\top$ be a collection of graphlets and $\phi_G(\boldsymbol{g})$ be a frequency vector, whose element is the frequency of occurrence of the corresponding graphlet in G. This vector can be normalized by dividing it by the sum of frequency of all graphlets in G and we gain the representation vector $\bar{\phi}_G(\boldsymbol{g})$ of G. It can be seen that the structure of a graph is encoded as a normalized frequency vector, two graphs G and G' with the similar representation vectors have a high probability to share the same graph structure. Therefore, graphlet-based methods measure the graph similarity by defining the distance between these vectors, e.g., inner product-based distance:

$$k_{\text{graphlet}}(G, G') = \bar{\phi}_G(\boldsymbol{g})^\top \bar{\phi}_G(\boldsymbol{g}') \tag{6}$$

This similarity measure can be used as kernel function in kernel-based machine learning frameworks such as SVM [56]. Other studies adopt the same idea of graph kernel in Eq. 6 but propose to use low-cost substructures such as subtree patterns or random walks.

Most works rely on Eq. 6 which assumes the independence between substructures. This is not always true. Let us consider an example on graphlets. A graphlet with size $K+1$ can be driven from a smaller graph with size K by adding nodes or edges, showing a strong relationship between graphlets. Deep Graph Kernel [57] incorporates such relationships by introducing a positive semi-definite matrix $\boldsymbol{M}$:

$$k(\boldsymbol{x}, \boldsymbol{x'}) = \phi(\boldsymbol{x})^\top \boldsymbol{M} \phi(\boldsymbol{x'}) \tag{7}$$

This matrix encodes the dependence between substructures and can be learned automatically. In some cases, if the relationship is explicit, $\boldsymbol{M}$ can be pre-computed, e.g., $\boldsymbol{M}(i, j)$ can be Levenshtein distance (edit distance - the number of node/edge addition/deletion steps to transform a substructure g_i to another substructure g_j). Eq. 7 is a general frameworks that can be applied to a lot of substructures such graphlets, subtree pattern [58, 59] or random walks [60].

HONE (Higher-order Network representation learning) [61] also uses a motif counting strategy to capture the higher-order structural dependence by first defining a set of motifs, which essentially are graphlets or orbits (graphlet automorphisms) and then building a weighted motif adjacency matrix for each motif, whose element (i, j) counts the number of occurrences of edge $(i, j) \in E$ in that motif. The structural matrix $\mathcal{A}$, which is actually a derived matrix, such as transition matrix, (normalized) Laplacian matrix, random walk-based Laplacian matrix, of each weighted motif adjacency matrix, is computed and the local/global embedding is learned for each k-step structural matrix $\mathcal{A}^k$ using matrix factorization techniques described in Subsec. III.2

To encode the higher order local structures for node embedding learning, mGCMN (graph convolutional multi-layer networks based on motifs) [62] follows the same idea of motif counting matrix similar to [61]. This means an element (u, v) of its custom motif matrix $\mathcal{A}$ indicates the number of occurrence of the nodes u and v in the corresponding motif. But unlike [61], mGCMN accepts the case when u and v are the same node. The custom motif matrix is fed into a graph neural network with convolutional layers, MLP layers and a softmax layer for a final embedding.

Ghadeer Abuoda et al. [63] leverage motifs to form high-order topological features to predict the existence of new links. The paper focuses on small motifs of size 3, 4 and 5, whose corresponding number of connected non-isomorphic motifs are 2, 6 and 21. Each motif forms a corresponding feature and therefore there are 29 features, which allows to capture the high level of topological details of graphs. Finally, a traditional classifier is built on these extracted features to perform a link prediction task.

Motif-based methods are powerful tools to learn the higher-order structures in graphs and have a wide range of applications in various areas, especially applications require structural information such as molecular or protein structures. Comparing with other graph structure approaches, motif discovery algorithms are still time-consuming in terms of high performance time in measuring statistics of motifs in large graphs, preventing us from employing motifs with larger size (i.e., >5 nodes). In the future, further efforts are needed to accelerate motif discovery algorithms.

IV. GRAPH ISOMORPHISM

When developing machine learning systems on graphs, one has to deal with another special characteristics of graph data: graph isomorphism. An isomorphism between two graphs G and G' is a bijection between their node sets V and V' that preserves adjacency [64]. In this case, two graphs are *isomorphic*. Graph isomorphism problem is to determine whether two given graphs are isomorphic or not. This is known to be unresolved and still an open problem in computer science. The complexity of the problem is unknown. The achievements in graph isomorphism research show that its complexity is not known to be in NP-complete or be solvable in polynomial time [20]. One of the most popular and practical algorithms to check graph isomorphism until now is Weisfeiler-Lehman test [22], which allows to efficiently eliminate non-isomorphic graphs. Since graph topologies are usually represented via popular structural matrices $\boldsymbol{\mathcal{A}}$ and their properties are encoded in feature matrices $\boldsymbol{X}$, isomorphism causes row permutation in $\boldsymbol{X}$ and row/column permutation in $\boldsymbol{\mathcal{A}}$. As a result, two isomorphic graphs look different but are actually the same and there exists a permutation matrix $\boldsymbol{P} \in \{0,1\}^{N_v \times N_v}$ transforming the structural and feature matrices of this graph into ones of the other. To develop an algorithm f handling graph isomorphism, there are two important definitions related to permutation invariance as follows:

- **Permutation invariance:** [65]: a function $f(\boldsymbol{\mathcal{A}}, \boldsymbol{X})$ is permutation invariant if the output is unchanged for any node ordering $f\left(\boldsymbol{P}\boldsymbol{\mathcal{A}}\boldsymbol{P}^{\mathsf{T}}, \boldsymbol{P}\boldsymbol{X}\right) = f(\boldsymbol{\mathcal{A}}, \boldsymbol{X})$.
- **Permutation equivariance** [65]: a function $f(\boldsymbol{\mathcal{A}}, \boldsymbol{X})$ is permutation equivariant if permuting the input or the output of the function obtains the same result $f\left(\boldsymbol{P}\boldsymbol{\mathcal{A}}\boldsymbol{P}^{\mathsf{T}}, \boldsymbol{P}\boldsymbol{X}\right) = \boldsymbol{P}f(\boldsymbol{\mathcal{A}}, \boldsymbol{X})$.

In recent years, designing permutation invariant graph learning systems has been receiving increased attention in literature. Specifically, a theoretical line of studies on permutation invariant architectures and universal approximation on graphs has been introduced for graph neural networks, starting with a seminal work of Deep Sets [66]. In [66], Manzil Zaheer et. al investigate permutation invariant and equivariant functions operating on sets. The key contribution is the theorem stating that a set function is invariant to the permutation of elements in a given set from a countable domain iff it can be decomposed into the following form:

$$f(\boldsymbol{\mathcal{A}}, \boldsymbol{X}) = f_{\rho}\left(\oplus f_{\psi}(\boldsymbol{\mathcal{A}}, \boldsymbol{X})\right) \tag{8}$$

where f_{ψ} is permutation equivariant, f_{ρ} is an arbitrary function and $\oplus$ is permutation invariant operator such as sum, min, max. Eq. 8 plays an important role because most graph neural networks with permutation invariance capacity are following this fundamental architecture. However, the constraint of countable domain in [66] hinders the invariant architecture above from effective implementation for real-world applications. [67] considers a specific case of sum-decomposition, where $\oplus$ is a summation, and extends the framework to continuity on uncountable domain for more practical usefulness.

To deeply investigate the invariance and equivariance properties of graphs, some studies [68] pay attention to linear layers that are the building-blocks of most deep networks. For graph data, [68] proves that the dimensions of all permutation invariant and equivariant linear layer spaces are 2 and 15 respectively. More importantly, these dimensions are independent of the number of vertices and therefore invariant and equivariant networks can be applied to different-sized graphs. Orthogonal basis of these spaces are computed and introduced in the paper to build linear layers with invariance and equivariance capacities.

Haggai Maron et. al. [69] examine on the capacity of neural networks, whose architecture consists of a sequence of linear equivariant layers, non-liner layers and invariant layers, similar to [66], to approximate any (continuous) invariant function. The paper considers the general cases of any subgroups of a symmetric group acting on $\mathbb{R}^n$ by permuting coordinates. Suppose that a G-invariant function is a function satisfying $f(g \cdot x) = f(x)$ for all element $x \in \mathbb{R}^n$ and action g in the subgroup. Results from the paper show that a G-invariant network can be a universal approximator of arbitrary continuous G-invariant function

if high-order tensors are allowed. If group is a set, only the first order G-invariant network (with maximal tensor order of 1) is required to be universal and one instance is shown in [66]. The work of k-IGNs [69] is the first step to discover the approximation power of invariant networks in general and deep networks on graphs in particular, rendering better understanding of the limitation and capacity of invariant architecture. The work of [69] is confirmed in [70], which introduces an alternative proof based on Stone-Weierstrass theorem for algebra of real-valued functions. Furthermore, [70] extends the results to equivariant functions using a novel generalized version of Stone-Weierstrass theorem.

It is notable that not all graph learning methods are equipped with permutation invariant or equivariant operators (for example, GraphSAGE with LSTM aggregation [71]) and hence, theoretically, they are unable to handle isomorphic graphs in practice. This section reviews common techniques to train permutation invariant graph learning systems. Further, although permutation invariance and equivariance are equally crucial in theory, invariance property has been receiving more attention for graph data due to its direct connection to graph isomorphism problem. Hence, we mainly focus on introducing graph learning methods with permutation invariance in the following part and leave the review on permutation equivariant systems for future work.

1. Aggregation function (readout function)

A fundamental approach to build a permutation invariant system relies on ordering invariant aggregation functions. Aggregation functions (or readout functions or pooling layers) are used to gather the information of a node and nodes/edges in its neighborhood. Most methods in this approach follows the framework proposed in [66] with a permutation equivariant function f_{ψ} and permutation invariance aggregation operator $\oplus$ followed by an arbitrary function f_{ρ} (Eq. 8). The most popular permutation invariant aggregators are sum, product, minimum, maximum, covariance, some of which are reviewed in this section.

[72] introduces a framework, named Graph Isomorphism Network (GIN), which is as powerful as Weisfeiler-Lehman graph isomorphism test. To ensure the permutation invariance capacity, a readout function/pooling layer aggregates node features to yield graph representations. The paper examines three permutation invariant functions including sum, mean and max aggregators and ranks them by expressive power. Namely, one with the most representational power is sum that describes the whole multi-set whilst mean expresses the distribution of the multi-set and max reduces the multi-set to a simple set. Since mean and max aggregators are not injective, they are weaker in distinguishing isomorphism than sum operators. This explains the common choice of sum operators in existing invariance systems [67, 72].

Set Transformer [73] is designed to model the interactions of input set elements. The proposed architecture consists of an encoder, which is equivariant to permutation, and a decoder, whose pooling layer is a permutation invariant transformation or essentially is a sum of outputs from the encoder. Unlike most popular networks, which are based on GCNs or their variants, Set Transformer is completely built on layers with attention mechanism and therefore it allows to learn the high-order connections between elements in sets.

Besides providing permutation invariance, some studies such as Bilinear GNN [74] extends the capacity of aggregators to capture interactions between neighbor nodes, which is not encoded by existing models. To that end, Bilinear GNN computes the sum of all node pairs in the neighborhood of a node v, namely $f(v) = \frac{2}{d_v(d_v+1)} \sum_{i\in\mathcal{N}(v)\cup\{v\}} \sum_{j\in\mathcal{N}(v)\cup\{v\}\&i<j} \boldsymbol{h}_i\boldsymbol{W}\odot\boldsymbol{h}_j\boldsymbol{W}$, where d_v is the degree of the node v. The interaction between two nodes is effectively modeled via element-wise product between their representation vectors. It is notable that the interaction aggregation takes the central node into account to enrich the aggregation result in the case of sparse local structures with several neighbors.

A drawback of aggregation operators of sum, max or min is its scalar output, causing the significant loss of information, especially for local structures with many nodes. Based on the idea of vector output in capsule networks, GCAPS-CNN (Graph Capsule Convolutional Neural Networks [75] is able to capture higher order statistical information of adjacent nodes. Given a hidden representation vector $\boldsymbol{H}^{(l)}$ at the layer l, GCAPS-CNN calculates the covariance of $\boldsymbol{H}^{(l)}$, which is known to be invariant to permutation, as $f\left(\boldsymbol{H}^{(l)}\right) = \frac{1}{N}\left(\boldsymbol{H}^{(l)}-\mu\right)^{\mathsf{T}}\left(\boldsymbol{H}^{(l)}-\mu\right)$, where μ is the mean vector of $\boldsymbol{H}^{(l)}$. Compared with scalar aggregators, much information of data distribution is preserved in covariance such as shape, norm and angles between node features. GCAPS-CNN can be classified as a permutation invariance aggregator based on matrix multiplication.

Similarly, PiNet (Permutation Invariant Graph Neural Network) [76] is an end-to-end permutation invariant network using matrix multiplication. A feature matrix $\boldsymbol{X}$ and an adjacency matrix $\boldsymbol{A}$ are passed through neural layers to obtain the representation matrices $\boldsymbol{Z}_{\boldsymbol{X}} \in \mathbb{R}^{N_v\times N_{\boldsymbol{X}}}$ and $\boldsymbol{Z}_{\boldsymbol{A}} \in \mathbb{R}^{N_v\times N_{\boldsymbol{A}}}$ respectively, where $N_{\boldsymbol{X}}$ and $N_{\boldsymbol{A}}$ are the corresponding hidden dimensions. Next, these two matrices are combined in a product layer $\boldsymbol{Z}_{\boldsymbol{X}}^{\mathsf{T}}\boldsymbol{Z}_{\boldsymbol{A}} \in \mathbb{R}^{N_{\boldsymbol{X}}\times N_{\boldsymbol{A}}}$, which is

invariant to node orders$(\boldsymbol{P}\boldsymbol{Z}_{\boldsymbol{X}})^{\mathsf{T}}(\boldsymbol{P}\boldsymbol{Z}_{\boldsymbol{A}}) = \boldsymbol{Z}_{\boldsymbol{X}}^{\mathsf{T}}\left(\boldsymbol{P}^{\mathsf{T}}\boldsymbol{P}\right)\boldsymbol{Z}_{\boldsymbol{A}} = \boldsymbol{Z}_{\boldsymbol{X}}^{\mathsf{T}}\boldsymbol{Z}_{\boldsymbol{A}}$ for any permutation matrix $\boldsymbol{P}$. Basically, the element (i, j) of the product matrix is the inner product of the i^{th} feature-related representation vector and the j^{th} structural representation vector across all nodes.

Due to the efficiency and scalability, especially for huge social networks, aggregation-based approach is the most practical choice to develop graph learning systems with permutation invariance power. However, compressing huge amount of neighborhood information into single scalars or vectors via aggregators such as sum or max causes the significant loss of information and results in the misclassifcation between non-isomorphic graphs. Although high-order tensors [69] help to improve the power of discrimination, they add more computation cost. Therefore, developing effective aggregation methods that balance the representation power and scalability is a challenging question in this research direction.

2. Ordering-based approach

To overcome the loss of information when using aggregation functions, a few attempts have been done to preserve the information of neighbor nodes but still satisfy permutation invariance constraint by sorting them to form a consistent sequence, which is the same for any node permutation. The idea is based on the basic property of sorting: given a sequence of real-valued numbers, sorting algorithms produce the same ordered sequence, e.g., ascending or descending order, for its permuted sequences and the ordered sequence can be considered as canonical form of the input sequence. As a result, two isomorphic graphs have the same representation after sorting step and then the goal of permutation invariance is obtained.

Deep Graph Convolutional Neural Network (DGCNN) [4] proposes a sorting-based pooling layer, named SortPooling, to reorder nodes by representation values in a consistent and meaningful manner. Support that after passing all graph convolution layers, the concatenated representation matrix is $\boldsymbol{H}^{(1:N_l)} \in \mathbb{R}^{N_v \times N_c}$,where $N_c = \sum_{l=1}^{N_l} N_c{}^{(l)}$ is the total number of channels over all representation layers, $\boldsymbol{H}^{(1:N_l)}$ is sorted row-wise according to the last column of $\boldsymbol{H}^{(1:N_l)}$. If two rows have the same last column values, the second last and the next one is used for comparison until an order can be identified. Basically, this sorting strategy is colexicographical ordering. The proposed SortPooling has two key advantages compared with scalar aggregation operators: (1) much information is allowed to pass, rendering more meaningful representations and (2) by storing the order of SortPooling layer's input, the initial state of the node (before sorting) can be restored and this also enables to do back-propagation for SortPooling.

PATCHY-SAN [24] is a graph learning method that strictly follows the intuition of CNNs' filters on local receptive fields with fixed size and defined neighboring order. To that end, for a selected node v, PATCHY-SAN first extracts K neighbor nodes using a breath-first search algorithm with increasing distance from the target node v and find an optimal order for this K-node sequence. Unlike the aforementioned methods which simply perform a sorting procedure, PATCHY-SAN implements an optimization step to find an optimal solution as canonical ordered sequence. In particular, the graph labeling optimization problem is $\hat{l} = \underset{l}{\operatorname{argmin}}\, \mathbb{E}\left[\left|f_{\boldsymbol{A}}^{\text{dis}}\left(\boldsymbol{A}_{G'_K}, \boldsymbol{A}_{G''_K}\right) - f^{\text{dis}}\left(G'_K, G''_K\right)\right|\right]$, where $f_{\boldsymbol{A}}^{\text{dis}}$ and f^{dis} are distance functions on graphs and adjacency matrices with K nodes. By solving this object function, we can obtain an optimal node order $\hat{l}$ such that, for any uniformly random graphs G'_K and G''_K from a graph collection with K nodes, the expected difference between random graphs is minimized. The advantage of optimization-based approach is that it not only works on isomorphic but also similar graphs with different connections and therefore more generalization in practice.

For molecule related problems, Coulomb matrix is adopted to describe the basic properties and structures of molecules and it is formulated as follows:

$$C(i, j) = \begin{cases} 0.5 z_i^{2.4} & \forall i = j \\ \frac{z_i z_j}{|\boldsymbol{r}_i - \boldsymbol{r}_j|} & \forall j \neq j \end{cases}$$

where, z_i and $\boldsymbol{r}_i$ are the charge and 3D position of the i^{th} atom in a given molecule. Basically, the elements on the diagonal indicate polynomial fit of the potential energies of the free atoms whilst the other elements are Coulomb repulsion between nuclei pairs. Since Coulomb matrix lacks the capacity of permutation invariance, rendering the exponential blow-up of Coulomb descriptors for the same molecule. [77] proposes to transform C to spectral form by computing the eigen-decomposition and use the sorted eigen-values of C as permutation invariant representation vector. Gregoire Montavon et. al [78] introduce another solution in spatial domain by reordering rows of C according to their norms. To deal with larger dimensionality, the paper also introduces the idea of adding a random noise to row norms before sorting to generate more sorted Coulomb matrices per molecule for data augmentation. Among three methods, randomly sorted Coulomb representation produces the best performance for the problem of atomization energy prediction [78].

EigenPool (Eigenvector-based Pooling Layers) [79] develops permutation invariance operators by sorting eigenvalues in spectral domain. EigenPool is a graph convolutional network that consists of coarse-to-fine groups of

convolution layers and pooling layers. The pooling layers have two steps of: (1) spectral clustering to divide the graph at the level l into several non-overlapping clusters and then (2) nodes with the same cluster are aggregated by summing their feature vectors. Since the spectral clustering always gives the same clustering results for all node permutation, the final graph representation is permutation invariant. Spectral graph clustering first converts initial graphs in spatial domain into spectral domain by applying graph Fourier transform on graph Laplacian matrix. The top K eigen-vectors are sorted by their corresponding eigenvalues in an ascending order to form a matrix $\boldsymbol{U}$ of $N_v \times K$, whose rows represent the clustering features of the corresponding nodes in spectral space. A clustering algorithm, such as K-means, can be applied to group nodes into clusters. Since the clustering feature of each node is unchanged under permutation transformation, the clustering results are completely independent of node order.

Graph isomorphism problem has a strong connection to graph ordering that is to find an optimal order for all nodes of a graph [80]. Graph ordering can be thought as a way to convert graphs into a canonical form, which is identical for any isomorphic graphs, rendering an invariance to node permutation. [80] proposes a graph ordering neural network, called Deep Ordering Network with Reinforcement Learning (DON-RL), towards graph visualization applications, where isomorphic graphs should have the consistent and unique visualization in a space. The key intuition is to find an optimal node ordering that maximizes the following accumulated locality score $f(\pi) = \sum_{0 \le \pi_u - \pi_v \le w} f^{\text{sim}}(v, u)$, where u and v are two ordered nodes in a windows of size w and $f^{\text{sim}}(v, u)$ is a similarity function. To overcome the combinatorial explosion of training space, DON-RL is trained to predict the next node, which should be added to the current ordered node subset of V. The selected node should be one that maximizes the accumulated locality score. By adding each node from V to the subset until all nodes are placed, a node ordering is obtained. It can be seen that DON-RL is similar to PATCHY-SAN[24] in terms of automatically learning node ordering, but unlike [24], optimization is obtained via more advanced Reinforcement Learning framework in DON-RL.

The benefit of ordering-based approach compared with aggregation function is to allow much information pass through the pooling/sorting layers. This helps to construct more distinctive and meaningful features for downstream tasks. The main drawback of this approach is more computation cost is required to arrange sequences in an expected order. Sometimes, due to noise and approximation errors (e.g., of eigen-decomposition implementation or optimization algorithms), sorting results are not always the same as in theory. However, ordering-based approach, along with aggregation functions, are still effective solutions for building graph learning systems invariant to node permutation.

3. Histogram-based methods

Histograms are fundamental and popular representations across many fields. Basically, a histogram is a frequency vector, where each index is associated with an object of interest and the corresponding element represents the number of times that object occurs. In graph learning, objects are substructures including subgraphs (also known as graphlets, motifs or graph fragments) [81] or special structures such as shortest paths, subtrees, random walks [24], cycles or cliques [82]. Given a dictionary $\boldsymbol{\Sigma}$ of common substructures, the idea of histogram-based approach to encode a graph is to count the frequency of substructures in the graph and map to a fixed size and ordered representation vector. Although graph isomorphism impacts on the order/label of nodes in graphs, local structures are unchanged and hence frequency vectors are consistent for isomorphic graphs. In other words, two isomorphic graphs share the same substructure distributions. However, since this approach does not take positional relationship of substructures into account, two graphs with different local substructure arrangement still have the same representation vector.

Substructure counting techniques are common in graph kernel methods that are based on idea of designing a kernel function to measure the similarity between any two graphs and then applying kernel methods, e.g., SVMs [56], to perform machine learning tasks. An example is the early work of graphlets [55], where the kernel function is defined as inner product between two frequency vectors, which count the frequency of graphlets with size of $k \le 5$, namely $k_{\text{graphlet}}(G, G') = \bar{\phi}_G(\boldsymbol{g})^{\mathsf{T}} \bar{\phi}_G(\boldsymbol{g}')$. Other substructures with lower computation cost can be used, for example, subtree patterns [57] or random walks [57]. Computing frequency of a graphlet with size of k requires to enumerate all graphlets of size k in a given input graph. The complexity is $O\left(N_v^k\right)$, which is expensive. Sampling is an alternative solution which draws a sufficient number of graphlets randomly from the input graph. The expectation is that if the number of drawn samples is large enough, the approximate graphlet distribution is close to the true distribution. By choosing the proper number of samples, this approach allows to balance between the accuracy and the efficiency in practical applications, especially ones related to large graphs.

Instead of building a huge graphlet dictionary from training data and suffering the expensive graph encoding cost due to too many graphlets need to be counted [57],

Dimistris Floros et. al [83] use a predefined dictionary of limited size of 16 graphlets to speed up the process. For any input graph with N_v vertices, an vertex-graphlet incident matrix $\in \mathbb{R}^{N_v \times |\mathbf{\Sigma}|}$ to describe the occurrence frequency of a graphlet in $\mathbf{\Sigma}$ at the i^{th} node. As a result, any two nodes at the same orbit (a subset of vertices symmetric under permutations [82, 83]) have the same frequency vector, resulting in orbit-invariance.

With the dominance of message passing based GNNs in literature, [82] introduces GSN (Graph Structure Networks) to overcome the limitations of conventional GNNs. Although substructures play an important role in many studies of networks, e.g., molecules or proteins, existing GNNs mostly discover the features of a node itself or the aggregated features of its neighbors rather than its local structures. As the result, the expressive power of GNNs is only equivalent to Weisfeiler-Lehman test. To leverage the source of rich information from substructures, GSN proposes to integrate a subgraph isomorphism counting function for node feature encoding. Let $\mathbf{\Sigma}$ be the collection of small substructures, e.g., cycles of fixed length or cliques. For a substructure $g^{\mathbf{\Sigma}} \in \mathbf{\Sigma}$, let $\text{Aut}\left(g^{\mathbf{\Sigma}}\right)$ denote the set of all unique automorphism of $g^{\mathbf{\Sigma}}$. For any input graph G, support that $g \in G$ is a subgraph of G and isomorphic to $g^{\mathbf{\Sigma}}$ via a bijection mapping f^{iso} from V_g to $V_{g^{\Sigma}}$, the structural feature at node v with respect to substructure $g^{\mathbf{\Sigma}}$ is defined as:

$$\boldsymbol{x}_v^{g^{\Sigma},i} = \frac{\left|\left\{g \simeq g^{\mathbf{\Sigma}} : v \in g \;\text{ s.t. }\; f^{\text{iso}}(v) \in O_{g^{\Sigma},i}\right\}\right|}{\left|\text{Aut}\left(g^{\mathbf{\Sigma}}\right)\right|}$$

$$\boldsymbol{x}_v^{g^{\Sigma}} = \left[\boldsymbol{x}_v^{g^{\Sigma},1}, \boldsymbol{x}_v^{g^{\Sigma},2}, ..., \boldsymbol{x}_v^{g^{\Sigma},N_{g^{\Sigma}}}\right]$$

where $O_{g^{\Sigma},i}$ is the i^{th} unique element of the quotient of the automorphism acting on the substructure $g^{\mathbf{\Sigma}}$. By enumerating all substructures $g_i^{\mathbf{\Sigma}}$ in $\mathbf{\Sigma}$ and concatenating their structural features, we obtain the vertex structural feature of a node v as: $\boldsymbol{x}_v = \left[\boldsymbol{x}_v^{g_1^{\Sigma}}, \boldsymbol{x}_v^{g_2^{\Sigma}}, ..., \boldsymbol{x}_v^{g_{|\Sigma|}^{\Sigma}}\right]$. Another edge-related structural feature is introduced in [82]. These structural features along with conventional hidden vertex features $\boldsymbol{h}_v^{(l)}$ are plugged into a message passing framework [84] defined in a general manner. GSNs are proven to be at least as powerful as MPNNs (Message Passing Neural Networks) [84] and 1-WL test.

Unlike nodes or edges, which are basic elements in graphs, substructures are higher-order components containing crucial topological information and unique characteristics of a graph. These components can help to distinguish different graphs better than many permutation invariance techniques [82]. But recognizing substructures of interest in a large graph is not a trivial task because substructures are essentially graphs and also affected by isomorphism. Furthermore, enumerating large graphs for higher-order substructure counting is time consuming, hindering the popularity of this approach and its applicability in practice. However, recent advances in discovering theoretical expressive power [85] of substructures and their practical representation capacity of real complex topology reveal their potential to develop more powerful graph representation systems.

4. Permutation sampling

Another line of studies, which does not explicitly design permutation invariance functions but forces instead models to deal with permutations, is permutation sampling. The intuition is to train models on data, which is randomly permuted. It seems to be unsound in theory at the first glance but it is actually associated with exchangeability in probability distributions [86, 87].

One of general form of permutation sampling is Janossy pooling [86] that represents permutation invariant functions as the average of a permutation-sensitive function acting on all reorderings of input sequences. Let $f^{\text{arb}}(\cdot)$ denote arbitrary function that can accept any variable-size finite input sequences $\boldsymbol{h}$ with the length of $|\boldsymbol{h}|$, and π is a permutation in a set $\Pi_{|\boldsymbol{h}|}$ of all permutation of $\boldsymbol{h}$.

$$f(|\boldsymbol{h}|, \boldsymbol{h}) = \frac{1}{|\boldsymbol{h}|!} \sum_{\pi \in \Pi_{|\boldsymbol{h}|}} f^{\text{arb}}(|\boldsymbol{h}|, \boldsymbol{h}) \tag{9}$$

Theoretically, $\Pi_{|\boldsymbol{h}|}$ and sum are invariant to the order of input sequences and hence $f(|\boldsymbol{h}|, \boldsymbol{h})$ becomes a permutation invariant function. Computing the sum over permutation set $\Pi_{|\boldsymbol{h}|}$ is expensive, an approximation is required for computational tractability. In [86], the authors propose three methods of approximation: 1) convert $\boldsymbol{h}$ into a canonical form, e.g., sorting (Subsec. IV.2), before feeding it into $f^{\text{arb}}(\cdot)$; 2) still employ Eq. 9 but on some small first k elements of input sequences; and 3) evaluate the sum via some random permutation samples from $\Pi_{|\boldsymbol{h}|}$ instead of all its elements. These methods offer significant computational savings and can be implemented in practice. Relational Pooling [88] is an extension of Janossy Pooling. Although they share the same idea of summing over all permutation set, Relational Pooling works directly on the structural matrix and the feature matrix $f(G) = \frac{1}{|V|!} \sum_{\pi \in \Pi_{|V|}} f^{\text{arb}}\left(\boldsymbol{\mathcal{A}}_{\pi,\pi}, \boldsymbol{X}_{\pi}\right)$. The experiments in the paper shows that RP-GNN, which is actually GNN with Relational Pooling, is more expressive than Weisfeiler-Lehman test [88]. It means that RP-GNN can distinguish pairs of non-isomorphic graphs that cannot be recognized by Weisfeiler-Lehman test.

Local Relational Pooling (LRP) [81] adopts the framework of Relational Pooling [88] but applies to local structures of egonets. An egonet of a depth r at a specific node v is an induced subgraph including v and its neighborhood $\mathcal{N}_v^r$ that consists of all nodes within the distance of r. For Relational Pooling, we have to consider all permutations of nodes in $\mathcal{N}_v^r$. However, since egonets are rooted graphs, the computational cost reduces to a subset of $\mathcal{N}_v^r$, which is equivalent to breath-first-search (BFS) at v. Formally, Local Relational Pooling is defined as: $f(v) = \frac{1}{|V^{\text{BFS}}|!} \sum_{\pi \in \Pi_{|V^{\text{BFS}}|}} g\left(\mathcal{A}^v_{\pi,\pi}\right)$. To further improve the speed, only k ordered nodes in BFS list is considered rather than taking all nodes in the egonets. For a network with a bounded degree, the complexity of the pooling layer grows linearly with respect to the number of vertices.

Edo Cohen-Karlik et. al [89] adopt the idea of [86] by feeding randomly permuted input sequences but the authors apply it to building a regularizer towards permutation invariance for RNNs. The goal of RNNs is to train a function: $f : \mathcal{S} \times \mathcal{X} \rightarrow S$ starting from an initial state $\boldsymbol{s}_0$. The state at the time $t+1$ is updated via the previous state as: $\boldsymbol{s}_{t+1} = f(\boldsymbol{s}_t, \boldsymbol{x}_t)$. To learn a permutation invariance function f, which produces the same output regardless of any input order, [86] introduces subset invariance regularization (SIRE). For a training subset with the same output, SIRE should be zero otherwise it should penalize for different outputs. More interestingly, the authors prove that this can be obtained efficiently by swapping any two elements $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ during training, resulting in the following regularization term: $\mathbb{E}_{\boldsymbol{s} \in \mathcal{S}}\left[(f(\boldsymbol{s}, \boldsymbol{x}_1, \boldsymbol{x}_2) - f(\boldsymbol{s}, \boldsymbol{x}_2, \boldsymbol{x}_1))^2\right]$, where $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are two new inputs and $f\left(\boldsymbol{s}, \boldsymbol{x}_i, \boldsymbol{x}_j\right) = f\left(f\left(\boldsymbol{s}, \boldsymbol{x}_i\right), \boldsymbol{x}_j\right)$. Moreover, RNNs trained with SIRE use fewer parameters than Deep Set based approach [66].

Some studies [71, 90] follow the strategy of permutation sampling but reduce the computational cost by drawing only one sample instead of a large batch of samples. Deep Collective Inference (DCI) [90] is an example, which implements a RNN-based framework for collective inference. Namely, for a particular node, DCI transforms the target node and its neighbors into an unordered node sequence before feeding into a RNN to predict corresponding class outputs. Meanwhile, GraphSAGE [71] offers to uniformly sample the fixed number of nodes in neighborhood in its aggregation step.

Permutation sampling approach can be applied to learn permutation invariance in many complex models, such as LSTMs or RNNs, for which it is challenging to explicitly design a permutation invariance function. Additionally, sampling allows significant computational savings, which is especially helpful for training heavy models. Although this approach is still gaining less attention from community in both theoretical and practical research, recent achievements [71, 90] are fairly promising.

V. DISCUSSION

In this section, we briefly summarize and analyze existing approaches and open issues for graph structure and permutation invariance learning:

Graph structure learning: *Structural matrix factorization* and *spectral domain-based methods* are classical approaches and usually face the problem of high computational cost, hindering their applicability in real-world scenarios. *Node embedding* are node-level models without parameter sharing and therefore they are not suitable for popular applications related to entire graphs. Two existing approaches, which most graph learning studies are focusing on, are ones based on *graph coloring* and *aggregation operators*. These techniques can be observed in recent state-of-the art graph neural networks nowadays due to their efficiency and accuracy. However, they usually rely on simple and easy-to-compute substructures and therefore discovering higher and complex structures should be considered and explored. For *motif-based methods*, although they are very powerful to represent local topology, their expensive cost is limiting the number of research on this direction. But integrating motifs into existing graph learning systems will be a potential to improve the power of graph structure learning.

Permutation invariant learning: *Histogram-based methods* are widely used in kernel methods but have not been studied in recent years because of their high cost to find patterns of interest in large graphs to construct histograms. By contrast, *aggregation function* and *ordering-based approaches* are dominating the literature to date due to better balance between accuracy and speed. Compared with ordering-based approach, aggregation functions are much faster but they usually have the problem of information loss. However, recent theoretical and practical results show that aggregation functions with higher-order tensors can expand the capacity of distinguishing isomorphic graphs, shining a light onto building better permutation invariance systems. For complex graph learning systems, which are infeasible or costly to integrate permutation invariance functions by design, training permutation invariance models using *permutation sampling* is a potential direction with some promising results [71, 90]. However, since there are a few studies following this direction, further research should be conducted to comprehensively verify and evaluate its effectiveness in both theoretical and practical aspects.

Some open issues of both graph structure and isomorphism invariance learning can be listed as follows:

- **Scalability:** Designing systems to work efficiently on large graphs is not only a big challenging for graph learning but also for any graph-related problems. Basically, the number of substructures and patterns increase according to graph size, resulting in a significant increase in the cost of enumerating nodes and encoding local structures. Many graph structure learning methods depend on fundamental structure matrices, which become inefficient on such large graphs. For isomorphism test, comparing two large graphs to verify whether they are isomorphic or not is expensive as well. Therefore, there is an open question about developing better algorithms that balance between speed and accuracy for large-scale graphs.
- **Complex Graphs**: The majority of graph structure learning and permutation invariance studies are conducted on undirected homogeneous graphs. But more sophisticated graphs can be seen in real-world applications. Some examples are directed graphs that imply the direction of the relationship between entities or heterogeneous groups whose nodes and edges can have different types or dynamic graphs whose nodes and edges change over time. All such types of graphs usually require algorithms with proper designs and optimizations in order to maintain good performance in real-world scenarios.
- **Higher-order substructures:** High-order substructures often contain more meaningful information. This not only allows to describe complicated graph topology but also helps to better distinguish more non-isomorphic graphs. However, since substructures are actually graphs, recognizing and encoding them require more cost with respect to their structure complexity. With the lack of efficient methods to process higher-order substructures, most existing graph learning models are still based on simple graph elements for efficiency. But we believe that, with the rapid growth and development of deep learning in recent years, discoveries and advances on higher-order substructures can bring us more powerful graph learning systems in the near future.

VI. CONCLUSION

This survey has given a comprehensive review on graph learning advances in recent years. Particularly, it focuses on two fundamental but important topics of graph learning, which are graph structure learning and permutation invariance learning. For each topic, we examine existing approaches and provide detailed analysis on its pros and cons. Future directions of these topics are also discussed and explained. We hope that this survey can help researchers to better understand these topics as well as the broader field of graph learning, one of hot areas in machine learning nowadays.

REFERENCES

[1] F. Xia, K. Sun, S. Yu, A. Aziz, L. Wan, S. Pan, and H. Liu, "Graph learning: A survey," *IEEE Transactions on Artificial Intelligence*, pp. 1–1, 2021.

[2] Z. Zhang, P. Cui, and W. Zhu, "Deep learning on graphs: A survey," 2018.

[3] J. Leskovec, "Representation Learning on Networks," *WWW-18 Tutorial*, 2018. [Online]. Available: http://snap.stanford.edu/proj/embeddings-www/

[4] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An End-to-End Deep Learning Architecture for Graph Classification," in *Proceedings of the 32nd Conference on Artificial Intelligence (AAAI)*, New Orleans, Louisiana, USA, February 2-7 2018, pp. 4438–4445.

[5] W. Cao, Z. Yan, Z. He, and Z. He, "A comprehensive survey on geometric deep learning," *IEEE Access*, vol. 8, pp. 35 929–35 949, 2020.

[6] M. Bronstein, J. Bruna, Y. Lecun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: Going beyond euclidean data," *IEEE Signal Processing Magazine*, vol. 34, no. 4, pp. 18–42, Jul. 2017.

[7] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation Learning on Graphs: Methods and Applications," *IEEE Data Eng. Bull.*, vol. 40, no. 3, pp. 52–74, 2017.

[8] H. Cai, V. Zheng, and K. Chang, "A comprehensive survey of graph embedding: Problems, techniques, and applications," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, pp. 1616–1637, 2018.

[9] M. Kinderkhedia, "Learning representations of graph data: A survey," 2019.

[10] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, and M. Sun, "Graph neural networks: A review of methods and applications," *CoRR*, 2018.

[11] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A Comprehensive Survey on Graph Neural Networks," *CoRR*, 2019.

[12] Z. Chen, F. Chen, L. Zhang, T. Ji, K. Fu, L. Zhao, F. Chen, and C. Lu, "Bridging the gap between spatial and spectral domains: A survey on graph neural networks," *CoRR*, 2020.

[13] K. Borgwardt, E. Ghisu, F. Llinares-López, L. O'Bray, and B. Rieck, "Graph kernels: State-of-the-art and future challenges," *Foundations and Trends in Machine Learning*, vol. 13, no. 5-6, pp. 531–712, 2020.

[14] G. Nikolentzos, G. Siglidis, and M. Vazirgiannis, "Graph kernels: A survey," *CoRR*, vol. abs/1904.12218, 2019.

[15] Y. Zhu, W. Xu, J. Zhang, Q. Liu, S. Wu, and L. Wang, "Deep graph structure learning for robust representations: A survey," *CoRR*, 2021.

[16] C. Zhuang and Q. Ma, "Dual graph convolutional networks for graph-based semi-supervised classification," in *Proceedings of World Wide Web Conference (WWW)*, Lyon, France, 2018, pp. 499–508.

[17] S. Cao, W. Lu, and Q. Xu, "Deep neural networks for learning graph representations," *Proceedings of the Conference on Artificial Intelligence (AAAI)*, vol. 30, no. 1, Feb. 2016.

[18] Y. Yang, H. Chen, and J. Shao, "Triplet enhanced autoencoder: Model-free discriminative network embedding," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI)*, Macao, China, August 10-16 2019, pp. 5363–5369.

[19] F. R. K. Chung, *Spectral Graph Theory*, Providence, RI, 1997.

[20] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[21] D. Bieber, "The Weisfeiler-Lehman Isomorphism test," 2019. [Online]. Available: https://davidbieber.com/post/2019-05-10-weisfeiler-lehman-isomorphism-test/

[22] B. Weisfeiler and A. Leman, "The reduction of a graph to canonical form and the algebra which appears therein," *Nauchno-Technicheskaya Informatsia*, 1968.

[23] L. Babai, P. Erdös, and S. M. Selkow, "Random graph isomorphism," *SIAM J. Comput.*, vol. 9, no. 3, pp. 628–635, 1980.

[24] M. Niepert, M. Ahmed, and K. Kutzkov, "Learning Convolutional Neural Networks for Graphs," in *Proceedings of The 33rd International Conference on Machine Learning (ICML)*, vol. 48, New York, New York, USA, 20–22 Jun 2016, pp. 2014–2023.

[25] M. D. G. Mallea, P. Meltzer, and P. J. Bentley, "Capsule Neural Networks for Graph Classification using Explicit Tensorial Graph Representations," *arXiv*, 2019.

[26] H. Jin, W. Yu, and S. Li, "Graph regularized nonnegative matrix tri-factorization for overlapping community detection," *Physica A: Statistical Mechanics and its Applications*, vol. 515, no. C, pp. 376–387, 2019.

[27] M. Huang, Q. Jiang, Q. Qu, and A. Rasool, "An overlapping community detection approach in ego-splitting networks using symmetric nonnegative matrix factorization," *Symmetry*, vol. 13, no. 5, 2021.

[28] G. Ceddia, P. Pinoli, S. Ceri, and M. Masseroli, "Matrix factorization-based technique for drug repurposing predictions," *IEEE J. Biomed. Health Informatics*, vol. 24, no. 11, pp. 3162–3172, 2020.

[29] A. Mitra, P. Vijayan, S. Parthasarathy, and B. Ravindran, "A unified non-negative matrix factorization framework for semi supervised learning on graphs," in *Proceedings of International Conference on Data Mining (SDM)*, Cincinnati, Ohio, USA, May 7-9 2020, pp. 487–495.

[30] F. Ye, C. Chen, and Z. Zheng, "Deep Autoencoder-like Nonnegative Matrix Factorization for Community Detection," in *Proceedings of the 27th International Conference on Information and Knowledge Management (CIKM)*, USA, 2018, pp. 1393–1402.

[31] J. Qiu, Y. Dong, H. Ma, J. Li, C. Wang, K. Wang, and J. Tang, "NetSMF: Large-scale network embedding as sparse matrix factorization," in *The World Wide Web Conference (WWW)*, San Francisco, CA, USA, May 13-17 2019, pp. 1509–1520.

[32] W. Jin, Y. Ma, X. Liu, X. Tang, S. Wang, and J. Tang, "Graph structure learning for robust graph neural networks," in *26th Conference on Knowledge Discovery and Data Mining (SIGKDD)*, Virtual Event, CA, USA, August 23-27 2020, pp. 66–74.

[33] L. Franceschi, M. Niepert, M. Pontil, and X. He, "Graph structure learning for GCNs," *A workshop paper at International Conference on Learning Representations (ICLR)*, 2019.

[34] W. L. Hamilton, "Graph representation learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 14, no. 3, pp. 1–159, 2020.

[35] M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu, "Asymmetric transitivity preserving graph embedding," in *Proceedings of International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, San Francisco, CA, USA, August 13-17 2016, pp. 1105–1114.

[36] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, 2016, pp. 855–864.

[37] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, 2014, pp. 701–710.

[38] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "LINE: large-scale information network embedding," in *Proceedings of the 24th International Conference on World Wide Web (WWW)*, Florence, Italy, May 18-22 2015, pp. 1067–1077.

[39] J. Zhao, X. Wang, C. Shi, B. Hu, G. Song, and Y. Ye, "Heterogeneous graph structure learning for graph neural networks," in *Conference on Artificial Intelligence (AAAI)*, February 2-9 2021, pp. 4697–4705.

[40] A. Micheli, "Neural network for graphs: A contextual constructive approach," *IEEE Transactions on Neural Networks*, vol. 20, no. 3, pp. 498–511, 2009.

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

[42] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An end-to-end deep learning architecture for graph classification," in *AAAI*, 2018.

[43] B. Jiang, Z. Zhang, D. Lin, J. Tang, and B. Luo, "Semi-supervised learning with graph learning-convolutional networks," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, Long Beach, CA, USA, June 16-20 2019, pp. 11 313–11 320.

[44] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," *International Conference on Learning Representations (ICLR)*, 2017.

[45] Z. Zhang, P. Cui, J. Pei, X. Wang, and W. Zhu, "Eigen-gnn: A graph structure preserving plug-in for gnns," *CoRR*, 2020.

[46] F. Errica, M. Podda, D. Bacciu, and A. Micheli, "A fair comparison of graph neural networks for graph classification," in *Proceedings of International Conference on Learning Representations (ICLR)*, 2020.

[47] D. Shuman, S. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains," *IEEE Signal Processing Magazine*, vol. 30, pp. 83–98, 2013.

[48] A. Sandryhaila and J. M. F. Moura, "Discrete signal processing on graphs: Graph Fourier transform," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013, pp. 6167–6170.

[49] S. Chen, R. Varma, A. Sandryhaila, and J. Kovačević, "Discrete signal processing on graphs: Sampling theory," *IEEE Transactions on Signal Processing*, vol. 63, no. 24, pp. 6510–6523, 2015.

[50] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," in *International Conference on Learning Representations (ICLR)*, Banff, AB, Canada, April 14-16 2014.

[51] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Advances in Neural Information Processing Systems (NIPS)*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29, 2016.

[52] R. Levie, F. Monti, X. Bresson, and M. M. Bronstein, "Cayleynets: Graph convolutional neural networks with complex rational spectral filters," *IEEE Transactions on Signal Processing*, vol. 67, no. 1, pp. 97–109, 2019.

[53] S. Yu, Y. Feng, D. Zhang, H. D. Bedru, B. Xu, and F. Xia, "Motif discovery in networks: A survey," *Computer Science Review*, vol. 37, p. 100267, 2020.

[54] N. Przulj, D. G. Corneil, and I. Jurisica, "Modeling interac-

tome: scale-free or geometric?." *Bioinform.*, vol. 20, no. 18, pp. 3508–3515, 2004.

[55] N. Shervashidze, S. Vishwanathan, T. Petri, K. Mehlhorn, and K. Borgwardt, "Efficient graphlet kernels for large graph comparison," in *Proceedings of International Conference on Artificial Intelligence and Statistics (AISTAT)*, vol. 5, Clearwater Beach, Florida USA, 16–18 Apr 2009, pp. 488–495.

[56] B. Schölkopf and A. J. Smola, *Learning with kernels : support vector machines, regularization, optimization, and beyond*, 2002.

[57] P. Yanardag and S. Vishwanathan, "Deep Graph Kernels," in *Proceedings of International Conference on Knowledge Discovery and Data Mining (KDD)*, USA, 2015, pp. 1365–1374.

[58] J. Ramon and T. Gärtner, "Expressivity versus efficiency of graph kernels," in *Proceedings of European Conference on Machine Learning (ECML/PKDD)*, Cavtat-Dubrovnik, Croatia, Sep 22-23 2003, pp. 65–74.

[59] P. Mahé and J. Vert, "Graph kernels based on tree patterns for molecules," *Mach. Learn.*, vol. 75, no. 1, pp. 3–35, 2009.

[60] T. Gï¿œrtner, P. A. Flach, and S. Wrobel, "On graph kernels: Hardness results and efficient alternatives." in *COLT*, vol. 2777, 2003, pp. 129–143.

[61] R. A. Rossi, N. K. Ahmed, E. Koh, S. Kim, A. Rao, and Y. Abbasi-Yadkori, "A structural graph representation learning framework," in *International Conference on Web Search and Data Mining (WSDM)*, Houston, TX, USA, 2020, pp. 483–491.

[62] X. Li, W. Wei, X. Feng, X. Liu, and Z. Zheng, "Representation learning of graphs using graph convolutional multilayer networks based on motifs," *Neurocomputing*, vol. 464, pp. 218–226, 2021.

[63] G. Abuoda, G. D. F. Morales, and A. Aboulnaga, "Link prediction via higher-order motif features," in *Proceedings of Machine Learning and Knowledge Discovery in Databases - European Conference (ECML) (PKDD)*, vol. 11906, Würzburg, Germany, Sep 16-20 2019, pp. 412–429.

[64] B. D. McKay and A. Piperno, "Practical graph isomorphism, II," *J. Symb. Comput.*, vol. 60, pp. 94–112, 2014.

[65] P. Velickovic, "Theoretical foundations of graph neural networks," *CST Wednesday Seminar*, Feb 2021.

[66] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola, "Deep Sets," in *Advances in Neural Information Processing Systems (NIPS)*, 2017, pp. 3391–3401.

[67] E. Wagstaff, F. Fuchs, M. Engelcke, I. Posner, and M. A. Osborne, "On the limitations of representing functions on sets," in *Proceedings of International Conference on Machine Learning (ICML)*, vol. 97, 2019, 9-15 June 2019, Long Beach, California, USA, 2019, pp. 6487–6494.

[68] H. Maron, H. Ben-Hamu, N. Shamir, and Y. Lipman, "Invariant and equivariant graph networks," in *International Conference on Learning Representations (ICLR)*, New Orleans, LA, USA, May 6-9, 2019.

[69] H. Maron, E. Fetaya, N. Segol, and Y. Lipman, "On the universality of invariant networks," in *Proceedings of International Conference on Machine Learning (ICML)*, vol. 97, Long Beach, California, USA, June, 9-15 2019, pp. 4363–4371.

[70] N. Keriven and G. Peyré, "Universal invariant and equivariant graph neural networks," in *Advances in Neural Information Processing Systems (NIPS)*, vol. 32, 2019.

[71] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive Representation Learning on Large Graphs," in *Advances in Neural Information Processing Systems (NIPS)*, 2017, pp. 1024–1034.

[72] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How Powerful are Graph Neural Networks?" in *the 7th International Conference on Learning Representations (ICLR)*, USA, May 6-9 2019.

[73] J. Lee, Y. Lee, J. Kim, A. Kosiorek, S. Choi, and Y. W. Teh, "Set transformer: A framework for attention-based permutation-invariant neural networks," in *Proceedings of International Conference on Machine Learning (ICML)*, vol. 97, Jun, 09-15 2019, pp. 3744–3753.

[74] H. Zhu, F. Feng, X. He, X. Wang, Y. Li, K. Zheng, and Y. Zhang, "Bilinear graph neural network with neighbor interactions," in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, (IJCAI)*, 2020, pp. 1452–1458.

[75] S. Verma and Z.-L. Zhang, "Graph Capsule Convolutional Neural Networks," *arXiv*, 2018.

[76] P. Meltzer, M. D. G. Mallea, and P. J. Bentley, "PiNet: A Permutation Invariant Graph Neural Network for Graph Classification," *CoRR*, 2019.

[77] M. Rupp, A. Tkatchenko, K.-R. Müller, and O. A. von Lilienfeld, "Fast and accurate modeling of molecular atomization energies with machine learning," *Physical Review Letters*, vol. 108, p. 058301, 2012.

[78] G. Montavon, K. Hansen, S. Fazli, M. Rupp, F. Biegler, A. Ziehe, A. Tkatchenko, A. Lilienfeld, and K.-R. Müller, "Learning invariant representations of molecules for atomization energy prediction," in *Advances in Neural Information Processing Systems (NIPS)*, vol. 25, 2012.

[79] Y. Ma, S. Wang, C. C. Aggarwal, and J. Tang, "Graph convolutional networks with eigenpooling," in *Proceedings of International Conference on Knowledge Discovery & Data Mining, (KDD)*, Anchorage, AK, USA, August, 4-8 2019, pp. 723–731.

[80] K. Zhao, Y. Rong, J. X. Yu, J. Huang, and H. Zhang, "Graph ordering: Towards the optimal by learning," *CoRR*, 2020.

[81] Z. Chen, L. Chen, S. Villar, and J. Bruna, "Can graph neural networks count substructures?" in *Advances in Neural Information Processing Systems (NeurIPS) 33*, December 6-12 2020.

[82] G. Bouritsas, F. Frasca, S. Zafeiriou, and M. M. Bronstein, "Improving graph neural network expressivity via subgraph isomorphism counting," *CoRR*, 2020.

[83] D. Floros, N. Pitsianis, and X. Sun, "Fast graphlet transform of sparse graphs," in *2020 IEEE High Performance Extreme Computing Conference, (HPEC)*, Waltham, MA, USA, Sep 22-24 2020, pp. 1–8.

[84] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *Proceedings of International Conference on Machine Learning, ICML*, vol. 70, Sydney, NSW, Australia, August, 6-11 2017, pp. 1263–1272.

[85] V. Arvind, F. Fuhlbrück, J. Köbler, and O. Verbitsky, "On Weisfeiler-Leman invariance: Subgraph counts and related graph properties," in *Proceedings of Fundamentals of Computation Theory (FCT)*, vol. 11651, Copenhagen, Denmark, August 12-14 2019, pp. 111–125.

[86] R. L. Murphy, B. Srinivasan, V. A. Rao, and B. Ribeiro, "Janossy pooling: Learning deep permutation-invariant functions for variable-size inputs," in *International Conference on Learning Representations (ICLR)*, New Orleans, LA, USA, May, 6-9 2019.

[87] B. Bloem-Reddy and Y. W. Teh, "Probabilistic symmetries and invariant neural networks," *J. Mach. Learn. Res.*, vol. 21, pp. 90:1–90:61, 2020.

[88] R. Murphy, B. Srinivasan, V. Rao, and B. Ribeiro, "Relational pooling for graph representations," in *Proceedings of International Conference on Machine Learning (ICML)*,

vol. 97, 09–15 Jun 2019, pp. 4663–4673.
[89] E. Cohen-Karlik, A. B. David, and A. Globerson, "Regularizing towards permutation invariance in recurrent models," in *Advances in Neural Information Processing Systems (NIPS)*, Dec, 6-12 2020.
[90] J. Moore and J. Neville, "Deep collective inference," in *Proceedings of Conference on Artificial Intelligence (AAAI)*, San Francisco, California, USA, Feb, 4-9 2017, pp. 2364–2372.

Tuyen Ho Thi Thanh is a PhD student in Computer Science at the University of Science – Vietnam National University, Ho Chi Minh City, Vietnam. She has been teaching courses related to Computer Science program since 2013. She is currently a lecturer and a member of strong research team at the School of Technology and Design - University of Economics Ho Chi Minh City.
Email: tuyenhtt@ueh.edu.vn