Chapter 13
Graph Neural Networks: Graph Matching

Xiang Ling, Lingfei Wu, Chunming Wu and Shouling Ji

Abstract The problem of graph matching that tries to establish some kind of structural correspondence between a pair of graph-structured objects is one of the key challenges in a variety of real-world applications. In general, the graph matching problem can be classified into two categories: i) the classic graph matching problem which finds an optimal node-to-node correspondence between nodes of a pair of input graphs and ii) the graph similarity problem which computes a similarity metric between two graphs. While recent years have witnessed the great success of GNNs in learning node representations of graphs, there is an increasing interest in exploring GNNs for the graph matching problem in an end-to-end manner. This chapter focuses on the state of the art of graph matching models based on GNNs. We start by introducing some backgrounds of the graph matching problem. Then, for each category of graph matching problem, we provide a formal definition and discuss state-of-the-art GNN-based models for both the classic graph matching problem and the graph similarity problem, respectively. Finally, this chapter is concluded by pointing out some possible future research directions.

Xiang Ling
Department College of Computer Science and Technology, Zhejiang University, e-mail: lingxiang@zju.edu.cn

Lingfei Wu
JD.COM Silicon Valley Research Center, e-mail: lwu@email.wm.edu

Chunming Wu
Department College of Computer Science and Technology, Zhejiang University, e-mail: wuchunming@zju.edu.cn

Shouling Ji
Department College of Computer Science and Technology, Zhejiang University, e-mail: sji@zju.edu.cn

13.1 Introduction

As graphs are natural and ubiquitous representations for describing sophisticated data structures, the problem of graph matching that tries to establish some kind of structural correspondence between two input graph-structured objects. The graph matching problem is one of the key challenges in a variety of research fields, such as computer vision (Vento and Foggia, 2013), bioinformatics (Elmsallati et al, 2016), cheminformatics (Koch et al, 2019; Bai et al, 2019b), computer security (Hu et al, 2009; Wang et al, 2019i), source/binary code analysis (Li et al, 2019h; Ling et al, 2021), and social network analysis (Kazemi et al, 2015), to name just as few. In particular, recent research advances in graph matching have been closely involved in many real-world applications in the field of computer vision, including visual tracking (Cai et al, 2014; Wang and Ling, 2017), action recognition (Guo et al, 2018a), pose estimation (Cao et al, 2017, 2019), *etc*. In addition to the study in computer vision, graph matching also serves as an important foundation of many other graph-based research tasks, *e.g*., node and graph classification tasks (Richiardi et al, 2013; Bai et al, 2019c; Ok, 2020), graph generation tasks (You et al, 2018b; Ok, 2020), *etc*.

In a broad sense, according to different goals of graph matching in a wide variety of real-world applications, the general graph matching problem can be classified into two categories (Yan et al, 2016) as follows. The first category is the *classic graph matching problem* (Loiola et al, 2007; Yan et al, 2020a) that tries to establish the node-to-node correspondence (and/or even edge-to-edge correspondence) between the pair of input graphs. The second category is the *graph similarity problem* (Bunke, 1997; Riesen, 2015; Ma et al, 2019a) with the purpose of computing a similarity score between two input graphs. Both categories have the same inputs (*i.e*., a pair of input graphs) but with different outputs, whereby the output of the first category is mainly formulated as a correspondence *matrix* while the output of the second category is usually expressed as a similarity *scalar*. From the perspective of outputs, the second graph similarity problem can be viewed as a special case of the first graph matching problem, as the similarity scalar reflects a more coarse-grained correspondence representation of graph matching than the correspondence matrix.

Generally, both categories of the graph matching problem are known to be NP-hard (Loiola et al, 2007; Yan et al, 2020a; Bunke, 1997; Riesen, 2015; Ma et al, 2019a), making both problems computationally infeasible for exact and optimum solutions in large-scale and real-world settings. Given the great importance and inherent difficulty of the graph matching problem, it has been heavily investigated in theory and practice and a huge number of approximate algorithms based on theoretical/empirical knowledge of experts have been proposed to find sub-optimal solutions in an acceptable time. Interested readers are referred to (Loiola et al, 2007; Yan et al, 2016; Foggia et al, 2014; Riesen, 2015) for a more extensive review, as these approximation methods are beyond the scope of this chapter. Unfortunately, despite various approximation methods have been devoted to resolving the graph matching problem for the past decades, it still suffers from the issue of poor scala-

bility as well as the issue of heavy reliance on expert knowledge, and thus remains as a challenging and significant research problem for many practitioners.

More recently, GNNs that attempt to adapt deep learning from image to non-euclidean data (*i.e.*, graphs) have received unprecedented attention to learn informative representation (*e.g.*, node or (sub)graph, *etc.*) of graph-structured data in an end-to-end manner (Kipf and Welling, 2017b; Wu et al, 2021d; Rong et al, 2020c). Hereafter, a surge of GNN models have been presented for learning effective node embeddings for downstream tasks, such as node classification tasks (Hamilton et al, 2017a; Veličković et al, 2018; Chen et al, 2020m), graph classification tasks (Ying et al, 2018c; Ma et al, 2019d; Gao and Ji, 2019), graph generation tasks (Simonovsky and Komodakis, 2018; Samanta et al, 2019; You et al, 2018b) as so on. The great success of GNN-based models on these application tasks demonstrates that GNN is a powerful class of deep learning model to better learn the graph representation for downstream tasks.

Encouraged by the great success of GNN-based models obtained from many other graph-related tasks, many researchers have started to adopt GNNs for the graph matching problem and a large number of GNN-based models have been proposed to improve the matching accuracy and efficiency (Zanfir and Sminchisescu, 2018; Rolínek et al, 2020; Wang et al, 2019g; Jiang et al, 2019a; Fey et al, 2020; Yu et al, 2020; Wang et al, 2020j; Bai et al, 2018, 2020b, 2019b; Xiu et al, 2020; Ling et al, 2020; Zhang, 2020; Wang et al, 2020f; Li et al, 2019h; Wang et al, 2019i). During the training stage, these models try to learn a mapping between the pair of input graphs and the ground-truth correspondence in a supervised learning and thus are more time-efficient during the inference stage than traditional approximation methods. In this chapter, we walk through the recent advances and developments of graph matching models based on GNNs. Particularly, we focus on how to incorporate GNNs into the framework of graph matching/similarity learning and try to provide a systematic introduction and review of state-of-the-art GNN-based methods for both categories of the graph matching problem (*i.e.*, the classic graph matching problem in Section 13.2 and the graph similarity problem in Section 13.3, respectively).

13.2 Graph Matching Learning

In this section, we start by introducing the first category of the graph matching problem, *i.e.*, the classic graph matching problem[1], and provide a formal definition of the graph matching problem. Subsequently, we will focus discussion on state-of-the-art graph matching models based on deep learning as well as more advanced GNNs in the literature.

[1] For simplicity, we represent the classic graph matching problem as the graph matching problem in the following sections of this chapter.

13.2.1 Problem Definition

A graph of size n (*i.e.*, numbers of nodes) can be represented as $\mathcal{G} = (\mathcal{V}, \mathcal{E}, A, X, E)$, in which $\mathcal{V} = \{v_1, \cdots, v_n\}$ denotes the set of nodes (also known as *vertices*), $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ denotes the set of edges, $A \in \{0,1\}^{n \times n}$ denotes the adjacency matrix, $X \in \mathbb{R}^{n \times \cdot}$ denotes the initial feature matrix of nodes, and $E \in \mathbb{R}^{n \times n \times \cdot}$ denotes an optional initial feature matrix of edges.

The purpose of the graph matching problem is to find an optimal node-to-node correspondence between two input graphs, *i.e.*, $\mathcal{G}^{(1)}$ and $\mathcal{G}^{(2)}$. Without loss of generality, we consider the graph matching problem whose two input graphs of equal size[2]. In particular, we provide a formal definition of the graph matching problem in Definition 13.1 as follows and give an example illustration of the node-to-node correspondence in Fig. 13.1.

Definition 13.1 (Graph Matching Problem). Given a pair of input graphs $\mathcal{G}^{(1)} = (\mathcal{V}^{(1)}, \mathcal{E}^{(1)}, A^{(1)}, X^{(1)}, E^{(1)})$ and $\mathcal{G}^{(2)} = (\mathcal{V}^{(2)}, \mathcal{E}^{(2)}, A^{(2)}, X^{(2)}, E^{(2)})$ of equal size n, the graph matching problem is to find a node-to-node correspondence matrix $S \in \{0,1\}^{n \times n}$ (*i.e.*, also called *assignment matrix* and *permutation matrix*) between the two graphs $\mathcal{G}^{(1)}$ and $\mathcal{G}^{(2)}$. Each element $S_{i,a} = 1$ if and only if the node $v_i \in \mathcal{V}^{(1)}$ in $\mathcal{G}^{(1)}$ corresponds to the node $v_a \in \mathcal{V}^{(2)}$ in $\mathcal{G}^{(2)}$.

Intuitively, the resulting correspondence matrix S represents the possibility of establishing a matching relation between any pair of nodes in two graphs. The graph matching problem is known to be NP-hard and has been investigated by formulating it as a quadratic assignment problem (QAP) (Loiola et al, 2007; Yan et al, 2016). We adopt the general form of Lawler's QAP (Lawler, 1963) with constraints as follows since it has been widely adopted in literature.

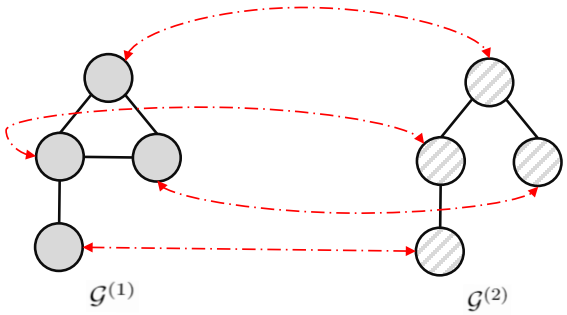

Fig. 13.1 An example illustration of the graph matching problem with two input graphs, *i.e.*, the left graph $\mathcal{G}^{(1)}$ and the right graph $\mathcal{G}^{(2)}$ to be matched. The red dotted lines represent the node-to-node correspondences between the two graphs.

[2] For simplicity, we assume that a pair of input graphs in the graph matching problem have the same number of nodes, but we can extend the problem to a pair of graphs with different number of nodes via adding dummy nodes, which is commonly adopted by graph matching literature Krishnapuram et al (2004).

$$\begin{aligned} \mathbf{s}^* &= \arg\max_{\mathbf{s}} \ \mathbf{s}^\top K \mathbf{s} \\ \text{s.t. } & S\mathbf{1}_n = \mathbf{1}_n \ \& \ S^\top \mathbf{1}_n = \mathbf{1}_n \end{aligned} \tag{13.1}$$

where $\mathbf{s} = \text{vec}(S) \in \{0,1\}^{n^2}$ is the column-wise vectorized version of the assignment matrix S and $\mathbf{1}_n$ is a column vector of length n whose elements are equal to 1. Particularly, $K \in \mathbb{R}^{n^2 \times n^2}$ is the corresponding second-order affinity matrix in which each element $K_{ij,ab}$ measures how well every pair of nodes $(v_i, v_j) \in \mathscr{V}^{(1)} \times \mathscr{V}^{(1)}$ matches $(v_a, v_b) \in \mathscr{V}^{(2)} \times \mathscr{V}^{(2)}$ and can be defined as follows (Zhou and De la Torre, 2012).

$$K_{\text{ind}(i,j),\text{ind}(a,b)} = \begin{cases} c_{ia} & \text{if } i = j \text{ and } a = b, \\ d_{ijab} & \text{else if } A^{(1)}_{i,j} A^{(2)}_{a,b} > 0, \\ 0 & \text{otherwise.} \end{cases} \tag{13.2}$$

where $\text{ind}(\cdot,\cdot)$ is a bijection function that maps a pair of nodes to an integer index, the diagonal element (*i.e.*, c_{ia}) encodes the node-to-node (*i.e.*, first-order) affinity between the node $v_i \in \mathscr{V}^{(1)}$ and the node $v_a \in \mathscr{V}^{(2)}$, and the off-diagonal element (*i.e.*, d_{ijab}) encodes the edge-to-edge (*i.e.*, second-order) affinity between the edge $(v_i, v_j) \in \mathscr{E}^{(1)}$ and the edge $(v_a, v_b) \in \mathscr{E}^{(2)}$.

Another important aspect for the formulation in Equation (13.1) is the constraint, *i.e.*, $S\mathbf{1}_n = \mathbf{1}_n$ and $S^\top \mathbf{1}_n = \mathbf{1}_n$. It demands that the matching output of the graph matching problem, *i.e.*, the correspondence matrix $S \in \{0,1\}^{n \times n}$, should be strictly constrained as a **doubly-stochastic matrix**. Formally the correspondence matrix S is a doubly-stochastic matrix if the summation of each column and each row of it is 1. That is, $\forall i, \sum_j S_{i,j} = 1$ and $\forall j, \sum_i S_{i,j} = 1$. Therefore, the resulting correspondence matrix of the graph matching problem should satisfy the requirement of the doubly-stochastic matrix.

In general, the main challenge in optimizing and solving Equation (13.1) lies in how to model the affinity model as well as how to optimize with the constraint for solutions. Traditional methods mostly utilize pre-defined affinity models with limited capacity (*e.g.*, Gaussian kernel with Euclid distance Cho et al (2010)) and resort to different heuristic optimizations (*e.g.*, graduated assignment (Gold and Rangarajan, 1996), spectral method (Leordeanu and Hebert, 2005), random walk (Cho et al, 2010), *etc.*). However, such traditional methods suffer from poor scalability and inferior performance for large-scale settings as well as a broad of application scenarios (Yan et al, 2020a). Recently, studies on the graph matching are starting to explore the high capacity of deep learning models, which achieve state-of-the-are performance. In the following subsections, we will first give a brief introduction of deep learning based graph matching models and then discuss state-of-the-art graph matching models based on GNNs.

13.2.2 Deep Learning based Models

Aiming at increasing the matching performance, extensive research interest in leveraging high capacity of deep learning models to solve the problem of graph matching has been ignited since Zanfir and Sminchisescu (2018), which introduces an end-to-end deep learning framework for the graph matching problem for the first time and receives the best paper honorable mention award in CVPR 2018[3].

Deep Graph Matching. In (Zanfir and Sminchisescu, 2018), Zanfir and Sminchisescu first relax the graph matching problem of Equation (13.1) with the ℓ_2 constraint as follows.

$$\begin{aligned} \mathbf{s}^* = \arg\max_{\mathbf{s}} \ & \mathbf{s}^\top K \mathbf{s} \\ \text{s.t.} \ & \|\mathbf{s}\|_2 = 1 \end{aligned} \tag{13.3}$$

To solve the problem, they attempt to introduce deep learning techniques to the graph matching and propose an end-to-end training framework with standard differentiable backpropagation and optimization algorithms. The proposed deep graph matching framework first uses the existing pre-trained CNN model (*i.e.*, VGG-16 (Simonyan and Zisserman, 2014b)) to extract node features (*i.e.*, $U^{(1)}$ and $U^{(2)} \in \mathbb{R}^{n\times d}$) and edge features (*i.e.*, $F^{(1)} \in \mathbb{R}^{p\times 2d}$ and $F^{(2)} \in \mathbb{R}^{q\times 2d}$) from the pair of input images in the scenario of computer vision applications. In particular, $F^{(1)}$ and $F^{(2)}$ are row-wise edge feature matrices with p and q as the number of edges in each graph, respectively. As each edge attribute is the concatenation of the start and the end node, the dimension of edge attribute is double $2d$ the dimension of node.

Next, based on extracted node/edge features, it builds the graph matching affinity matrix K via a novel factorization method of graph matching (Zhou and De la Torre, 2012) as follows.

$$\begin{aligned} K &= \lceil \mathrm{vec}(K_p) \rfloor + (G_2 \otimes G_1) \lceil \mathrm{vec}(K_e) \rfloor (H_2 \otimes H_1)^\top \\ &= \left\lceil \mathrm{vec}(U^{(1)} U^{(2)\top}) \right\rfloor + (G_2 \otimes G_1) \left\lceil \mathrm{vec}(F^{(1)} \Lambda F^{(2)}) \right\rfloor (H_2 \otimes H_1)^\top \end{aligned} \tag{13.4}$$

where $\lceil X \rfloor$ denotes a diagonal matrix whose diagonal elements are all X; $\otimes$ denotes the Kronecker product; G_i and H_i $(i = \{1,2\})$ are the node-edge incidence matrices that are recovered from the adjacency matrices $A^{(i)}$, *i.e.*, $A^{(i)} = G_i H_i^\top$ $(i = \{1,2\})$; $K_p \in \mathbb{R}^{n\times n}$ encodes the node-to-node similarity and is directly obtained from the product of two node feature matrices, *i.e.*, $K_p = U^{(1)} U^{(2)\top}$; $K_e \in \mathbb{R}^{p\times q}$ encodes the edge-to-edge similarity and is calculated by $K_e = F^{(1)} \Lambda F^{(2)}$. It is worth to note that $\Lambda \in \mathbb{R}^{2d\times 2d}$ is a learnable parameter matrix and thus the built graph matching affinity matrix K in Equation (13.4) is a learnable affinity model.

Then, with the spectral matching technique (Leordeanu and Hebert, 2005), the graph matching problem is translated into computing the leading eigenvector $\mathbf{s}^*$ which can be approximated by the power iteration algorithm as follows.

[3] https://www.thecvf.com/?page_id=413

$$\mathbf{s}_{k+1} = \frac{K\mathbf{s}_k}{\|K\mathbf{s}_k\|_2} \tag{13.5}$$

in which $\mathbf{s}$ is initialized with $\mathbf{s}_0 = \mathbf{1}$ and K is computed from Equation (13.4). It is also worth to note that the spectral graph matching solver in Equation (13.5) is differentiable but un-learnable. Because the resulting $\mathbf{s}_{k+1}$ is not a doubly-stochastic matrix, it employs a bi-stochastic normalization layer to iteratively normalize the matrix by columns and rows over and over again.

Finally, the whole graph matching model is trained in an end-to-end fashion with a displacement loss $\mathscr{L}_{disp}$ which operates the difference between predicted displacement and the ground-truth displacement.

$$\mathscr{L}_{disp} = \sum_{i=0}^{n} \sqrt{\|\mathbf{d}_i - \mathbf{d}_i^{gt}\|_2 + \varepsilon} \quad \text{and} \quad \mathbf{d}_i = \sum_{v_a \in \mathscr{V}^{(2)}} (S_{i,a} P_a^{(2)}) - P_i^{(1)} \tag{13.6}$$

where $P^{(1)}$ and $P^{(2)}$ are coordinates of nodes in both images; the vector of $\mathbf{d}_i$ measures the pixel offset; $\mathbf{d}_i^{gt}$ is the corresponding ground-truth; and ε is a small value for robust penalty.

Deep Graph Matching via Black-box Combinatorial Solver. Motivated by advances in incorporating a combinatorial optimization solver into a neural network (Pogancic et al, 2020), Rolínek et al (2020) propose an end-to-end neural network which seamlessly embeds a black-box combinatorial solver, namely BB-GM, for the graph matching problem. To be specific, given two cost vectors (*i.e.*, $\mathbf{c}^v \in \mathbb{R}^{n^2}$ and $\mathbf{c}^e \in \mathbb{R}^{|\mathscr{E}^{(1)}||\mathscr{E}^{(2)}|}$) for both node-to-node and edge-to-edge correspondences, the graph matching problem is formulated as follows.

$$\mathrm{GM}(\mathbf{c}^v, \mathbf{c}^e) = \underset{(\mathbf{s}^v, \mathbf{s}^e) \in \mathrm{Adm}(\mathscr{G}^{(1)}, \mathscr{G}^{(2)})}{\arg\min} \{\mathbf{c}^v \cdot \mathbf{s}^v + \mathbf{c}^e \cdot \mathbf{s}^e\} \tag{13.7}$$

where GM denotes the black-box combinatorial solver; $\mathbf{s}^v \in \{0,1\}^{n^2}$ is the indicator vector of matched nodes; $\mathbf{s}^e \in \{0,1\}^{|\mathscr{E}^{(1)}||\mathscr{E}^{(2)}|}$ is the indicator vector of matched edges; $\mathrm{Adm}(\mathscr{G}^{(1)}, \mathscr{G}^{(2)})$ represents a set of all possible matching results between $\mathscr{G}^{(1)}$ and $\mathscr{G}^{(2)}$.

By the formulation, the core of the graph matching problem is to construct the two cost vectors $\mathbf{c}^v$ and $\mathbf{c}^e$. Therefore, BB-GM first employs a pre-trained VGG-16 model to extract node embeddings and learns edge embeddings via SplineCNN (Fey et al, 2018). Then, based on the learned node embeddings, $\mathbf{c}^v$ is computed by a weighted inner product similarity between the pair of node embeddings between two graphs, along with a learnable neural network based on the graph-level feature vector. Similarly, $\mathbf{c}^e$ is also computed by a weighted inner product similarity between the pair of edge embeddings between two graphs, along with the same neural network.

13.2.3 Graph Neural Network based Models

More recently, GNNs have started to be studied to deal with the graph matching problem. This is because GNNs bring about new opportunities on the tasks over graph-like data and further improve the model capability taking structural information of graphs into account. Besides, GNNs can be easily incorporated with other deep learning architectures (*e.g.*, CNN, RNN, MLP, *etc.*) and thus provide an end-to-end learning framework for the graph matching problem.

Cross-graph Affinity based Graph Matching. Wang et al (2019g) claim that it is the first work that employs GNNs for deep graph matching learning (as least in computer vision). By exploiting the highly efficient learning capabilities of GNNs that can update the node embeddings with the structural affinity information between two graphs, the graph matching problem, *i.e.*, the quadratic assignment problem, is translated into a linear assignment problem that can be easily solved.

In particular, the authors present the cross-graph affinity based graph matching model with the permutation loss, namely PCA-GM. PCA-GM consists of three steps. First, to enhance learned node embeddings of individual graph with a standard message-passing network (*i.e.*, intra-graph convolution network), PCA-GM further updates node embeddings with an extra cross-graph convolution network, *i.e.*, CrossGConv which not only aggregates the information from local neighbors, but also incorporates the information from the similar nodes in the other graph. Fig. 13.2 illustrates an intuitive comparison between the intra-graph convolution network and the cross-graph convolution network formulated as follows.

$$\begin{aligned} H^{(1)(k)} &= \text{CrossGConv}\left(\hat{S}, H^{(1)(k-1)}, H^{(2)(k-1)}\right) \\ H^{(2)(k)} &= \text{CrossGConv}\left(\hat{S}^{\top}, H^{(2)(k-1)}, H^{(1)(k-1)}\right) \end{aligned} \tag{13.8}$$

where $H^{(1)(k)}$ and $H^{(2)(k)}$ are the k-layer node embeddings for the graph $\mathscr{G}^{(1)}$ and $\mathscr{G}^{(2)}$; k denotes the k-th iteration; $\hat{S}$ denotes the predicted assignment matrix which is computed from shallower node embedding layers; and the initial embeddings,

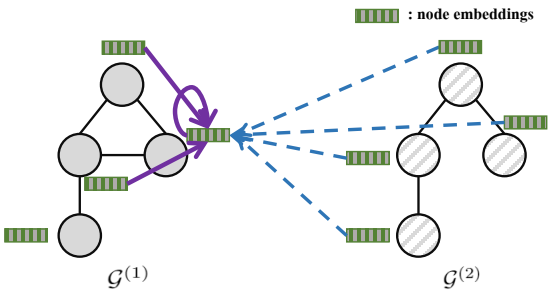

Fig. 13.2 For one node in the left graph $\mathscr{G}^{(1)}$, the intra-graph convolution network only operates on its own graph, *i.e.*, the purple solid lines in $\mathscr{G}^{(1)}$. However, the cross-graph convolution network operates on both its own graph (*i.e.*, the purple solid lines in $\mathscr{G}^{(1)}$) as well as the other graph (*i.e.*, blued dashed lines from all nodes in $\mathscr{G}^{(2)}$ to the node in $\mathscr{G}^{(1)}$).

i.e., $H^{(1)(0)}$ and $H^{(2)(0)}$, are extracted via a pre-trained VGG-16 network in line with Zanfir and Sminchisescu (2018).

Second, based on the resulting node embeddings $\widetilde{H}^{(1)}$ and $\widetilde{H}^{(2)}$ for both graphs, PCA-GM computes the node-to-node assignment matrix S by a bi-linear mapping followed by an exponential function as follows.

$$\widetilde{S} = \exp\left(\frac{\widetilde{H}^{(1)} \Theta \widetilde{H}^{(2)\top}}{\tau}\right) \tag{13.9}$$

where Θ denotes the learnable parameter matrix for the assignment matrix learning and $\tau > 0$ is a hyper-parameter. As the obtained $\widetilde{S} \in \mathbb{R}^{n \times n}$ does not satisfy the constraint of the doubly-stochastic matrix, PCA-GM uses the Sinkhorn (Adams and Zemel, 2011) operation for the relaxed linear assignment problem because it is fully differentiable and has been proven effective for the final graph matching prediction.

$$S = \text{Sinkhorn}(\widetilde{S}) \tag{13.10}$$

Finally, PCA-GM adopts the combinatorial permutation loss that computes the cross entropy loss between the final predicted permutation S and ground truth permutation S^{gt} for supervised graph matching learning.

$$\mathscr{L}_{perm} = -\sum_{v_i \in \mathscr{V}^{(1)}, v_a \in \mathscr{V}^{(2)}} S^{gt}_{i,a} \log(S_{i,a}) + (1 - S^{gt}_{i,a}) \log(1 - S_{i,a}) \tag{13.11}$$

Experiment results in (Wang et al, 2019g) demonstrated that graph matching models with the permutation loss outperform that with the displacement loss in Equation (13.6).

Graph Learning–Matching Network. Most prior studies on the graph matching problem rely on established graphs with fixed structure information, *i.e.*, the edge set with or without attributes. Differently, Jiang et al (2019a) present a graph learning-matching network, namely GLMNet, which incorporates the graph structure learning (*i.e.*, learning the graph structure information) into the general graph matching learning to build a unified end-to-end model architecture. To be specific, based on the pair of node feature matrices $X^{(l)} = \{\mathbf{x}_1^{(l)}, \cdots, \mathbf{x}_n^{(l)}\}$ $(l = \{1,2\})$, GLMNet attempts to learn a pair of optimal graph adjacency matrices $A^{(l)}$ $(l = \{1,2\})$ for better serving for the latter graph matching learning and each element is computed as follows.

$$A^{(l)}_{i,j} = \phi(\mathbf{x}_i^{(l)}, \mathbf{x}_j^{(l)}; \theta) = \frac{\exp(\sigma(\theta^\top [\mathbf{x}_i^{(l)}, \mathbf{x}_j^{(l)}]))}{\sum_{j=1}^{n} \exp(\sigma(\theta^\top [\mathbf{x}_i^{(l)}, \mathbf{x}_j^{(l)}]))}, \; l = \{1,2\} \tag{13.12}$$

where σ is the activation function, *e.g.*, ReLU; $[\cdot,\cdot]$ denotes the concatenation operation; and θ denotes the trainable parameter for the graph structure learning which is shared for both input graphs.

Following PCA-GM (Wang et al, 2019g), GLMNet also explores a series of graph convolution modules to learn informative node embeddings of both input graphs for the latter affinity matrix learning and matching prediction. Based on the obtained $A^{(l)}$ and $X^{(l)}$ ($l = \{1,2\}$), GLMNet employs the graph smoothing convolution layer (Kipf and Welling, 2017b), the cross-graph convolution layer Wang et al (2019g) and the graph sharpening convolution layer (*i.e.*, defined as the counterpart of Laplacian smoothing in (Kipf and Welling, 2017b)) to further learn and update their node embeddings *i.e.*, $\hat{X}^{(l)}$ ($l = \{1,2\}$). After that, GLMNet directly computes the node-to-node assignment matrix S by Equations (13.9) and (13.10), which is exactly the same as PCA-GM (Wang et al, 2019g) does.

In addition to the permutation cross entropy loss $\mathscr{L}_{perm}$ defined in Equation (13.11), GLMNet adds an extra constraint regularized loss $\mathscr{L}_{con}$ for better satisfying the permutation constraint, *i.e.*, $\mathscr{L} = \mathscr{L}_{perm} + \lambda \mathscr{L}_{con}$ with $\lambda > 0$, in which $\mathscr{L}_{con}$ is defined as follows.

$$\begin{aligned} \mathscr{L}_{con} &= \sum_{v_i, v_j \in \mathscr{V}^{(1)}} \sum_{v_a, v_b \in \mathscr{V}^{(2)}} U_{ij,ab} S_{i,a} S_{j,b} \\ U_{ij,ab} &= \begin{cases} 1 & \text{if } i = j, a \neq b \text{ or } i \neq j, a = b; \\ 0 & \text{otherwise.} \end{cases} \end{aligned} \tag{13.13}$$

where $U \in \mathbb{R}^{n^2 \times n^2}$ represents the conflict relationships of all matches and the optimum correspondence S means $\sum_{v_i, v_j \in \mathscr{V}^{(1)}} \sum_{v_a, v_b \in \mathscr{V}^{(2)}} U_{ij,ab} S_{i,a} S_{j,b} = 0$.

Deep Graph Matching with Consensus. In (Fey et al, 2020), Fey et al also employ GNNs to learn the graph correspondence as previous work, but additionally introduce a *neighborhood consensus* Rocco et al (2018) to further refine the learned correspondence matrix. Firstly, they use common GNN models along with the Sinkhorm operation to compute an initial correspondence matrix S^0 as follows. Ψ_{θ_1} denotes the shared GNN model for both graphs.

$$\begin{aligned} H^{(l)} &= \Psi_{\theta_1}(X^{(l)}, A^{(l)}, E^{(l)}),\ l = \{1,2\} \\ S^0 &= \text{Sinkhorn}(H^{(1)} H^{(2)\top}) \end{aligned} \tag{13.14}$$

Then, to reach a neighborhood consensus between the pair of matched nodes, they refine the initial correspondence matrix S^0 via another trainable GNN model (*i.e.*, Ψ_{θ_2}) and an MLP model (*i.e.*, ϕ_{θ_3}).

$$\begin{aligned} O^{(1)} &= \Psi_{\theta_2}(I_n, A^{(1)}, E^{(1)}) \\ O^{(2)} &= \Psi_{\theta_2}(S^{k\top} I_n, A^{(2)}, E^{(2)}) \\ S^{k+1}_{i,a} &= \text{Sinkhorn}\left(S^k_{i,a} + \phi_{\theta_3}(\mathbf{o}^{(1)}_i - \mathbf{o}^{(2)}_a)\right) \end{aligned} \tag{13.15}$$

where I_n is the identify matrix and $\mathbf{o}^{(1)}_i - \mathbf{o}^{(2)}_a$ is computed as the neighborhood consensus between the node pair $(v_i, v_a) \in \mathscr{V}^{(1)} \times \mathscr{V}^{(2)}$ between two graphs (*e.g.*, $\mathbf{o}^{(1)}_i - \mathbf{o}^{(2)}_a \neq \mathbf{0}$ means a false matching over the neighborhoods of v_i and v_j). Finally,

S^K is obtained after K iterations and the final loss function incorporates both feature matching loss and neighborhood consensus loss, *i.e.*, $\mathscr{L} = \mathscr{L}^{init} + \mathscr{L}^{refine}$.

$$\begin{aligned} \mathscr{L}^{init} &= -\sum_{v_i \in \mathscr{V}^{(1)}} \log\left(S^0_{i,\pi_{gt}(i)}\right) \\ \mathscr{L}^{refine} &= -\sum_{v_i \in \mathscr{V}^{(1)}} \log\left(S^K_{i,\pi_{gt}(i)}\right) \end{aligned} \tag{13.16}$$

where $\pi_{gt}(i)$ denotes the ground truth correspondence.

Deep Graph Matching with Hungarian Attention. Yu et al (2020) present an end-to-end deep learning model which is almost identical to Wang et al (2019g), including a graph embedding layer based on GNNs, an affinity learning layer (*i.e.*, Equations (13.9) and (13.10)), and the permutation loss (*i.e.*, Equation (13.11)). However, they improve the model with two main contributing aspects. The first aspect is adopting a novel node/edge embedding operation (*i.e.*, CIE) to replace the commonly used GCN operation that simply updates node embeddings while ignores the rich edge attributes. Since the edge information provides a crucial role in determining the graph matching result, CIE updates both node and edge embedding simultaneously by a channel-wise updating function in a multi-head fashion. Interested readers are referred to Section 3.2 in (Yu et al, 2020). Another aspect is a novel loss function. As the previously used permutation loss is prone to overfitting, the authors devise a novel loss function that introduces a Hungarian attention Z into the permutation loss as follows.

$$\begin{aligned} Z &= \text{Attention}(\text{Hungarian}(S), S^{gt}) \\ \mathscr{L}_{hung} &= -\sum_{v_i \in \mathscr{V}^{(1)}, v_a \in \mathscr{V}^{(2)}} Z_{i,a}\left(\mathbf{S}^{gt}_{i,a}\log(\mathbf{S}_{i,a}) + (1-\mathbf{S}^{gt}_{i,a})\log(1-\mathbf{S}_{i,a})\right) \end{aligned} \tag{13.17}$$

where Hungarian denotes a black-box Hungarian algorithm and the role of Z is like a mask that attempts to focus more on those mismatched node pairs and focus less on node pairs that are matched exactly.

Graph Matching with Assignment Graph. Differently, Wang et al (2020j) reformulate the graph matching problem as the problem of selecting reliable nodes in the constructed *assignment graph* (Cho et al, 2010) in which each node represents a potential node-to-node correspondence. The formal definition of assignment graph is given in Definition 13.2 and one example is illustrated in Fig. 13.3.

Definition 13.2 (Assignment Graph). Given two graphs $\mathscr{G}^{(1)} = (\mathscr{V}^{(1)}, \mathscr{E}^{(1)}, X^{(1)}, E^{(1)})$ and $\mathscr{G}^{(2)} = (\mathscr{V}^{(2)}, \mathscr{E}^{(2)}, X^{(2)}, E^{(2)})$, an *assignment graph* $\mathscr{G}^{(A)} = (\mathscr{V}^{(A)}, \mathscr{E}^{(A)}, X^{(A)}, E^{(A)})$ is constructed as follows. $\mathscr{G}^{(A)}$ takes each candidate correspondence $(v^{(1)}_i, v^{(2)}_a) \in \mathscr{V}^{(1)} \times \mathscr{V}^{(2)}$ between two graphs as a node $v_{ia} \in \mathscr{V}^{(A)}$ and link an edge between a pair of nodes $v^{(A)}_{ia}, v^{(A)}_{jb} \in \mathscr{V}^{(A)}$ (*i.e.*, $(v^{(A)}_{ia}, v^{(A)}_{jb}) \in \mathscr{E}^{(A)}$) if and only if both edges *i.e.*, $(v^{(1)}_i, v^{(1)}_j) \in \mathscr{E}^{(1)}$ and $(v^{(2)}_a, v^{(2)}_b) \in \mathscr{E}^{(2)}$, exist in its original graph. Optionally, for

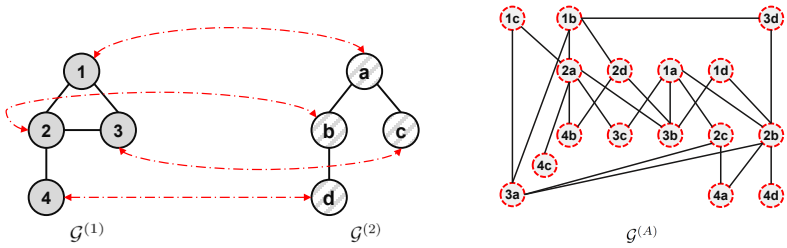

Fig. 13.3: Example illustration of building an assignment graph $\mathcal{G}^{(A)}$ from the pair of graphs $\mathcal{G}^{(1)}$ and $\mathcal{G}^{(2)}$.

node attributes $X^{(A)}$ and edge attributes $E^{(A)}$, each of them could be obtained by concatenating attributes of the pair of nodes or edges in the original graph, respectively.

With the constructed assignment graph $\mathcal{G}^{(A)}$, the reformulated problem of selecting reliable nodes in $\mathcal{G}^{(A)}$ is quite similar to binary node classification tasks Kipf and Welling (2017b) that classify nodes into positive or negative (*i.e.*, meaning matched or un-matched). To solve the problem, the authors propose a fully learnable model based on GNNs which takes the $\mathcal{G}^{(A)}$ as input, iteratively learns node embeddings over graph structural information and predicts a label for each node in $\mathcal{G}^{(A)}$ as output. Besides, the model is trained with a similar loss function to (Jiang et al, 2019a).

13.3 Graph Similarity Learning

In this section, we will first introduce the second category of the general graph matching problem – the graph similarity problem. Then, we will provide an extensive discussion and analysis of state-of-the-art graph similarity learning models based on GNNs.

13.3.1 Problem Definition

Learning a similarity metric between an arbitrary pair of graph-structured objects is one of the fundamental problems in a variety of applications, ranging from similar graph searching in databases (Yan and Han, 2002), to binary function analysis (Li et al, 2019h), unknown malware detection (Wang et al, 2019i), semantic code retrieval (Ling et al, 2021), *etc*. According to different application backgrounds, the similarity metric can be defined by different measures of structural similarity, such as graph edit distance (GED) (Riesen, 2015), maximum common subgraph (MCS) (Bunke, 1997; Bai et al, 2020c), or even more coarse binary similarity (*i.e.*,

similar or not) (Ling et al, 2021). As GED is equivalent to the problem of MCS under a fitness function (Bunke, 1997), in this section, we mainly consider the GED computation and focus more on state-of-the-art graph similarity learning models based on GNNs.

Basically, the graph similarity problem intends to compute a similarity score between a pair of graphs, which indicates how similar the pair of graphs is. In the following Definition 13.3, the general graph similarity problem is defined.

Definition 13.3 (Graph Similarity Problem). Given two input graphs $\mathcal{G}^{(1)}$ and $\mathcal{G}^{(2)}$, the purpose of graph similarity problem is to produce a similarity score s between $\mathcal{G}^{(1)}$ and $\mathcal{G}^{(2)}$. In line with the notations in Section 13.2.1, the $\mathcal{G}^{(1)} = (\mathcal{V}^{(1)}, \mathcal{E}^{(1)}, A^{(1)}, X^{(1)})$ is represented as set of n nodes $v_i \in \mathcal{V}^{(1)}$ with a feature matrix $X^{(1)} \in \mathbb{R}^{n \times d}$, edges $(v_i, v_j) \in \mathcal{E}^{(1)}$ formulating an adjacency matrix $A^{(1)}$. Similarly, $\mathcal{G}^{(2)} = (\mathcal{V}^{(2)}, \mathcal{E}^{(2)}, A^{(2)}, X^{(2)}$ is represented as set of m nodes $v_a \in \mathcal{V}^{(2)}$ with a feature matrix $X^{(2)} \in \mathbb{R}^{m \times d}$, edges $(v_a, v_b) \in \mathcal{E}^{(2)}$ formulating an adjacency matrix $A^{(2)}$.

For the similarity score s, if $s \in \mathbb{R}$, the graph similarity problem can be considered as the *graph-graph regression tasks*. On the other hand, if $s \in \{-1, 1\}$, the problem can be considered as the *graph-graph classification tasks*.

Particularly, the computation of GED (Riesen, 2015; Bai et al, 2019b) (sometimes normalized in $[0, 1]$) is a typical case of graph-graph regression tasks. To be specific, GED is formulated as the cost of the shortest sequence of edit operations over nodes or edges which have to undertake to transform one graph into another graph, in which an edit operation can be an insertion or a deletion of a node or an edge. In Fig. 13.4, We give an illustration of GED computation.

Similar to the classic graph matching problem, the computation of GED is also a well-studied NP-hard problem. Although there is a rich body of work (Hart et al, 1968; Zeng et al, 2009; Riesen et al, 2007) that attempts to find sub-optimal solutions in polynomial time via a variety of heuristics (Riesen et al, 2007; Riesen, 2015), these heuristic methods still suffer from the poor scalability (*e.g.*, large search space or excessive memory) and heavy reliance on expert knowledge (*e.g.*, various heuristics based on different application cases). Currently, learning-based models which incorporate GNNs into an end-to-end learning framework for graph similarity learning are gradually becoming more and more available, demonstrating the

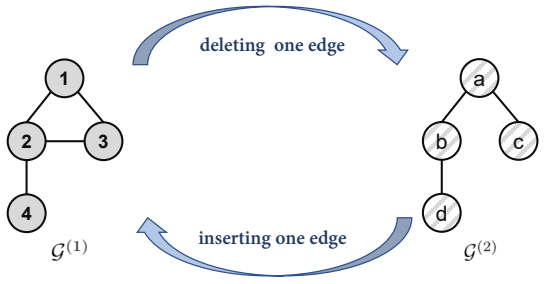

Fig. 13.4 Illustration of computing the GED score between $\mathcal{G}^{(1)}$ and $\mathcal{G}^{(2)}$. Since $\mathcal{G}^{(1)}$ can be transform into in $\mathcal{G}^{(2)}$ by deleting the edge (v_2, v_3) or $\mathcal{G}^{(2)}$ can be transformed into in $\mathcal{G}^{(1)}$ by inserting the edge (v_b, v_c), the GED between two graphs is 1.

superiority by traditional heuristic methods in both effectiveness and efficiency. In two following subsections, we will discuss state-of-the-art GNN-based graph similarity models for graph-graph regression tasks and graph-graph classification tasks, respectively.

13.3.2 Graph-Graph Regression Tasks

As mentioned above, the graph-graph regression task refers to computing a similarity score between a pair of graphs and we focus on the graph similarity learning on GED in this subsection.

Graph Similarity Learning with Convolutional Set Matching. Aiming at accelerating the graph similarity computation while preserving a good performance, Bai et al (2018) first turn the computation of GED into a learning problem rather than approximation methods with combinatorial search, and then propose an end-to-end framework, namely GSimCNN, for the graph similarity learning. For GSimCNN in (Bai et al, 2018) (or GraphSim in (Bai et al, 2020b)[4]), it is probably the first work that applies both GNNs and CNNs for the task of GED computation and consists of three steps in general. First, GSimCNN employs multiple layers of standard GCNs to generate the node embedding vector for each node in the pair of graphs. Second, in each layer of GCNs, GSimCNN uses the BFS node-ordering scheme (You et al, 2018b) to re-order the node embeddings and compute the inner product between the re-ordered node embeddings in two graphs to generate a node-to-node similarity matrix. Finally, after padding or resizing resulting node-to-node similarity matrices into square matrices, the authors transform the task of graph similarity computation into an image processing problem and explore standard CNNs and MLPs for the final graph similarity prediction. GSimCNN is trained with a mean squared error loss function based on predicted similarity scores and the corresponding ground-truth scores.

Graph Similarity Learning with Graph-Level Interaction. Soon after, Bai et al present another GNN-based model, called SimGNN, for graph similarity learning. In SimGNN, it takes not only node-level interactions but also graph-level interactions as considerations for jointly learning the graph similarity score. For the node-level similarity between two graphs, it first adopts a similar approach like GSimCNN to generate the node-to-node similarity matrix, and then extract a histogram feature vector from the matrix as the node-level comparison information. For the graph-level similarity between two graphs, SimGNN first employs a simple graph pooling model via an attention mechanism to generate one graph-level embedding vector for each graph ($\mathbf{h}_{\mathcal{G}^{(1)}}$ and $\mathbf{h}_{\mathcal{G}^{(2)}}$) and then adopts a trainable neural tensor network (NTN) (Socher et al, 2013) to model the relationship between the two graph-

[4] It seems that the model architecture of GSimCNN in (Bai et al, 2018) is the same as that of GraphSim in (Bai et al, 2020b), which evaluates the model with additional datasets and similarity metrics (*i.e.*, both GED and MCS).

level embedding vectors as follows.

$$\mathrm{NTN}(\mathbf{h}_{\mathscr{G}^{(1)}}, \mathbf{h}_{\mathscr{G}^{(2)}}) = \sigma\Big(\mathbf{h}_{\mathscr{G}^{(1)}}^{\top} W^{[1:K]} \mathbf{h}_{\mathscr{G}^{(2)}} + V\big[\begin{smallmatrix}\mathbf{h}_{\mathscr{G}^{(1)}} \\ \mathbf{h}_{\mathscr{G}^{(2)}}\end{smallmatrix}\big] + \mathbf{b}\Big) \tag{13.18}$$

where σ is the activation function and $[\cdot\,;\cdot]$ denotes the concatenation operation. In addition, $W^{[1:K]}$, V and $\mathbf{b}$ are parameters in NTN to be learned and K is a hyper-parameter which determines the length of the graph-level similarity vector calculated by NTN. Finally, to compute the similarity score between two graphs, SimGNN concatenates two similarity vectors from the node level and the graph level along with a small MLP network for prediction.

Graph Similarity Learning based on Hierarchical Clustering. In (Xiu et al, 2020), Xiu et al argue that if two graphs are similar, their corresponding compact graphs should be similar with each other and conversely if two graphs are dissimilar, their corresponding compact graphs should also be dissimilar. They believe that, for the input pair of graphs, different views in regard to different pairs of compact graphs can provide different scales of similarity information between two input graphs and thus benefit the graph similarity computation. To this end, a hierarchical graph matching network (HGMN) (Xiu et al, 2020) is presented to learn the graph similarity from a multi-scale view. Concretely, HGMN first employs multiple stages of hierarchical graph clustering to successively generate more compact graphs with initial node embeddings to provide a multi-scale view of differences between two graphs for subsequent model learning. Then, with the pairs of compact graphs in different stages, HGMN computes the final graph similarity score by adopting a GraphSim-like model (Bai et al, 2020b), including node embeddings update via GCNs, similarity matrices generation and prediction via CNNs. However, in order to ensure permutation invariance of generated similarity matrices, HGMN devises a different node-ordering scheme based on earth mover distance(EMD) (Rubner et al, 1998) rather than BFS node-order method in (Bai et al, 2020b). According to the EMD distance, HGMN first aligns nodes for both input graphs in each stage and then produces the corresponding similarity matrix in the aligned order.

Graph Similarity Learning with Node-Graph Interaction. To learn richer interaction features between a pair of input graphs for computing the graph similarity in an end-to-end fashion, Ling et al propose a multi-level graph matching network (MGMN) (Ling et al, 2020) which consists of a siamese graph neural network (SGNN) and a novel node-graph matching network (NGMN). To learn graph-level interactions between two graphs, SGNN first utilizes a multi-layer of GCNs with the siamese network to generate node embeddings $H^{(l)} = \{\mathbf{h}_i^{(l)}\}_{i=1}^{\{n,m\}} \in \mathbb{R}^{\{n,m\}\times d}$ for all nodes in graph $G^{(l)}$, $l = \{1,2\}$ and then aggregates a corresponding graph-level embedding vector for each graph. On the other hand, to learn cross-level interaction features between two graphs, NGMN further employs a node-graph matching layer to update node embeddings with learned cross-level interactions between node embeddings of a graph and a corresponding graph-level embedding of the other whole

graph. Taking a node $v_i \in \mathscr{V}^{(1)}$ in $\mathscr{G}^{(1)}$ as an example, NGMN first computes an attentive graph-level embedding vector $\mathbf{h}_{G^{(2)}}^{i,att}$ for $\mathscr{G}^{(2)}$ by weighted averaging all node embeddings in $\mathscr{G}^{(2)}$ based on the corresponding cross-graph attention coefficient towards v_i as follows.

$$\mathbf{h}_{G^{(2)}}^{i,att} = \sum_{v_j \in \mathscr{V}^{(2)}} \alpha_{i,j} \mathbf{h}_j^{(2)}, \text{ where } \alpha_{i,j} = \text{cosine}(\mathbf{h}_i^{(1)}, \mathbf{h}_j^{(2)}) \; \forall v_j \in \mathscr{V}^{(2)} \tag{13.19}$$

where *att* in the superscript of $\mathbf{h}_{G^{(2)}}^{i,att}$ means it is an attentive graph-level embedding vector of $G^{(2)}$ in terms of the node v_i in $\mathscr{G}^{(1)}$.

Then, to update the node embedding of v_i with cross-graph interactions, NGMN learns similarity feature vector between the node embedding (*i.e.*, $\mathbf{h}_i^{(1)}$) and the attentive graph-level embedding vector (*i.e.*, $\mathbf{h}_{G^{(2)}}^{i,att}$) via a multi-perspective matching function. After performing the above node-graph matching layer over all nodes for both graphs, NGMN aggregates a corresponding graph-level embedding vector for each graph. The full model MGMN concatenates the two aggregated graph-level embeddings from both SGNN and NGMN for each graph and feed those concatenated embeddings into a final small prediction network for the graph similarity computation.

Graph Similarity Learning based on GRAPH-BERT. As previous studies on the graph similarity learning are mostly trained in a supervised manner and cannot guarantee the basic properties (*e.g.*, triangle inequality) of the graph similarity metric like GED, Zhang introduces a novel training framework of GB-DISTANCE (Zhang, 2020) based on GRAPH-BERT (Zhang et al, 2020a). First, GB-DISTANCE adapts the pre-trained GRAPH-BERT model to update node embeddings and further aggregate a graph-level representation embedding of vector $\mathbf{h}_{\mathscr{G}^{(i)}}$ for the graph $\mathscr{G}^{(i)}$. Then, GB-DISTANCE computes the graph similarity $d_{i,j}$ between the pair of graphs $(\mathscr{G}^{(i)}, \mathscr{G}^{(j)})$ with several fully connected layers as follows.

$$d(\mathscr{G}^{(i)}, \mathscr{G}^{(j)}) = 1 - \exp\Big(- \text{FC}\big((\mathbf{h}_{\mathscr{G}^{(i)}} - \mathbf{h}_{\mathscr{G}^{(j)}}) * *2\big)\Big) \tag{13.20}$$

where FC denotes the employed fully connected layers and $(\cdot) * *2$ denotes the element-wise square of the input vector. In (Zhang, 2020), GB-DISTANCE considers a scenario that inputs a set of m graphs (*i.e.*, $\{\mathscr{G}^{(i)}\}_{i=1}^{m}$) and outputs the similarity between any pair of graphs, *i.e.*, a similarity matrix $D = \{D_{i,j}\}_{i,j=1}^{i,j=m} = \big\{d(\mathscr{G}^{(i)}, \mathscr{G}^{(j)})\big\}_{i,j=1}^{i,j=m} \in \mathbb{R}^{m \times m}$, and formulates the graph similarity problem in a supervised or semi-supervised settings as follows.

$$\begin{aligned} &\min \|M \odot (D - \hat{D})\|_p \text{ with } M_{i,j} = \begin{cases} 1 & \text{if } D_{i,j} \text{ is labeled} \\ \alpha & \text{if } D_{i,j} \text{ is unlabeled } \wedge i \neq j \\ \beta & \text{if } i = j \end{cases} \\ &\text{s.t.} D_{i,j} \leq D_{i,k} + D_{k,j}, \forall i, j, k \in \{1, \cdots, m\} \end{aligned} \tag{13.21}$$

where $\|\cdot\|_p$ denotes the L_p norm; $\hat{D}$ denotes the ground-truth similarity matrix; M is a mask matrix for the semi-supervised learning with two hyper-parameters α and β; the constraint of $D_{i,j} \leq D_{i,k} + D_{k,j}, \forall i,j,k \in \{1,\cdots,m\}$ tries to ensure the triangle inequality of graph similarity metrics. To optimize the model with such constraints, GB-DISTANCE devises a two-phase training algorithm with the constrained metric refining methods.

Graph Similarity Computation based on A*. It is obviously observed that all these aforementioned approaches directly compute the GED similarity score between two graphs, however, failing to produce the edit path, which can explicitly express the sequence of edit operations for transforming one graph into the other graph. To output the edit path like the traditional A* (Hart et al, 1968; Riesen et al, 2007) algorithm, Wang et al propose a graph similarity learning model GENN-A* (Wang et al, 2020f) which incorporates the existing solution of A* with a learnable GENN model based on GNNs. A* (Hart et al, 1968; Riesen et al, 2007) is a tree-searching algorithm which explores the space of all possible node/edge mappings between two graphs as an ordered search tree and further expands successors of a node p in the search tree by the minimum induced edit cost $g(p)+h(p)$, in which $g(p)$ is the cost of current partial edit path induced so far and $h(p)$ is the estimated cost of edit path between the remaining un-matched sub-graphs. Because of the poor scalability of A*, GENN-A* thus replaces the heuristics with a learning-based model (*i.e.*, GENN) to predict $h(p)$. GENN is almost the same as SimGNN (Bai et al, 2019b) with the removal of the histogram module and is used to predict a normalized GED score $s(p) \in (0,1)$ between the remaining un-matched sub-graphs. After that, the $h(p)$ is obtained as follows where $\hat{n}$ and $\hat{m}$ denote the numbers of nodes of the un-matched sub-graphs.

$$h(p) = -0.5(\hat{n}+\hat{m})\log(s(p)) \tag{13.22}$$

13.3.3 Graph-Graph Classification Tasks

In addition to the computation of GED, learning a binary label $s \in \{-1,1\}$ (*i.e.*, similar or not) between a pair of graphs can be view as a task of the graph-graph classification learning[5] and has been widely studied in many real-world applications, including binary code analysis, source code analysis, malware detection, *etc*.

Graph Similarity Learning via Cross-graph Matching. In the scenario of detecting whether two binary functions are similar or not, Li et al present a message-passing based graph matching network (GMN) (Li et al, 2019h) to learn a similarity label between the two control-flow graphs (CFGs) which represent two input binary functions. In particular, GMN employs a similar cross-graph matching network

[5] The termed graph-graph classification learning is totally different from the general graph classification task (Ying et al, 2018c; Ma et al, 2019d) that only predicts a label for one input graph rather than a pair of input graphs.

based on standard message-passing GNNs to iteratively generate more discriminative node embeddings (*e.g.*, $H^{(l)} = \{\mathbf{h}_i^{(l)}\}_{v_i \in \mathcal{V}^{(l)}},\ l = \{1,2\}$) for two input graphs. Intuitively, it updates the node embeddings of one input graphs by incorporating the attentive association information of another through a soft attention, which is similar to the cross-graph convolution network introduced in Equation (13.8) and Fig. 13.2. Subsequently, in order to calculate the similarity score, GMN adopts an aggregation operation (Li et al, 2016b) as follows to output a graph-level embedding vector (*i.e.*, $\mathbf{h}_{G^{(l)}},\ l = \{1,2\}$) for each graph and applies an existing similarity function for the final similarity prediction, *i.e.*, $s(\mathbf{h}_{G^{(1)}}, \mathbf{h}_{G^{(2)}}) = f_s(\mathbf{h}_{G^{(1)}}, \mathbf{h}_{G^{(2)}})$, where f_s can be an arbitrary existing similarity function such as Euclidean, cosine or Hamming similarity function.

$$\mathbf{h}_{G^{(l)}} = \text{MLP}_{\theta 1}\Big(\sum_{v_i \in \mathcal{V}^{(l)}} \sigma\big(\text{MLP}_{\theta 2}(\mathbf{h}_i^{(l)})\big) \odot \text{MLP}_{\theta 3}(\mathbf{h}_i^{(l)})\Big),\ l = \{1,2\} \tag{13.23}$$

where σ denotes the activation function; $\odot$ denotes the element-wise multiplication operation; $\text{MLP}_{\theta 1}$, $\text{MLP}_{\theta 2}$, $\text{MLP}_{\theta 3}$ are MLP networks to be trained. Based on different supervisions of training samples (*e.g.*, the ground-truth binary label between two graphs or relative similarity among three graphs), GMN adopts two margin-based loss functions, *i.e.*, the pair loss function and the triplet loss function. As for different similarity functions f_s employed, the formulation of the corresponding loss function is quite different. Thus, we refer interested readers for the loss functions to (Li et al, 2019h).

Graph Similarity Learning on Heterogeneous Graphs. Motivated by ever-growing malware threats, a heterogeneous graph matching network (MatchGNet) framework (Wang et al, 2019i) is proposed for unknown malware detection. To better represent programs (*e.g.*, benign or malicious) in enterprise systems and capture interaction relationships between system entities (*e.g.*, files, processes, sockets, *etc.*), a heterogeneous invariant graph is constructed for each program. Therefore, the malware detection problem is equivalent to detecting whether two representation graphs (*i.e.*, the graph of the input program and the graph of the existing benign program) are similar or not. Due to the heterogeneity of the invariant graph, MatchGNet employs a hierarchical attention graph neural encoder (HAGNE)-based GNN to learn a graph-level embedding vector for each program. Particularly, HAGNE first identifies path-relevant sets of neighbors via meta-paths (Sun et al, 2011) and then updates node embeddings by aggregating the entities under each path-relevant neighbor set. The graph-level embedding over all the meta-paths is computed by a weighted summarization of all embeddings of meta-paths. Finally, MatchGNet directly calculates the cosine similarity between the two graph-level embedding vector as the final predicted label for malware detection.

13.4 Summary

In this chapter, we have introduced the general graph matching learning, whereby objective functions are formulated for establishing an optimal node-to-node correspondence matrix between two graphs for the classic graph matching problem and computing a similarity metric between two graphs for the graph similarity problem, respectively. In particular, we have thoroughly analyzed and discussed state-of-the-art GNN-based graph matching models and graph similarity models. In the future, for better graph matching learning, some directions we believe are requiring more efforts:

- **Fined-grained cross-graph features.** For the graph matching problem which inputs the pair of graphs, interaction features between two graphs are fundamental and key features in both the graph matching learning and the graph similarity learning. Although several existing methods (Li et al, 2019h; Ling et al, 2020) have been devoted to learning interacted features between two graphs for better representation learning, these models have caused non-negligible extra computational overhead. Better fined-grained cross-graph feature learning with efficient algorithms could make a new state of the art.
- **Semi-supervised learning and un-supervised learning.** Because of the complexity of graphs in the real-world application scenarios, it is common to train the model in a semi-supervised setting or even in an un-supervised setting. Making full use of relationships between existing graphs and, if possible, the other data that is not directly relevant to the graph matching problem could further promote the development of graph matching/similarity learning in more practical applications.
- **Vulnerability and robustness.** Although adversarial attacks have been extensively studied for image classification tasks (Goodfellow et al, 2015; Ling et al, 2019) and node/graph classification tasks (Zügner et al, 2018; Dai et al, 2018a), there is currently only one preliminary work (Zhang et al, 2020f) that studies adversarial attacks on the graph matching problem. Therefore, studying the vulnerability of the state-of-the-art graph matching/similarity models and further building more robust models is a highly challenging problem.

Editor's Notes: Graph Matching Networks is an emerging research topic recently and have drawn significant number of interests in both research community and industrial community due to its broad range of application domains such as computer vision (Chapter 20), Natural Language Processing (Chapter 21), Program Analysis (Chapter 22), Anomaly Detection (Chapter 26). Graph Matching Networks is built on graph node representation learning (Chapter 4) but focuses more on the interaction of two graphs from low-level nodes to high-level graphs. It has tight connection with link prediction (Chapter 10) and self-supervised learning (Chapter 18), where graph matching could be formulated as one of the sub-tasks for these graph learning tasks. Obviously, adversarial robustness (Chapter 8) could have direct impact of graph matching networks, which has recently been extensively studied as well.