Chapter 4
Subgraph Augmentation with Application to Graph Mining

Jiajun Zhou, Jie Shen, Yalu Shan, Qi Xuan, and Guanrong Chen

Abstract Graph classification, which aims to identify the category labels of graphs, plays a significant role in drug classification, toxicity detection, protein analysis etc. However, the limitation of the general scale of benchmark datasets makes easily causes graph classification models to fall into overfitting and undergeneralization. In this chapter, the *M-Evolve* framework is introduced for graph classification (Zhou et al., Data augmentation for graph classification. In: Proceedings of the 29th ACM International Conference on Information & Knowledge Management, pp. 2341–2344, 2020; Zhou et al., M-evolve: structural-mapping-based data augmentation for graph classification. In: IEEE Transactions on Network Science and Engineering, pp. 1–1, 2020), a novel technique for expanding graph structured data spaces and optimizing graph classifiers. Typical graph tasks such as node classification and link prediction are unified to generate graph classification patterns, demonstrating some applications to multiple tasks in graph mining. One of the main contributions of this chapter is to apply the technique of subgraph augmentation for various tasks. The *M-Evolve* is general and flexible, which can be easily combined with existing graph classification models. Extensive experiments are conducted on real datasets to illustrate the effectiveness of our framework.

J. Zhou · J. Shen · Y. Shan · Q. Xuan (✉)
Institute of Cyberspace Security, Zhejiang University of Technology, Hangzhou, China

College of Information Engineering, Zhejiang University of Technology, Hangzhou, China
e-mail: xuanqi@zjut.edu.cn

G. Chen
Department of Electrical Engineering, City University of Hong Kong, Hong Kong SAR, China

Q. Xuan et al. (eds.), *Graph Data Mining*, Big Data Management,
https://doi.org/10.1007/978-981-16-2609-8_4

4.1 Introduction

Graph classification, or network classification, has recently attracted considerable attention from different scientific and engineering communities such as bioinformatics [1] and chemoinformatics [2]. In bioinformatics, proteins or enzymes can be represented as labeled graphs, in which nodes are atoms and edges represent chemical bonds that connect atoms. The task of graph classification is to classify these molecular graphs according to their chemical properties like carcinogenicity, mutagenicity and toxicity.

However, in bioinformatics and chemoinformatics, the scale of the known benchmark graph datasets is generally in the range of tens to thousands, which is far from the scale of real-world social network datasets like COLLAB and IMDB [3]. Despite the advances of various graph classification methods, from graph kernels, graph embedding to graph neural networks, the limitation of data scale makes them easily fall into overfitting and undergeneralization. Overfitting refers to a modeling error that occurs when a model learns a function with high variance to perfectly fit the limited data. A natural idea to address overfitting is data augmentation, which is widely applied in computer vision (CV) and natural language processing (NLP). Data augmentation encompasses a number of techniques that enhance both the scale and the quality of training data such that the models of higher performance can be learnt satisfactorily. In CV, image augmentation methods include geometric transformation, color depth adjustment, neural style transfer, adversarial training, etc. However, different from image data, which have a clear grid structure, graphs have irregular topological structures, making it hard to generalize some basic augmentation operations to graphs.

To solve the above problem, we take an effective approach to study data augmentation on graphs and develop a subgraph augmentation method, called *motif-similarity mapping*. The idea is to generate more virtual data for small datasets via heuristic modification of graph structures. Since the generated graphs are artificial and treated as weakly labeled data, their validity remains to be verified. Therefore, we introduce a concept of label reliability, which reflects the matching degree between examples and their labels, to filter fine augmented examples from generated data. Furthermore, we introduce a model evolution framework, named *M-Evolve* [4, 5], which combines subgraph augmentation, data filtration and model retraining to optimize classifiers. We demonstrate that *M-Evolve* achieves a significant improvement of performance on graph classification.

We further explore the transferability of *M-Evolve* to other graph tasks such as node classification and link prediction. Motivated by recent works of graph neural networks (GNNs) [6, 7], we show that GNNs achieve node classification/link prediction by extracting a local subgraph around each target node/link, and by learning a function mapping the subgraph patterns to node labels/link existence. Hence, we unify both node classification and link prediction into graph classification, i.e., we achieve node classification or link prediction via local subgraph classification. We demonstrate that *M-Evolve* can be transferred to these graph tasks and achieve significant improvement of performance.

The rest of this chapter is organized as follows. In Sect. 4.2, we briefly review some related work on graph classification and data augmentation in graph mining. In Sect. 4.3, we introduce the subgraph augmentation technique and the model evolution framework. In Sect. 4.4, we discuss the application of the new framework to graph classification, node classification and link prediction, which involve a comparison between the results obtained from the original datasets and those obtained from subgraph augmentation. We conclude the chapter in Sect. 4.5.

4.2 Related Work

4.2.1 Graph Classification

4.2.1.1 Graph Kernel Methods

Graph kernels perform graph comparison by recursively decomposing pairwise graphs from the dataset into atomic substructures and then using a similarity function among these substructures. Intuitively, graph kernels can be understood as functions measuring the similarity of pairwise graphs. Generally, the kernels can be designed by considering various structural properties like the similarity of local neighborhood structures (WL kernel [8], propagation kernel [9]), the occurrence of certain graphlets or subgraphs (graphlet kernel [10]), the number of walks in common (random walk kernel [11–14]), and the attributes and lengths of the shortest paths (SP kernel [15]).

4.2.1.2 Embedding Methods

Graph embedding methods [16–19] capture the graph topology and derive a fixed number of features, ultimately achieving vector representations for the whole graph. For prediction on the graph level, this approach is compatible with any standard machine learning classifier such as support vector machine (SVM), k-nearest neighbors and random forest. Widely used embedding methods include graph2vec [20], structure2vec [21], subgraph2vec [22], etc.

4.2.1.3 Deep Learning Methods

Recently, increasing attention is drawn to the application of deep learning to graph mining and a wide variety of graph neural network (GNN) frameworks have been proposed for graph classification, including methods inspired by convolutional neural network (CNN), recurrent neural network (RNN), etc. One typical approach is to obtain a representation of the entire graph by aggregating the node embeddings that are the output of GNNs [2, 23]. Some sequential methods [24–26] handle these

graphs with varying sizes by transforming them into sequences of fixed-length vectors and then feeding them to RNN. In addition, some hierarchical clustering methods [27–29] are used to learn hierarchical graph representations by combining GNNs with clustering algorithms. Notably, some recent works design universal graph pooling modules, which can learn the hierarchical representations of graphs and are compatible with various GNN architectures, e.g., DiffPool [30] learns a differentiable soft cluster assignment for vertices and then maps them to a coarsened graph layer by layer, and EigenPool [31] compresses the node features and local structures into coarsened signals via graph Fourier transform.

4.2.2 Data Augmentation in Graph Learning

Data augmentation, which aims to improve generalization of machine learning models, has been broadly studied in different fields such as computer vision (CV) and natural language processing (NLP). However, data augmentation for graph data remains under-explored. This is mainly due to the complex non-Euclidean structure of graphs, which limits possible manipulation operations. Recently, Zhao et al. [32] proposed a graph data augmentation framework named GAUG, which improves performance in GNN-based node classification via link prediction, based on these techniques that link prediction can effectively encode class-homophilic structure to promote intra-class edges and demote inter-class edges in given graph structure. Wang et al. [33] presented a node-parallel augmentation (NodeAug) strategy, which conducts data augmentation by changing both the node attributes and the graph structure, to improve the performance of GCN on semi-supervised node classification. Spinelli et al. [34] proposed a data augmentation strategy based on the application of GINN [35], in which a similarity graph is constructed using both labeled and unlabeled data and then a customized graph autoencoder is applied to augment new data. These studies focus on semi-supervised node classification task. While the work to be presented in this chapter is similar in motivation, it mainly tackles augmentation in the graph classification task.

4.3 The Model Evolution Framework for Graph Classification

4.3.1 Problem Formulation

Let $G = (V, E)$ be an undirected and unweighted graph, which consists of a node set $V = \{v_i \mid i = 1, \ldots, n\}$ and an edge set $E = \{e_i \mid i = 1, \ldots, m\}$. The topological structure of graph G is represented by an $n \times n$ adjacency matrix A with $A_{ij} = 1$ if $(i, j) \in E$ and $A_{ij} = 0$ otherwise. Dataset that contains a series of graphs

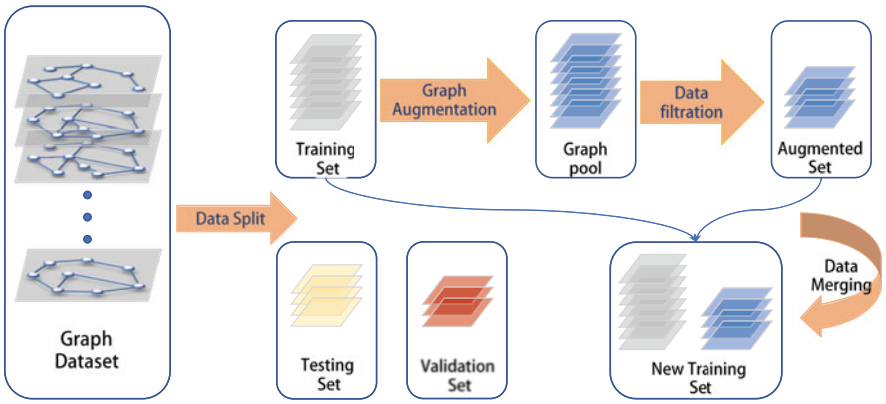

Fig. 4.1 An illustration of data augmentation application in graph (Subgraph Augmentation)

is denoted as $D = \{(G_i, y_i) \mid i = 1, \ldots, t\}$, where y_i is the label of graph G_i. For D, an upfront split will be applied to yield disjoint training, validation and testing set, denoted as D_{train}, D_{val} and D_{test}, respectively. The original classifier C will be pre-trained on D_{train} and D_{val}.

We further explore data augmentation technique for graph classification from a heuristic approach and consider optimizing graph classifier. Figure 4.1 demonstrates the application of data augmentation to graph structured data, which consists of two phases: subgraph augmentation and data filtration. Specifically, we aim to update a classifier with augmented data, which are first generated via subgraph augmentation and then filtered in terms of their label reliability. During subgraph augmentation, our purpose is to map the graph $G \in D_{train}$ to a new graph G' with the formal format: $f : (G, y) \mapsto (G', y)$. We treat the generated graphs as weakly labeled data and classify them into two groups via a label reliability threshold θ learnt from D_{val}. Then we merge the augmented set D'_{train}, filtered from generated graph pool D_{pool}, with D_{train} to produce the training set:

$$D^{new}_{train} = D_{train} + D'_{train}\,, \quad D'_{train} \subset D_{pool}\,. \tag{4.1}$$

Finally, we finetune or retrain the classifier with D^{new}_{train}, and evaluate it on the testing set D_{test}.

4.3.2 Subgraph Augmentation

Subgraph augmentation aims to expand training data via artificially creating more reasonable virtual data from a limited set of graphs. In this chapter, we consider augmentation as a topological mapping, which is conducted via heuristic modification of the graph structure. In order to ensure the approximate reasonability

of the generated virtual data, our subgraph augmentation method will follow two principles: (1) edge modification, where G' is a partially modified graph with some of the edges added/removed from G; (2) structural property preservation, where augmentation operation keeps the graph connectivity and the number of edges constant. During edge modification, those edges removed from the graph are sampled from the candidate edge set E^c_{del}, while the edges added to the graph are sampled from the candidate pairwise nodes set E^c_{add}. The construction of candidate sets varies for different methods, as further discussed below.

4.3.2.1 Random Mapping

Here, consider *random mapping* as a simple baseline. The candidate sets are constructed as follows:

$$E^c_{del} = E\ , \quad E^c_{add} = \{(v_i, v_j) \mid A_{ij} = 0;\ i \neq j\}. \tag{4.2}$$

Notably, in a random scenario, E^c_{del} is the edge set of graph, and E^c_{add} is the set of virtual edges, which consist of unlinked pairwise nodes. Then, one can get the set of edges added/removed from G via sampling from the candidate sets randomly:

$$\begin{aligned} E_{del} &= \{e_i \mid i = 1, \ldots, \lceil m \cdot \beta \rceil\} \subset E^c_{del}\ , \\ E_{add} &= \{e_i \mid i = 1, \ldots, \lceil m \cdot \beta \rceil\} \subset E^c_{add}\ , \end{aligned} \tag{4.3}$$

where β is the budget of edge modification and $\lceil x \rceil = \mathbf{ceil}(x)$. Finally, based on the *random mapping*, the connectivity structure of the original graph is modified to generate a new graph:

$$G' = (V, (E \cup E_{add}) \backslash E_{del})\ . \tag{4.4}$$

4.3.2.2 Motif-Similarity Mapping

Graph motifs are subgraphs that repeat themselves in a specific graph or even among various graphs. Each of these sub-graphs, defined by a particular pattern of interactions between nodes, may describe a framework, in which particular functions are achieved efficiently. Here, for simplicity, only consider open-triad motifs with chain structures. As shown on the left of Fig. 4.2, open-triad $\wedge^a_{ij}$ is equivalent to length-2 paths emanating from the head node v_i that induce a triangle.

The *motif-similarity mapping* aims to finetune these motifs to be approximately equivalent ones via edge swapping. During edge swapping, edge addition takes effect between the head and the tail nodes of the motif, while edge deletion removes an edge in the motif via weighted random sampling. For all open-triad motifs $\wedge_{ij}$, which has head node v_i and tail node v_j, the candidate set of pairwise nodes is

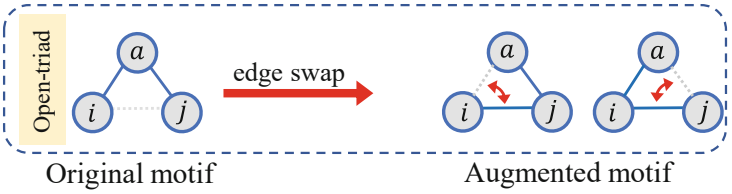

Fig. 4.2 Open-triad motif and heuristic edge swapping

denoted as

$$E^c_{add} = \{(v_i, v_j) \mid A_{ij} = 0, A^2_{ij} \neq 0; i \neq j\} . \tag{4.5}$$

Then, we get E_{add}, the set of edges added to G, via weighted random sampling from E^c_{add}. For each $\wedge_{ij}$ involving pairwise nodes (v_i, v_j) in E_{add}, we remove one edge from it via weighted random sampling, and all of these removed edges constitute E_{del}.

Note that we assign all entries in E^c_{add} and $\wedge_{ij}$ with relative sampling weights that are associated with the node similarity scores. Specifically, before sampling, we compute the similarity scores over all entries in E^c_{add} using the *Resource Allocation (RA)* index, which has superiority among several local similarity indices [36]. For each entry (v_i, v_j) in E^c_{add}, the *RA* score s_{ij} and addition weight w^{add}_{ij} can be computed as follows:

$$\begin{aligned} s_{ij} &= \sum\nolimits_{z \in \Gamma(i) \cap \Gamma(j)} \frac{1}{d_z}, \\ S &= \{s_{ij} \mid \forall (v_i, v_j) \in E^c_{add}\}, \\ w^{add}_{ij} &= \frac{s_{ij}}{\sum_{s \in S} s}, \\ W_{add} &= \{w^{add}_{ij} \mid \forall (v_i, v_j) \in E^c_{add}\}, \end{aligned} \tag{4.6}$$

where $\Gamma(i)$ denotes the one-hop neighbors of v_i and d_z denotes the degree of node z. Weighted random sampling means that the probability for an entry in E^c_{add} to be selected is proportional to its addition weight w^{add}_{ij}. Similarly, during edge deletion, the probability of edge sampled from $\wedge_{ij}$ is proportional to the deletion weight w^{del}_{ij}, as follows:

$$\begin{aligned} w^{del}_{ij} &= 1 - \frac{s_{ij}}{\sum_{s \in S} s}, \\ W_{del} &= \{w^{del}_{ij} \mid \forall (v_i, v_j) \in \wedge_{ij}\}, \end{aligned} \tag{4.7}$$

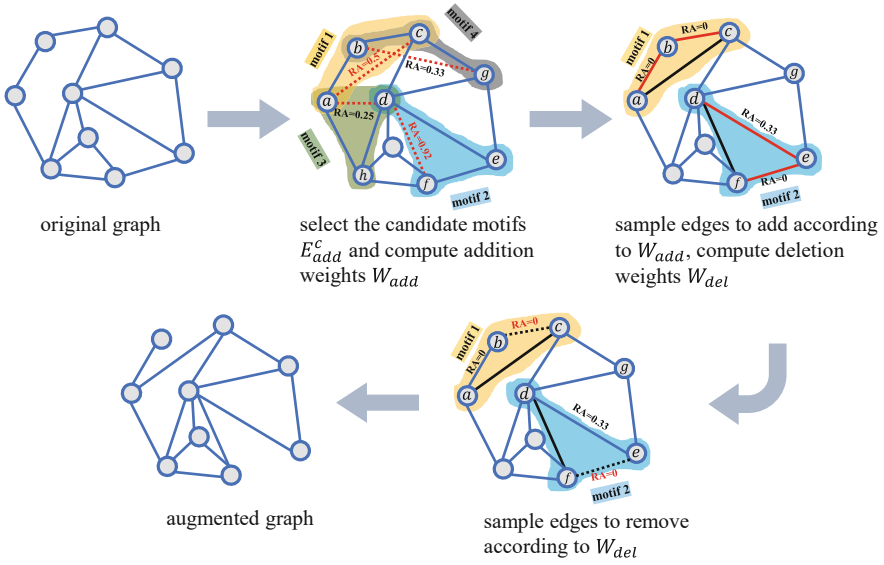

Fig. 4.3 An example of subgraph augmentation via motif-similarity mapping; red lines is the candidates and black lines is the modified edges

which means that edges with smaller *RA* scores have more chance to be removed. It is worth noting that many other similarity indices such as *Common Neighbors* (CN) and *Katz* [36] can also be applied into this scheme. Finally, the augmented graph can be obtained via Eq. (4.4). The example of subgraph augmentation using motif-similarity mapping is shown in Fig. 4.3.

Algorithm 1: Motif-similarity mapping

Input: Target network G, length of motif l, proportion of modification β.
Output: Augmented graph G'

Get E^c_{add} via Eq. (4.5) ;
Compute the addition weights W_{add} via Eq. (4.6) ;
$E_{add} \leftarrow \mathsf{weightRandomSample}(E^c_{add}, \lceil m \cdot \beta \rceil, W_{add})$;
Initialize $E^c_{del} = \emptyset$;
for $each(v_i, v_j) \in E_{add}$ **do**
    Get the length-l motif h^l_{ij} via path search: $h^l_{ij} \leftarrow \mathsf{pathSearch}(i, j, l)$;
    Compute the deletion weights W_{del} via Eq. (4.7) ;
    $e_{del} \leftarrow \mathsf{weightRandomSample}(h^l_{ij}, 1, W_{del})$;
    Add e_{del} to E^c_{del} ;
end
Get augmented graph G' via Eq. (4.4) ;
end ;
return G';

4.3.3 Data Filtration

Due to the topological dependency of graph structured data, the examples generated via subgraph augmentation may lose some original semantics. By assigning the label of the original graph to the generated graph during subgraph augmentation, one cannot determine whether the assigned label is reliable. Therefore, the concept of label reliability is employed here to measure the matching degree between examples and labels.

Each graph G_i in D_{val} will be fed into classifier C to obtain the prediction vector $\mathbf{p}_i \in \mathbb{R}^{|Y|}$, which represents the probability distribution as how likely an input example belongs to each possible class. Here, $|Y|$ is the number of classes for labels. Then, a probability confusion matrix $\mathbf{Q} \in \mathbb{R}^{|Y|\times|Y|}$, in which the entry q_{ij} represents the average probability that the classifier classifies the graphs of the i-th class to the j-th class, is computed as follows:

$$\begin{aligned} \mathbf{q}_k &= \frac{1}{\Omega_k} \sum_{y_i=k} \mathbf{p}_i \ , \\ \mathbf{Q} &= [\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_{|Y|}] \ , \end{aligned} \tag{4.8}$$

where Ω_k is the number of graphs belonging to the k-th class in D_{val} and $\mathbf{q}_k$ is the average probability distribution of the k-th class.

The label reliability of an example (G_i, y_i) is defined as the product of example probability distribution $\mathbf{p}_i$ and class probability distribution $\mathbf{q}_{y_i}$ as follows:

$$r_i = \mathbf{p}_i{}^\top \mathbf{q}_{y_i} \ . \tag{4.9}$$

A threshold θ used to filter the generated data is defined as

$$\theta = \arg\min_{\theta} \sum_{(G_i, y_i)\in D_{val}} \Phi[(\theta - r_i) \cdot g(G_i, y_i)] \ , \tag{4.10}$$

where $g(G_i, y_i) = 1$ if $C(G_i) = y_i$ and $g(G_i, y_i) = -1$ otherwise, and $\Phi(x) = 1$ if $x > 0$ and $\Phi(x) = 0$ otherwise.

4.3.4 Model Evolution Framework

Model evolution aims to iteratively optimize classifiers via subgraph augmentation, data filtration and model retraining, and ultimately improve the performance of graph classification. Figure 4.4 and Algorithm 2 demonstrate the workflow and the procedure of *M-Evolve*, respectively. Here, a variable, number of iterations T, is introduced for repeating the above workflow to continuously augment the dataset and optimize the classifier.

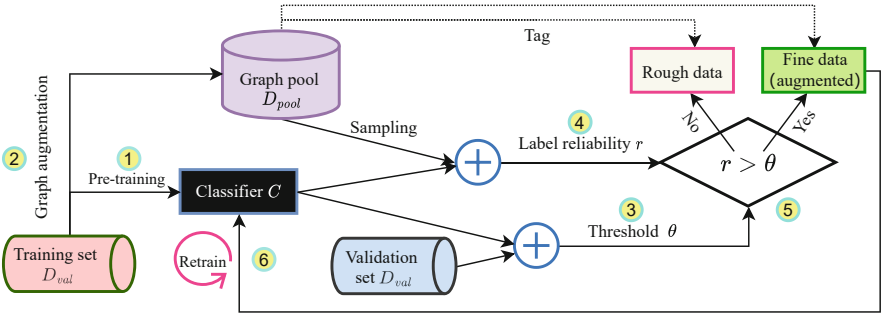

Fig. 4.4 The architecture of the model evolution. The complete workflow proceeds as follows: (1) pre-train graph classifier using training set; (2) apply subgraph augmentation to generate data pool; (3) compute the label reliability threshold using validation set; (4) compute the label reliability of examples sampled from graph pool; (5) filter data and obtain augmented set using threshold; (6) retrain graph classifier using the union of training set and augmented set

Algorithm 2: M-Evolve

Input: Training set D_{train}, validation set D_{val}, subgraph augmentation f, number of iterations T.
Output: Evolutive model C'

Pre-training classifier C using D_{train} and D_{val} ;
Initalize $iteration = 0$;
for $iteration < T$ **do**
  Graph augmentation: $D_{pool} \leftarrow f(D_{train})$;
  For all graphs G_i in D_{val} classified by C, get $\mathbf{p}_i$;
  Get probability confusion matrix $\mathbf{Q}$ via Eq. (4.8) ;
  For all graphs G_i in D_{val} classified by C, get r_i via Eq. (4.9);
  Get the label reliability threshold θ via Eq. (4.10) ;
  For all samples (G_i, y_i) in D_{pool} classified by C, compute r_i, **if** $r_i > \theta$, $D_{train}.append((G_i, y_i))$;
  Get evolutive classifier: $C' \leftarrow \mathsf{retrain}(C, D_{train})$;
  $iteration \leftarrow iteration + 1$;
  $C \leftarrow C'$;
end
end ;
return C';

4.4 Application of Subgraph Augmentation

The *M-Evolve* framework is compatible with different graph classification models. In order to explore the transferability of *M-Evolve* to other graph tasks, we unify both node classification and link prediction into graph classification pattern, as shown in Fig. 4.5. For node classification/link prediction, we first extract the local subgraph around each target node/link, and all the extracted subgraphs are treated as dataset to train a graph classifier, which learns a function mapping the subgraph patterns to node labels/link existence. In this chapter, we choose a recent deep neural architecture DGCNN [37] as the default graph classifier.

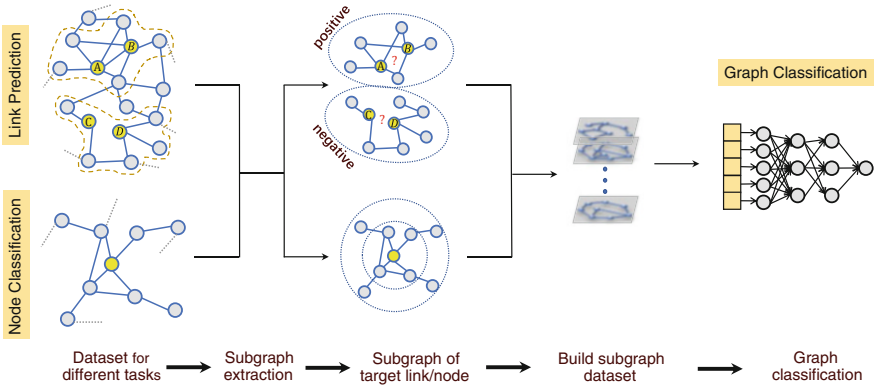

Fig. 4.5 Unify multiple tasks into graph classification. For link prediction, extract the local subgraph of target pairwise nodes, and the labels of subgraphs reflect the link existence. For node classification, extract the local subgraph of target nodes. The subgraph labels are equivalent to the corresponding node labels

4.4.1 Graph Classification

Graph classification focuses on identifying the category labels of graphs in a dataset, and plays a significant role in drug classification, toxicity detection, protein analysis etc. Widely used methods concentrate on graph kernels [8, 9, 14], graph embedding [16–18] and graph pooling [30, 31].

4.4.1.1 Experimental Setting

Data We evaluate the proposed methods on two benchmark datasets: Mutag [38] and PTC-MR [39]. The two datasets represent the graph collections of chemical compounds, in which nodes correspond to molecular structures and edges indicate chemical bonds between them. The specifications of datasets are given in Table 4.1.

Table 4.1 Dataset properties, where $|D|$ is the number of graphs in dataset, $|Y|$ is the number of classes for labels, $Avg.|V|$ / $Avg.|E|$ is the average number of nodes/edges, *bias* is the proportion of the dominant class and *Attri* is the feature dim of the node in dataset

Task	Dataset	$\|D\|$	$\|Y\|$	*Avg.* $\|V\|$	*Avg.* $\|E\|$	*bias* (%)	*Attri*
Graph classification	MUTAG	188	2	17.93	19.79	66.5	–
	PTC-MR	344	2	14.29	14.69	55.8	–
Link prediction	Router	3822	2	50.00	50.15	50	–
	Celegans	2964	2	90.11	225.25	50	–
Node classification	BlogCatalog	5196	6	67.11	359.57	16.7	8189
	Flickr	7575	9	64.30	1139.25	11.1	12,047

Parameter Settings We first split each dataset into training, validation and testing sets with a proportion of 7:1:2. We re-split the experimental dataset 25 times and report the average accuracy across all trials. We use the default setting of DGCNN architecture, i,e., four graph convolution layers with 32,32,32,1 channels and a dense layer with 128 neurons.

4.4.2 Link Prediction

Link prediction, which aims to uncover missing links or predicting future interactions between pairwise nodes based on observable links and other external information, has been applied to friend recommendation [40], knowledge graph completion [41] and network reconstruction [42]. Widely used methods concentrate on similarity-based algorithms [36], maximum likelihood methods [43, 44], probabilistic models [45, 46] and graph autoencoder (GAE) [47, 48].

4.4.2.1 Subgraph Extraction

For an original graph $G = (V, E)$, given target pairwise nodes $v_i, v_j \in V$, the h-hop subgraph for (v_i, v_j) is the $G^h_{i,j}$ induced from G by the set of nodes $\{v_k \mid d(v_k, v_i) \leq h \text{ or } d(v_k, v_j) \leq h\}$, where $d(a, b)$ is the length of shortest path between nodes a and b. The subgraph $G^h_{i,j}$ describes the "h-hop surrounding environment" of the target link (v_i, v_j).

Node Importance Labeling Since the size of a subgraph is constrained by the radiation order of the target link, nodes with different relative positions to the target link have different structural importance, i.e., having different contributions to the prediction of link existence. Empirically, in a graph, pairwise nodes that are close to each other generally have higher influence than those that are far apart, so the natural to set the node importance label based on the distance of nodes from the central pairwise nodes (target link) in the subgraph. We set the node importance label using the Double-Radius Node Labeling (DRNL) criterion [7]:

$$f_{lp}(k) = 1 + \min\left(d_i, d_j\right) + (d/2)[(d/2) + (d\%2) - 1] \tag{4.11}$$

where $d_i := d(k, i)$, $d_j := d(k, j)$, $d := d_i + d_j$, $(d/2)$ and $(d\%2)$ are the integer quotient and remainder of d divided by 2. And for those nodes with $d(k, i) = \pm\infty$ or $d(k, j) = \pm\infty$, we give them a null label 0. Figure 4.6 shows an example of node importance labeling via DRNL. These node importance labels are uniquely hot-coded and followed by the node's own attribute features as node features for subsequent graph classification tasks.

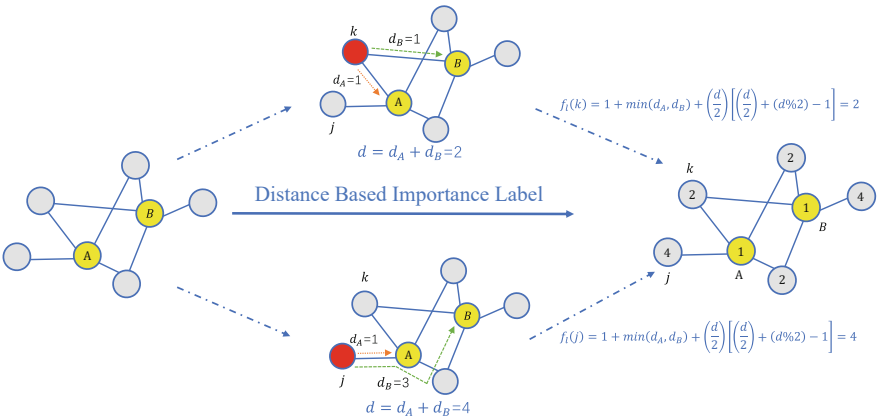

Fig. 4.6 An example for node importance labeling in link prediction

4.4.2.2 Experimental Setting

Data We evaluate the proposed methods on two benchmark datasets: Router [49] and Celegans [50]. Router is router-level Internet and Celegans is the neural network of Celegans. The specifications of datasets are given in Table 4.1.

Baselines To verify the effectiveness of DGCNN in solving link prediction problem, we compare DGCNN with GNN-based link prediction methods: graph autoencoder (GAE) and variational graph autoencoder (VGAE) [47].

Parameter Settings We split each dataset into training, validation and testing sets with a proportion of 1:1:3. We re-split the experimental dataset 25 times and report the average accuracy across all trials. We adjust the setting of DGCNN architecture, i,e., four graph convolution layers with 32,32,32,1 channels and a dense layer with 256 neurons. For subgraph extraction, we set the neighborhood hop counts h to 2.

4.4.3 Node Classification

Node classification is typically a semi-supervised learning task, in which only a few nodes' labels are known for predicting the labels of other nodes in graph datasets. It is widely used for recommendation, entity alignment of knowledge graphs, etc. Common methods include walk-based embedding methods [51, 52], spectral domain convolution methods [53], and spatial domain convolution methods [54].

4.4.3.1 Subgraph Extraction

For an original graph $G = (V, E)$, given a target node $v_i \in V$, the h-hop subgraph for v_i is the G_i^h induced from G by the set of nodes $\{v_k \mid d(v_k, v_i) \leq h\}$.

Node Importance Labeling Similar to the node importance labeling in link prediction task, we set the importance label of nodes by their distances from the target node:

$$f_{nc}(k) = d(k, i) \ . \tag{4.12}$$

4.4.3.2 Experimental Setting

Data We evaluate the proposed methods on two social datasets: BlogCatalog and flickr [55], in which the posted keywords or tags are used as node attribute information. The specifications of datasets are given in Table 4.1.

Baseline To verify the effectiveness of DGCNN in solving link prediction problem, we compare DGCNN with popular node classification deep models: graph convolution network (GCN) [6] and graph attention network (GAT) [56].

Parameter Settings We also split each dataset into training, validation and testing sets with a proportion of 1:2:7, We re-split the experimental dataset 25 times and report the average accuracy across all trials. We adjust the setting of DGCNN architecture, i,e., two graph convolution layers with 32,1 channels and a dense layer with 256 neurons. For subgraph extraction, we set the neighborhood hop counts h to 1.

4.4.4 Experimental Results

We unify multiple tasks into graph classification pattern, and use DGCNN architecture to complete all tasks by graph classification. Tables 4.2, 4.3 and 4.4 report the results of performance comparison among baselines, DGCNN and the evolutive models for different tasks.

First, from the comparison between DGCNN and the evolutive models, one can see that there is a boost in classification performance across all six datasets, indicating that our *M-Evolve* can effectively improve the performance of graph classification model. We speculate that the original DGCNN models trained with limited training data are overfitting, and on the contrary, *M-Evolve* enriches the scale of training data via subgraph augmentation and optimizes graph classifiers via iterative retraining, therefore can improve the generalization and avoid overfitting to a certain extent.

Table 4.2 Graph classification results of original and evolutive models. The best results are marked in bold. The far-right column gives the average relative improvement rate (Avg. RIMP) in accuracy

Dataset	Mapping	Budget			Avg. RIMP
		0.10	**0.15**	**0.20**	
MUTAG	DGCNN	0.8447			–
	DGCNN + random	0.8447	0.8533	**0.8458**	+**0.38**%
	DGCNN + m-s	**0.8450**	**0.8547**	0.8436	+0.36%
PTC_MR	DGCNN	0.5775			–
	DGCNN + random	0.5739	0.5764	**0.5860**	+0.22%
	DGCNN + m-s	**0.5849**	**0.5962**	0.5733	+**1.26**%

Table 4.3 Link prediction results in baselines, DGCNN and evolutive models. The best results are marked in bold. The far-right column gives the average relative improvement rate (Avg. RIMP) in accuracy

Dataset	Mapping	Budget			Avg. RIMP
		0.10	**0.15**	**0.20**	
Router	GAE	0.5130			–
	VGAE	0.4999			–
	DGCNN	0.6721			–
	DGCNN + random	0.6430	0.6512	0.6694	−2.6%
	DGCNN + m-s	**0.6858**	**0.6852**	**0.6854**	+**1.7**%
Celegans	GAE	0.5256			–
	VGAE	0.5053			–
	DGCNN	0.6323			–
	DGCNN + random	0.6170	0.6125	0.6176	−2.6%
	DGCNN + m-s	**0.6353**	**0.6379**	**0.6379**	+**0.7**%

Table 4.4 Node classification results in baselines, DGCNN and evolutive models. The best results are marked in bold. The far-right column gives the average relative improvement rate (Avg. RIMP) in accuracy

Dataset	Mapping	Budget			Avg. RIMP
		0.10	**0.15**	**0.20**	
Blog	GCN	0.7200			–
	GAT	0.6630			–
	DGCNN	0.7453			–
	DGCNN + random	0.7502	0.7493	**0.7483**	+0.53%
	DGCNN + m-s	**0.7589**	**0.7560**	0.7457	+**1.10**%
Flickr	GCN	**0.5460**			–
	GAT	0.3590			–
	DGCNN	0.4192			–
	DGCNN + random	0.4471	0.4499	0.4505	+7.15%
	DGCNN + m-s	0.4888	0.4884	0.5014	+**17.57**%

Now, we define the relative improvement rate (RIMP) in accuracy as follows:

$$RIMP = \frac{Acc_{en} - Acc_{ori}}{Acc_{ori}}, \tag{4.13}$$

where Acc_{en} and Acc_{ori} refer to the accuracy of the evolutive and the original models, respectively. In Tables 4.2, 4.3 and 4.4, the far-right column gives the average relative improvement rate (Avg. *RIMP*) in accuracy, from which one can see that the *M-Evolve* combined with motif-similarity (m-s) mappings obtains better results overall. These results indicate that both similarity and motif mechanisms play positive roles in enhancing graph classification. As a reasonable explanation, similarity mechanism tends to link nodes with higher similarity and is capable of optimizing topological structure legitimately, which is similar to the finding method in [57]. And the motif mechanism achieves edge modification via local edge swapping, which has subtle effect on both the degree distribution and the clustering coefficient of the graph.

We further investigate the effectiveness of unifying both link prediction and node classification tasks into graph classification patterns. Specifically, we compare the performances of DGCNN and baselines. Notably, we use small training set that only accounts for 20%/10% of the entire dataset on link prediction/node classification. For link prediction, DGCNN significantly outperforms baselines since both GAE and VGAE have poor performances when being trained with extremely sparse graphs. For node classification, DGCNN outperforms baselines on Blog, but has poorer performance than GCN. As a reasonable explanation, when constructing the subgraph dataset, we extract 1-hop subgraph in order to reduce the running cost of the algorithm, while GCN aggregates the 2-hop neighborhoods and uses more neighborhood information. Overall, DGCNN achieves competitive performances against baselines in both link prediction and node classification tasks, indicating that the idea of unifying both link prediction and node classification tasks into graph classification is effective.

4.5 Conclusion

In this chapter, we unify node classification and link prediction tasks into graph classification patterns, and introduce a concept of subgraph augmentation to generate weakly labeled data for small benchmark datasets through heuristic transformations of graph structures. In addition, we propose a general model evolution framework that combines subgraph augmentation, data filtering, and model retraining, to optimize a pre-trained graph classifier. Experiments on six datasets show that our proposed framework performs well and helps existing graph classification models mitigate the overfitting problem when being trained on small-scale benchmark datasets, and achieve significant improvement in classification performance.

References

1. Borgwardt, K.M., Ong, C.S., Schönauer, S., Vishwanathan, S.V.N., Smola, A.J., Kriegel, H.-P.: Protein function prediction via graph kernels. Bioinformatics **21**(suppl_1), i47–i56 (2005)
2. Duvenaud, D.K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A., Adams, R.P.: Convolutional networks on graphs for learning molecular fingerprints. In: Advances in Neural Information Processing Systems, pp. 2224–2232 (2015)
3. Yanardag, P., Vishwanathan, S.V.N.: Deep graph kernels. In: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1365–1374 (2015)
4. Zhou, J., Shen, J., Xuan, Q.: Data augmentation for graph classification. In: Proceedings of the 29th ACM International Conference on Information & Knowledge Management, pp. 2341–2344 (2020)
5. Zhou, J., Shen, J., Yu, S., Chen, G., Xuan, Q.: M-evolve: structural-mapping-based data augmentation for graph classification. In: IEEE Transactions on Network Science and Engineering, pp. 1–1 (2020)
6. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: International Conference on Learning Representations (ICLR) (2017)
7. Zhang, M., Chen, Y.: Link prediction based on graph neural networks. In: Advances in Neural Information Processing Systems, pp. 5165–5175 (2018)
8. Shervashidze, N., Schweitzer, P., Van Leeuwen, E.J., Mehlhorn, K., Borgwardt, K.M.: Weisfeiler-Lehman graph kernels. J. Mach. Learn. Res. **12**(9) (2011)
9. Neumann, M., Garnett, R., Bauckhage, C., Kersting, K.: Propagation kernels: efficient graph kernels from propagated information. Mach. Learn. **102**(2), 209–245 (2016)
10. Shervashidze, N., Vishwanathan, S.V.N., Petri, T., Mehlhorn, K., Borgwardt, K.: Efficient graphlet kernels for large graph comparison. In: Artificial Intelligence and Statistics, pp. 488–495 (2009)
11. Gärtner, T., Flach, P., Wrobel, S.: On graph kernels: hardness results and efficient alternatives. In: Learning Theory and Kernel Machines, pp. 129–143 (Springer, Berlin, 2003)
12. Kashima, H., Tsuda, K., Inokuchi, A.: Marginalized kernels between labeled graphs. In: Proceedings of the 20th International Conference on Machine Learning (ICML-03), pp. 321–328 (2003)
13. Mahé, P., Ueda, N., Akutsu, T., Perret, J.-L., Vert, J.-P.: Extensions of marginalized graph kernels. In: Proceedings of the twenty-first International Conference on Machine Learning, p. 70 (2004)
14. Sugiyama, M., Borgwardt, K.: Halting in random walk kernels. In: Advances in Neural Information Processing Systems, pp. 1639–1647 (2015)
15. Borgwardt, K.M., Kriegel, H.-P.: Shortest-path kernels on graphs. In: Fifth IEEE International Conference on Data Mining (ICDM'05), p. 8. IEEE, Piscataway (2005)
16. Cai, H., Zheng, V.W., Chang, K.C.-C.: A comprehensive survey of graph embedding: problems, techniques, and applications. IEEE Trans. Knowl. Data Eng. **30**(9), 1616–1637 (2018)
17. Chen, F., Wang, Y.-C., Wang, B., Jay Kuo, C.-C.: Graph representation learning: a survey. In: APSIPA Transactions on Signal and Information Processing, vol. 9 (2020)
18. Fu, C., Zheng, Y., Liu, Y., Xuan, Q., Chen, G.: Nes-tl: network embedding similarity-based transfer learning. IEEE Trans. Netw. Sci. Eng. **7**(3), 1607–1618 (2019)
19. Guo, W., Shi, Y., Wang, S., Xiong, N.N.: An unsupervised embedding learning feature representation scheme for network big data analysis. IEEE Trans. Netw. Sci. Eng. **7**(1), 115–126 (2019)
20. Narayanan, A., Chandramohan, M., Venkatesan, R., Chen, L., Liu, Y., Jaiswal, S.: graph2vec: learning distributed representations of graphs. Preprint. arXiv:1707.05005 (2017)
21. Dai, H., Dai, B., Song, L.: Discriminative embeddings of latent variable models for structured data. In: International Conference on Machine Learning, pp. 2702–2711 (2016)

22. Narayanan, A., Chandramohan, M., Chen, L., Liu, Y., Saminathan, S.: subgraph2vec: learning distributed representations of rooted sub-graphs from large graphs. Preprint. arXiv:1606.08928 (2016)
23. Gilmer, J., Schoenholz, S.S., Riley, P.F., Vinyals, O., Dahl, G.E.: Neural message passing for quantum chemistry. Preprint. arXiv:1704.01212 (2017)
24. Li, Y., Tarlow, D., Brockschmidt, M., Zemel, R.: Gated graph sequence neural networks. Preprint. arXiv:1511.05493 (2015)
25. Jin, Y., JaJa, J.F.: Learning graph-level representations with recurrent neural networks. Preprint. arXiv:1805.07683 (2018)
26. You, J., Ying, R., Ren, X., Hamilton, W.L., Leskovec, J.: GraphRNN: generating realistic graphs with deep auto-regressive models. Preprint. arXiv:1802.08773 (2018)
27. Defferrard, M., Bresson, X., Vandergheynst, P.: Convolutional neural networks on graphs with fast localized spectral filtering. In: Advances in Neural Information Processing Systems, pp. 3844–3852 (2016)
28. Simonovsky, M., Komodakis, N.: Dynamic edge-conditioned filters in convolutional neural networks on graphs. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 3693–3702 (2017)
29. Fey, M., Lenssen, J.E., Weichert, F., Müller, H.: SplineCNN: fast geometric deep learning with continuous b-spline kernels. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 869–877 (2018)
30. Ying, Z., You, J., Morris, C., Ren, X., Hamilton, W., Leskovec, J.: Hierarchical graph representation learning with differentiable pooling. In: Advances in Neural Information Processing Systems, pp. 4800–4810 (2018)
31. Ma, Y., Wang, S., Aggarwal, C.C., Tang, J.: Graph convolutional networks with eigenpooling. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 723–731 (2019)
32. Zhao, T., Liu, Y., Neves, L., Woodford, O., Jiang, M., Shah, N.: Data augmentation for graph neural networks. Preprint. arXiv:2006.06830 (2020)
33. Wang, Y., Wang, W., Liang, Y., Cai, Y., Liu, J., Hooi, J.: NodeAug: semi-supervised node classification with data augmentation. In: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 207–217 (2020)
34. Spinelli, I., Scardapane, S., Scarpiniti, M., Uncini, A.: Efficient data augmentation using graph imputation neural networks. In: Progresses in Artificial Intelligence and Neural Systems, pp. 57–66 (Springer, Berlin, 2020)
35. Spinelli, I., Scardapane, S., Uncini, A.: Missing data imputation with adversarially-trained graph convolutional networks. Neural Netw. **129**, 249–260 (2020)
36. Zhou, T., Lü, L., Zhang, Y.-C.: Predicting missing links via local information. Eur. Phys. J. B **71**(4), 623–630 (2009)
37. Zhang, M., Cui, Z., Neumann, M., Chen, Y.: An end-to-end deep learning architecture for graph classification. In: Thirty-Second AAAI Conference on Artificial Intelligence (2018)
38. Debnath, A.K., Lopez de Compadre, R.L., Debnath, G., Shusterman, A.J., Hansch, C.: Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. Correlation with molecular orbital energies and hydrophobicity. J. Med. Chem. **34**(2), 786–797 (1991)
39. Helma, C., King, R.D., Kramer, S., Srinivasan, A.: The predictive toxicology challenge 2000–2001. Bioinformatics **17**(1), 107–108 (2001)
40. Adamic, L.A., Adar, E.: Friends and neighbors on the web. Soc. Netw. **25**(3), 211–230 (2003)
41. Nickel, M., Murphy, K., Tresp, V., Gabrilovich, E.: A review of relational machine learning for knowledge graphs. Proc IEEE **104**(1), 11–33 (2015)
42. Oyetunde, T., Zhang, M., Chen, Y., Tang, Y., Lo, C.: BoostGAPFILL: improving the fidelity of metabolic network reconstructions through integrated constraint and pattern-based methods. Bioinformatics **33**(4), 608–611 (2017)
43. Clauset, A., Moore, C., Newman, M.E.J.: Hierarchical structure and the prediction of missing links in networks. Nature **453**(7191), 98–101 (2008)

44. White, H.C., Boorman, S.A., Breiger, R.L.: Social structure from multiple networks. I. Blockmodels of roles and positions. Am. J. Soc. **81**(4), 730–780 (1976)
45. Getoor, L., Friedman, N., Koller, D., Pfeffer, A.: Learning probabilistic relational models. In: Relational Data Mining, pp. 307–335. Springer, Berlin (2001)
46. Heckerman, D., Meek, C., Koller, D.: Probabilistic entity-relationship models, PRMS, and plate models. In: Introduction to Statistical Relational Learning, pp. 201–238 (2007)
47. Kipf, T.N., Welling, M.: Variational graph auto-encoders. Preprint. arXiv:1611.07308 (2016)
48. Pan, S., Hu, R., Long, G., Jiang, J., Yao, L., Zhang, C.: Adversarially regularized graph autoencoder for graph embedding. Preprint. arXiv:1802.04407 (2018)
49. Spring, N., Mahajan, R., Wetherall, D., Anderson, T.: Measuring ISP topologies with rocketfuel. IEEE/ACM Trans. Netw. **12**(1), 2–16 (2004)
50. Watts, D.J., Strogatz, S.H.: Collective dynamics of 'small-world' networks. Nature **393**(6684), 440–442 (1998)
51. Perozzi, B., Al-Rfou, R., Skiena, S.: Deepwalk: online learning of social representations. In: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 701–710 (2014)
52. Grover, A., Leskovec, J.: node2vec: Scalable feature learning for networks. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 855–864 (2016)
53. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. Preprint. arXiv:1609.02907 (2016)
54. Hamilton, W., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. In: Advances in Neural Information Processing Systems, pp. 1024–1034 (2017)
55. Wu, J., He, J., Xu, J.: Demo-net: degree-specific graph neural networks for node and graph classification. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, New York (2019)
56. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y.: Graph attention networks. In: International Conference on Learning Representations (2018). A poster
57. Zheng, M., Allard, A., Hagmann, P., Alemán-Gómez, Y., Serrano, M.Á.: Geometric renormalization unravels self-similarity of the multiscale human connectome. Proc. Natl. Acad. Sci. **117**(33), 20244–20253 (2020)