Cross-Graph: Robust and Unsupervised Embedding for Attributed Graphs with Corrupted Structure

Chun Wang[†], Bo Han[‡], Shirui Pan[§], Jing Jiang[†], Gang Niu[¶], Guodong Long[†]
[†]Australian Artificial Intelligence Institute, University of Technology Sydney, Australia
[‡]Department of Computer Science, Hong Kong Baptist University, Hong Kong SAR, China
[§]Faculty of Information Technology, Monash University, Australia
[¶]RIKEN Center for Advanced Intelligence Project, Japan
chun.wang-1@student.uts.edu.au, bhanml@comp.hkbu.edu.hk, shirui.pan@monash.edu,
jing.jiang@uts.edu.au, gang.niu@riken.jp, guodong.long@uts.edu.au

***Abstract*—Graph embedding has shown its effectiveness to represent graph information and capture deep relationships in graph data. Most recent graph embedding methods focus on attributed graphs, since they preserve both structure and content information in the network. However, corruption can exist in the graph structure as well as the node content of the graph, and both can lead to inferior embedding results. Unfortunately, few existing graph embedding algorithms have considered the corruption problem, and to the best of our knowledge, none has studied structural corruption in attributed graphs, including missing and redundant edges. This field is difficult for previous methods, mainly due to two challenges: (1) the existence of various corruption causes has made it difficult to recognize corruptions in graphs, and (2) the complexity of graph-structured data has increased the difficulty of handling corruption therein for graph embedding methods. These facts lead us here to propose a novel autoencoder-based graph embedding approach, which is robust against structural corruption. Our idea comes from the recent discovery of memorization effects in deep learning. Namely, deep neural networks prefer to fit clean data first, before they over-fit corrupted data. Specifically, we train two autoencoders simultaneously and let them learn the reliability of the edges in the graph from each other. The two autoencoders would evaluate the edges according to their reconstructed structure and manipulate this by devaluing those distrusted edges to update the structure information. The updated structure would be used further in the next iteration as the ground-truth of its peer-network. Experiments on different versions of real-world graphs show state-of-the-art results and demonstrate the robustness of our model against structural corruption.**

***Index Terms*—Graph autoencoder, graph convolutional network, network representation, structural corruption.**

I. Introduction

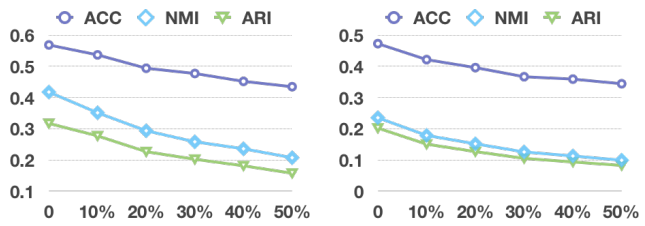
Graphs have attracted much more attention in recent years with the development of networked applications, such as social networks, citation networks, knowledge graphs and protein-protein interactions [1], [2]. Unlike traditional data format, graphs are adept at characterizing individual properties as well as capturing the pairwise relationships between the individuals in the networks. The complexity of graph information has made graph analyzing significant, yet challenging.

For the past few years, graph embedding has evolved as a general solution to various graph analyzing tasks [3], [4]. Its main strength is to preserve and combine different sides of graph information into a unified low-dimensional feature space. Based on the learnt embedding, classical task-oriented methods could be applied to handle various tasks such as classification [5], clustering [6], [7] or link prediction [8], which would otherwise be complicated for graph data.

Recent graph embedding methods lean towards embedding attributed graphs for more comprehensive graph information. Attributed graph information consists of node content and edge connections between the nodes. Corruption, like noise or outliers, can occur in both aspects and affect the analysis of graph data. However, previous embedding works have not pay enough attention to the corruption in attributed graphs. A few works tried to detect isolated outlier nodes out of the graph [9], [10], but they were not able to recover the graphs' property from systematic corruption. So, they are not able to benefit the other graph analyzing tasks applied to the same graph. On the other hand, little previous works question the structure information in the provided graph data.

Edges could be citations among academic papers or friend relationships in a social network, etc.. They only represent some certain kinds of relationship between the two nodes and are always sparse in a graph. In other words, the disconnection of two nodes is commonplace in graphs and could always make some sense. Furthermore, current existing studies are able to help against one aspect of structural corruption, namely missing edges. This problem can be solved quite well by various link-prediction methods [11]. Our intention, in this paper, is to focus on the other aspect of structural corruption, namely redundant edges. We therefore define the structural corruption in this paper as redundant edge connections, rather than the missing ones.

Broadly speaking, that redundant edges improperly connect two nodes is ubiquitous and considerably more problematical. These edges can be generated by malicious nodes, like robot account, to influence its neighbors or hide itself, or might be created unintentionally by the users or systems [12]. Adversarial poisoning attacks on graph data also tend to add edges to bring noise to the learnt node embedding [13], [14]. Furthermore, with the rapid development of graph learning, many CV or NLP-oriented methods also employ graph neural networks. They construct graphs from their plain data with a graph kernel. Such constructed graphs are always dense with



(a) Cora dataset (b) Citeseer dataset

Fig. 1. We randomly add redundant edges to the Cora and Citeseer graph structure and then run Graph Autoencoder (GAE) on them for 10 times to record the average clustering performance evaluated by 3 clustering metrics. The X-axis shows the number of added edges represented as percentage to the number of original edges. It shows that redundant edges can easily ruin the performance of the graph.

a high percentage of redundancies. Even in a hypothetical "clean" graph, some of the edges could be regarded as partly abnormal for a given task. For instance, in citation networks, it is common for academic papers to cite weakly-related papers, which will appear the same as those key citations in the graph; in social networks, a parent-child connection may be considered abnormal when classifying users according to their hobbies, since they are connected due to family relationship and are unlikely to have similar hobbies.

Though effective graph embedding methods have emerged endlessly in recent years, most of these embedding methods are based on the assumption that they have no difficulty accessing perfect graph-structure data. This assumption is too ideal to hold in real-world problems and may limit the efficacy of the learnt embedding. As illustrated in Fig. 1, the performance of the embedding can be ruined easily, simply by adding random redundant edges to the graph. In a word, a graph embedding method robust against structural corruption is highly desirable.

To deal with this problem, we propose a novel Cross-Graph framework, to learn robust graph embedding, strengthened against structural corruption. Since label information is not easily accessible either, we decide to use autoencoders to perform unsupervised learning. Our autoencoder-based method can learn effective embedding without access to, not only the label guidance, but also clean graph-structure data. We are inspired by the Co-training approach [15], and designed a dual graph interaction framework called Cross-Graph Learning. Based on the deep learning memorization effect that deep neural networks fit clean data first [16], [17], we maintain two autoencoders and update them alternatively. In each iteration, since the trustworthy edges fit faster and will be closer to the ground-truth, the two autoencoders can evaluate the reliability of every graph edge with their reconstructed graph structure. Each autoencoder then updates its structure by slightly devaluing those distrusted edges. This updated graph structure is then passed to its peer-autoencoder, working as a provided "opinion" on how the real structure should present. The peer-autoencoder would take this updated structure as the input to the next iteration. Through the learning process, those redundant edges will be devalued faster, over and over again, and eventually filtered out. Since the two autoencoders have different embedding abilities, different types of corruptions may be selected out, including some misjudgments. Meanwhile, benefit from our interactive process, these misjudgments caused by a single autoencoder, can be reduced by its peer one. This fact further strengthened the robustness of our model.

Our contribution can be summarized as follows:

- To the best of our knowledge, we are the first to discuss the influence of structural corruption, especially redundant edges, on attributed graph embedding. We show that corrupted graph structure could ruin the performance of the graph embedding learned on the basis of it.
- We propose a Cross-Graph strategy to learn graph embedding based on the graph structure and node content, which is unsupervised and robust against structural corruption.
- We conduct extensive experiments, compare our model with novel unsupervised graph embedding baselines on various kinds of structure-corrupted graphs. The results show that our Cross-Graph strategy significantly improves the performance when confronting structural corruption.

II. Related Literature

We briefly review related works, including graph embedding models, graph autoencoder, co-training-based methods and outlier-oriented graph models.

A. Graph Embedding Models

Based on the information available, graph embedding models can be separated into two categories: topological and attributed graph.

Topological embedding models are provided only with the graph structure information and subsequently focus on exploring and preserving this information. DeepWalk [18] uses random walk to generate context for graph nodes embedding learning; matrix factorization approaches like M-NMF [19] and HOPE [20] learn graph representation from the transformation of the adjacency matrix; probabilistic models, like LINE [21] and node2vec [22], have also been developed.

Later, graph research focused more on attributed graph embedding, since it has proven effective when node attributes are available. Many algorithms have been proposed to exploit and embed them simultaneously. TADW [23] uses a matrix factorization approach to model the content and graph structure interaction.

These models employ various approaches. However, these are generally old or traditional approaches that cannot compete with recent deep embedding methods. Also, these models all consider their data real and clean and, as such, are vulnerable to corruptions.

B. Graph Autoencoders for Unsupervised Learning

We focus on robust learning where no clean training data is available. Since label information is not easily accessible

either, we consider an unsupervised embedding to be more valuable and give this our attention.

Graph Neural Networks (GNNs) [1], [24] have shown great ability dealing with different graph-based problems. Knowledge graphs [25], heterogeneous graphs [26], graph with time series [27] can all be modeled with different GNNs.

When it comes to unsupervised learning, the graph autoencoder stands out from various GNNs as a kind of mainstream solution, as it can encode graph data and learn latent representation in a purely unsupervised manner without accessing label information.

Early methods use an autoencoder to exploit graph structure information only. They adopt either traditional denoising autoencoder [28] or sparse autoencoder [29], to learn representation of the nodes from their topological relevance. Later on, based on graph convolutional networks (GCNs) [5] and the variational autoencoder, (variational) graph autoencoder (GAE & VGAE) [30] are developed, allowing the encoding of both the graph structure and node content information into a unified latent representation. Thereupon, ARGA [31] manipulates the autoencoder-learned embedding with an adversarial regularizer. MGAE [6] uses a marginalized single-layer autoencoder to learn embedding for graph clustering.

However, the default of all these graph autoencoders is that their provided graph is real and clean. None has considered their performance on graph structure with corruptions, e.g., redundant edges.

C. Co-training based Methods

To learn from noisy data, a promising approach is to filter out some clean data for training. The Co-training strategy [15], though not focusing on this problem, has made it easy to solve by training two learning algorithms separately and has derived many methods based on this approach. MentorNet [32] pretrains an extra network to select clean instances and supervise the training of the StudentNet; Decoupling [33] trains two networks simultaneously, and only uses those samples leading to different predictions in the two networks for updating; Co-teaching [34] also maintains two networks, and each network selects small-loss instances in each mini-batch to guide its peer-network.

Unfortunately, all these methods are designed for one channel of plain data, like images, and not well-suited to the graph domain.

D. Outlier-Oriented Graph Models

Outlier has been a significant research topic for decades. For plain graph, Zhang et al. proposed an outlier edge detection method [12] by comparing the actual and expected number of edges in an ego-network.

Specially for outliers in attributed graph data, most previous methods only try to detect outlier nodes. Radar [9] defines three kinds of outlier nodes in the attributed graph, and detects them by analyzing the residuals of the attribute information; ALAD [10] exploits the attributed graph information by context extraction and a mini-batch SGD-based method is developed to accelerate its optimization.

Nevertheless, these methods only consider detecting isolated outlier, not robust learning on the corrupted graph, and they are not able to recover the graph from large-scale corruption. What is more, they investigate only outlier nodes, neglecting the possibility of edges being the outlier.

III. Problem Definition

We consider unsupervised graph embedding on attributed graphs in this paper. A graph is represented as $G = (V, E, X)$, in which $V = \{v_i\}_{i=1,\cdots,n}$ consists of a set of nodes, $E = \{e_{ij}\}$ is a set of edges between nodes. The topological structure of graph G can be simplified as an adjacency matrix A, where $A_{i,j} = 1$ if $(v_i, v_j) \in E$; otherwise $A_{i,j} = 0$. $X = \{x_1; \ldots; x_n\}$ are the attribute values where $x_i \in R^m$ is a m-dimension real-value attribute vector associated with vertex v_i.

In our setting, the graph structure is a corrupted one with redundant edges. In other words, the adjacency matrix $A \neq A_{clean}$, where it has many more extra 1 values instead of 0 in the corresponding positions of the redundant edges.

Our purpose is to learn latent representations $Z \in \mathbb{R}^{n \times d}$ to map the nodes $v_i \in V$ to a low-dimensional space, in which $z_i^\top$ mapping v_i is the i-th row of the matrix Z. For previous graph embedding methods, learning with the corrupted structure A would result in much worse embedding result compared with the one learned from A_{clean}, while we aim to keep up its performance against the structural corruption.

IV. Proposed Method

We present our proposed Cross-Graph framework in this section. After briefly introducing a basic graph autoencoder, we propose the Cross-Graph framework for robust learning. The mechanism and some analysis of our model are also presented.

A. Graph Autoencoder

The graph autoencoder aims to learn low-dimensional embedding of each node based on the graph data $G = (V, E, X)$. The main idea is to integrate both graph structure A and node content X through an encoder into a latent embedding Z, and reconstruct the graph structure A through a decoder to optimize Z.

Graph Convolutional Encoder: One of the most basic and popular kind of graph encoder is developed as a variant of the graph convolutional network (GCN) [5], [35]. It encodes both graph structure A and node content X into a hidden representation by extending the operation of convolution to the graph domain, and performs a layer-wise transformation by a spectral convolution function $f(Z^{(l)}, A)$:

$$Z^{(l+1)} = f(Z^{(l)}, A) = \sigma(\widetilde{D}^{-\frac{1}{2}}\widetilde{A}\widetilde{D}^{-\frac{1}{2}}Z^{(l)}W^{(l)}). \quad (1)$$

Here, σ is a nonlinerity function. $Z^{(l)}$ is the input of the l-th layer transformation function, and $Z^{(l+1)}$ is the corresponding output. $\widetilde{D}^{-\frac{1}{2}}\widetilde{A}\widetilde{D}^{-\frac{1}{2}}$ is an approximation of the spectral

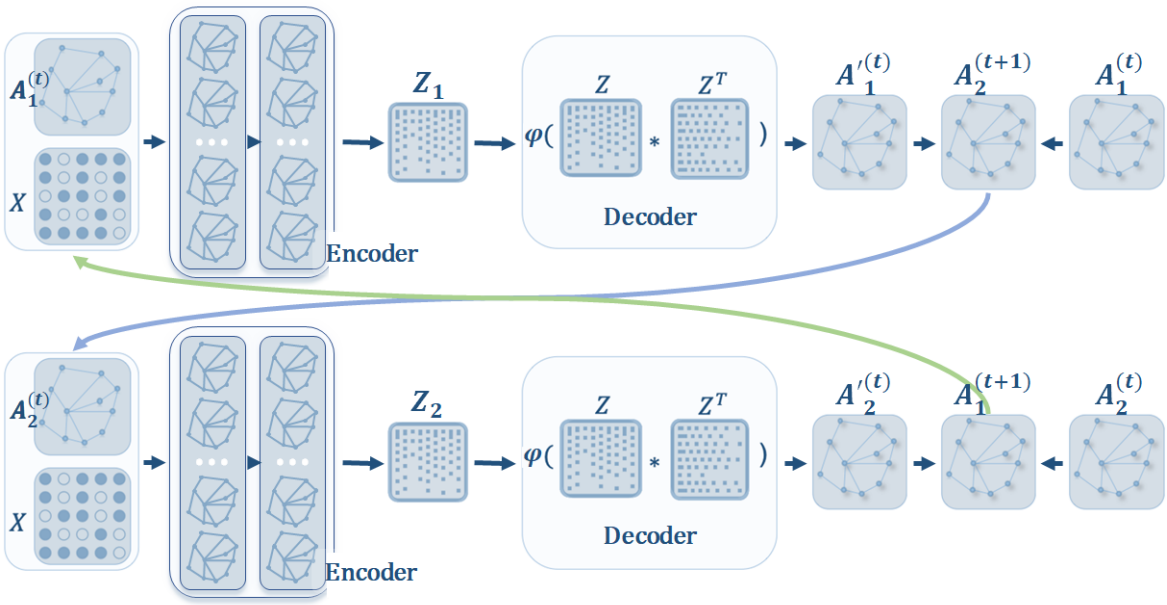


Fig. 2. Conceptual framework of Cross-Graph Autoencoder. Given a graph G with graph structure A and node content X, we maintain two autoencoders. Each autoencoder encodes A and X into a latent embedding Z, and then a decoder tries to reconstruct the structure A from Z and obtains A'. We regard A' partly as a reliability score of the edges in A and manipulate A according to it. Two updated A's are thereby formed and passed to the peer-autoencoder as the input for the next iteration.

convolution transformation $I + D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$, with $\widetilde{A} = A + I$, $\widetilde{D}_{ii} = \sum_j \widetilde{A}_{ij}$, and I being the identity matrix.

The standard graph encoder $Enc(X, A)$ takes the node content X as the input to the first layer $Z^{(1)}$, and stacks two graph convolution layers to obtain a hidden representation Z:

$$Z = Enc(X, A) = f(f(X, A), A). \tag{2}$$

Inner Product Decoder: The decoder reconstructs the graph data from hidden representation. Previous works have tried to reconstruct different sides of the graph information, and reconstructing the graph structure A has been validated as the best solution, since it also can be more flexible and fit situations in which no content information is available. The inner product decoder is commonly used here to predict whether there is a link between two nodes. The reconstructed link prediction layer $Dec(Z)$ is trained based on the hidden graph representation:

$$A' = Dec(Z) = \text{sigmoid}(Z^\top Z), \tag{3}$$

where A' is the reconstructed structure matrix of the input graph.

Reconstruction Loss: The graph autoencoder learns to optimize the latent representation Z by minimizing the reconstruction error measuring the binary cross-entropy loss between the original structure A and the reconstructed A':

$$\mathcal{L} = \text{cross-entropy}(A, A'). \tag{4}$$

B. Cross-Graph Learning Framework

Graph autoencoders have superior ability embedding attributed graphs, but they lack protection against graph corruptions. Specially facing structural corruption, adding redundant edges to the graph substantially weakens the performance of the graph autoencoders (as shown in Fig. 1). To confront this challenge, we develop a Cross-Graph learning framework.

Our framework trains two graph autoencoders simultaneously (Fig. 2). In each epoch t, the two autoencoders take their graph structure $A_1^{(t)}$ and $A_2^{(t)}$ (which are both A in the first epoch) as the input, and obtain their reconstructed structure $A_1'^{(t)}$ and $A_2'^{(t)}$:

$$A'^{(t)}_1 = Dec_1(Enc_1(X, A_1^{(t)})), \tag{5}$$

$$A'^{(t)}_2 = Dec_2(Enc_2(X, A_2^{(t)})), \tag{6}$$

$$A_1^{(1)} = A_2^{(1)} = A. \tag{7}$$

Then, new structure matrix $A_1^{(t+1)}$ and $A_2^{(t+1)}$ are constructed, as a compromise between the input graph structure $A_1^{(t)}\&A_2^{(t)}$ and the reconstructed $A'^{(t)}_1\&A'^{(t)}_2$:

$$A_1^{(t+1)} = (1-\gamma)A_2^{(t)} + \gamma A'^{(t)}_2 \cdot A_2^{(t)}, \tag{8}$$

$$A_2^{(t+1)} = (1-\gamma)A_1^{(t)} + \gamma A'^{(t)}_1 \cdot A_1^{(t)}, \tag{9}$$

where γ is a coefficient control the balance between the two structure matrices, which we can adjust with the valid data. We dot product the $A'^{(t)}$ part with $A^{(t)}$ to make sure it learns the edge values taking the current structure as the frame of reference. Please notice that the new structure matrices are formed based on the information from the peer-autoencoder. In the following epochs of training, the autoencoders keep using the updated graph structure A_1 and A_2 from the previous epoch as their input.

Algorithm 1 Cross-Graph Autoencoder

Require:
X: The feature matrix of the graph;
A: The adjacency matrix of the graph;
T: The number of iterations;
Ensure:
$Z1$ & $Z2$: the learnt embeddings of the autoencoders;

Construct the two autoencoders.
for $t = 1$ to T **do**
Calculate $A'^{(t)}_1$ and $A'^{(t)}_2$ according to Eq.(5), Eq.(6) and Eq.(7);
Construct $A^{(t+1)}_1$ and $A^{(t+1)}_2$ according to Eq.(8) and Eq.(9) for the next iteration to use.
end for
Calculate Z_1 & Z_2 according to Eq.(2) with $A^{(T)}_1$ and $A^{(T)}_2$ being the input A respectively.

C. Algorithm Description and Deeper Insights

The algorithm is summarized in Algorithm 1. In each iteration, we obtain the reconstruction from each autoencoder. The input structure compromise with it and an updated structure is thereby formed, which is further provided to the peer-autoencoder as the input structure of the next iteration. We use the hidden representation in the last iteration as our final embedding results. Fig. 3 gives an illustrated example on our procedure to update the graph structure to handle corrupted graphs.

Our Cross-Graph learning framework is simple but effective, which keeps up the performance of graph autoencoders against redundant edges. In the following, we will provide deep insights into our algorithm through answering two key questions.

Question 1: Why can manipulating the graph structure A towards the reconstructed structure A' help improve the robustness against structural corruption?

To answer question 1, we need to make it clear in advance what the elements in A and A' represent. A is the adjacency matrix of the graph, in which all elements are 0 or 1. $A_{jk} = 1$ shows there is an edge connecting node j and k in the graph, otherwise the two nodes are disconnected. On the other hand, A' is the reconstructed structure matrix, since it is constrained by the sigmoid function, its elements are all valued between 0 and 1. Therefore, A'_{jk} can be regarded as a predicted probability whether node j and k are connected with an edge from the autoencoder.

Furthermore, we do not fully trust the connections shown in the input graph structure. So, we take the reconstructed structure partly as a reference, to construct a new adjacency matrix for the training of the next iteration. The coefficient γ can be seen as the degree how the structure is trusted. So here comes a further question: Why the reconstructed structure can be trusted to judge if an edge is an outlier or not?

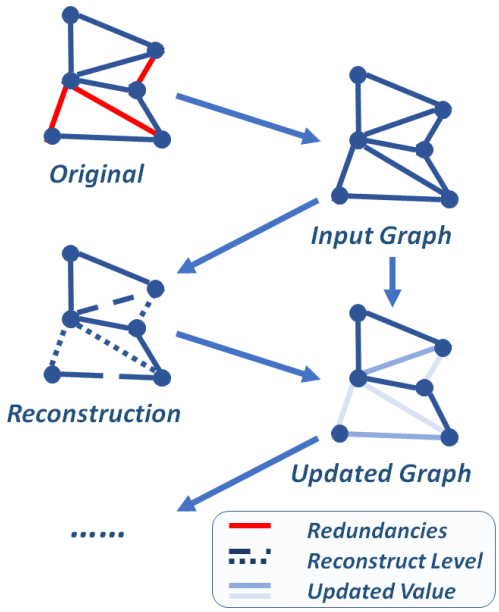


Fig. 3. The Cross-Graph Working Mechanism. The reconstruction level (the reconstruction's similarity to the input graph) could show its opinion of the possibility of each edge being redundant, represented by the sparsity degree of the dotted line. So, based on it and the input graph, our Cross-Graph constructs a new updated graph, which devalue those suspected edges. The edge value is positively correlated with the reconstruction level, represented by the gradation of the edge color. This updated graph is provided to the peer-autoencoder for the next iteration update.

When the optimization starts, the autoencoder tries to reconstruct the input structure. Deep Neural Networks (DNNs), including graph autoencoders, have a strong ability fit with noisy data. However, according to a recent discovery of the DNN memorization mechanism, the network will better fit those easy and clean instances first [16], [17]. Thus, when the reconstructed A' tries to fit the input A, those elements trying to fit the real 1s in A will converge to 1 fast, while those trying to fit the corruptions will converge relatively slower. Therefore, when A compromises with the reconstructed A', those suspect edges in A will be given a balanced value instead of 1 to represent the reliability of the edge and supervise the subsequent training.

Question 2: Why do we need two autoencoders and that they update their parameters learning each other?

As for question 2, maintaining two autoencoders seems unnecessary, since a single autoencoder can work according to the above analysis. Indeed, a single autoencoder already has the ability to distinguish and filter out corruptions following our process. However, the performance of this will be unstable, because the reconstruction in the first few iterations are very random, highly depending on the initial parameter state.

Therefore, we further use two autoencoders, to reduce the impact of individual bias. Both autoencoders have an independent ability to learn and filter out corruptions. We exchange their reconstruction results so one might get advice from the other. Thereby, different corruptions can be selected

out and our model is less possible to misjudge the corruptions because of a particular initial state.

a) Incorporate with Other Algorithms: It is worth mentioning that, our Cross-Graph learning framework can work on, not only the basic graph convolutional autoencoder, but also other graph embedding frameworks, as long as they optimize their embedding by minimizing the difference between the original graph structure and the algorithm constructed structure.

V. Experiments

Since we focus on unsupervised learning as explained before, we evaluate our graph embedding mainly with three unsupervised graph analytic tasks: node clustering, link prediction and network visualization. We conduct experiments for various algorithms on both corrupted and uncorrupted datasets to analyze the effect of our model. The implementation is provided for reproducibility[1].

A. Datasets

We use three graph datasets widely-used in attributed graph learning evaluation, summarized in Table I. Cora and Citeseer are citation networks where nodes are publications categorized by the research sub-fields and edges denote the citation relationships. Wiki is a web-page network containing web-page documents, and the edges represent there are web-links between the two pages.

To construct the corrupted version of these datasets, we randomly add a certain percentage of edges to the network structure. We demonstrate in the parameter study that, our Cross-Graph can strengthen the performance of the baselines under any ratio of corruption, and with more edges added, the promotion become more apparent. So, we simply add 40%, 50% and 20% redundancy edges to the Cora, Citeseer and Wiki datasets respectively as default corruption to make a clear and comprehensive report in our experiments. For example, for the Cora dataset, originally with $5,278$ edges, we add 40%, that is $2,111$ redundant edges to the graph structure to determine our corrupted Cora dataset used in the experiments. All the experiments with corrupted data use this corruption ratio unless otherwise stated.

B. Baselines

We mainly focus on comparing the performance of different graph autoencoders with their performance when incorporating our Cross-Graph framework. So, we choose two representative graph autoencoders as our main baselines as below:

- **GAE** [30] leverages both topological and content information from the graph data with a graph convolution operation and learns unsupervised graph embedding.
- **ARGA** [31] is a graph convolutional autoencoder-based method that manipulates the learnt embedding with an adversarial regularizer.

For the clustering task, we also include some representative clustering algorithms for comparison on the corrupted data:

[1]https://github.com/FakeTibbers/Cross-Graph.

Dataset	Nodes	Features	Clusters	Links
Cora	2,708	1,433	7	5,278
Citeseer	3,327	3,703	6	4,552
Wiki	2,405	4,973	17	11,596

TABLE I
Benchmark Graph Datasets

- **Spectral Clustering** uses the eigenvalues of the similarity matrix to perform dimensionality reduction before clustering.
- **DeepWalk** [18] performs random walk on the graph structure to learn representations.
- **DNGR** [28] uses stacked denoising autoencoder to encode each vertex into a low-dimensional representation.

C. Node Clustering on Corrupted Data

1) Evaluation Metrics: We follow Xia et al. [36], and employ six metrics to validate our clustering performance, namely accuracy (ACC), normalized mutual information (NMI), F-score, precision, recall and average rand index(ARI). A better clustering scheme should lead to higher scores for all these six metrics.

2) Experimental Setup: For the baseline algorithms, we keep the settings used in the original papers as far as possible and select parameters carefully following the procedures in the original papers. For example, for the GAE algorithm, we construct the encoders with a 32-neuron hidden layer and a 16-neuron embedding layer and train 200 epochs to get the result. On the other hand for ARGA, we use a 32-neuron embedding layer instead and train 100 epochs, in accordance with the original paper.

For our algorithm, we incorporate our Cross-Graph with two representative autoencoders for graph learning: Graph Convolutional Autoencoder (GAE) and Adversarially Regularized Graph Autoencoder (ARGA). For a fair comparison, the autoencoder part in our algorithm uses the exact same settings as the original method. For the Cross-Graph part, we set the coefficient γ to 0.02 in accordance with the parameter study result.

We learn the graph embedding for each method, and then perform k-means clustering on the embedding to get the results. We run the experiments under each setting 10 times to get an average score for report and comparison.

3) Experimental Results: The clustering results of various algorithms on the corrupted Cora, Citeseer and Wiki datasets are reported in Tables II, III and IV respectively.

It can be observed from the result that all the algorithms suffer from the corruption of the data. For instance, the DNGR algorithm completely loses its clustering ability on the corrupted Citeseer and Wiki datasets. Our Cross-Graph framework can improve significantly the performance of both GAE and ARGA when they are used to learn clustering from the corrupted datasets. Take the experiments on the corrupted Cora dataset as an example. Incorporating it with our Cross-Graph has increased the performance of GAE from 13.0% in terms of accuracy to 54.4% in terms of precision, and

TABLE II
CLUSTERING RESULTS ON CORRUPTED **CORA** DATASET

	ACC(↑)	NMI(↑)	F-score(↑)	Precision(↑)	Recall(↑)	ARI(↑)
Spectral Clustering	33.55%	15.81%	26.81%	26.89%	26.80%	10.86%
DeepWalk	41.55%	20.48%	30.56%	30.97%	30.28%	15.57%
DNGR	38.33%	17.47%	26.40%	28.19%	24.84%	11.56%
GAE	45.19%	23.58%	11.73%	12.34%	12.41%	18.13%
GAE+Cross-Graph	51.05%	30.07%	18.00%	19.05%	18.74%	24.40%
ARGA	59.53%	37.92%	58.73%	59.80%	61.39%	32.98%
ARGA+Cross-Graph	64.25%	40.28%	61.63%	62.13%	63.73%	38.64%

TABLE III
CLUSTERING RESULTS ON CORRUPTED **CITESEER** DATASET

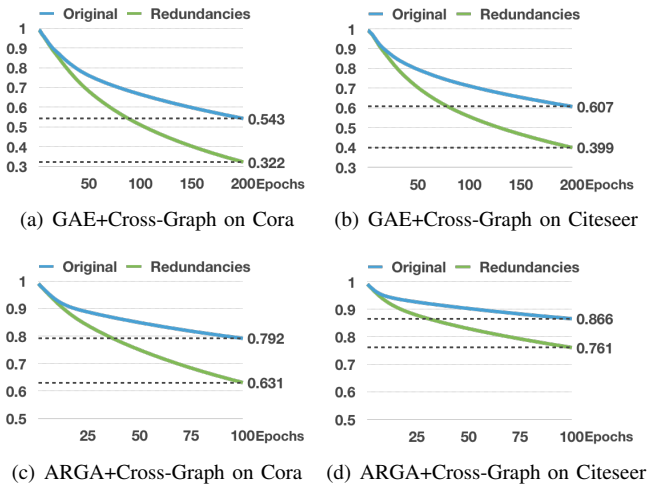
	ACC(↑)	NMI(↑)	F-score(↑)	Precision(↑)	Recall(↑)	ARI(↑)
Spectral Clustering	27.64%	5.48%	23.98%	20.90%	28.56%	4.33%
DeepWalk	31.26%	5.91%	23.20%	22.35%	24.19%	5.69%
DNGR	21.23%	0.31%	30.29%	17.86%	99.71%	0.01%
GAE	34.42%	9.91%	16.12%	16.79%	16.63%	8.24%
GAE+Cross-Graph	37.33%	12.35%	16.64%	17.23%	17.01%	10.29%
ARGA	39.39%	13.87%	38.27%	41.64%	38.24%	11.58%
ARGA+Cross-Graph	42.82%	15.05%	41.81%	44.57%	41.40%	13.04%

has also increased the performance of ARGA from 3.8% in terms of recall to 17.2% in terms of average rand index. Both comparisons demonstrate the ability of our Cross-Graph to increase robustness against edge corruption.

Detailed Analysis. We further monitor the value of both the redundant and original edges respectively in the Cora and Citeseer datasets along with the optimization of both GAE and ARGA. The change of averaged values is presented in Fig. 4.

The results show a clear trend. Both the original and redundancy edges are valued universally as 1 at the beginning of the training, because they have no difference in the input graph. However, our Cross-Graph process has a strong ability to filter out the redundancy edges, based on the mechanism explained in Section 4. After 200 epochs of training for the GAE-based algorithm and 100 epochs for the ARGA-based counterpart (the default epochs for these algorithms), the original and redundancy edges show a clear gap. Taking the GAE-based Cross-Graph as an example, the original edges keep a relatively high value, the average is 0.543 on the Cora dataset and 0.607 on the Citeseer dataset; while the average value of redundancy edges has decreased on these datasets to 0.322 and 0.399 respectively, much closer to 0, which means there is no edge in between. The difference between ARGA and ARGA-based Cross-Graph is not as high as the GAE-based one. This perhaps, mainly because (1) it has fewer training epochs by default; and (2) the adversarial regularizer brings ARGA a stronger ability in fitting those abnormal edges. This fact also results in an insignificant performance improvement on ARGA, compared to the improvement on GAE by our Cross-Graph.

Parameter Study. We vary the corruption level of the



(a) GAE+Cross-Graph on Cora (b) GAE+Cross-Graph on Citeseer

(c) ARGA+Cross-Graph on Cora (d) ARGA+Cross-Graph on Citeseer

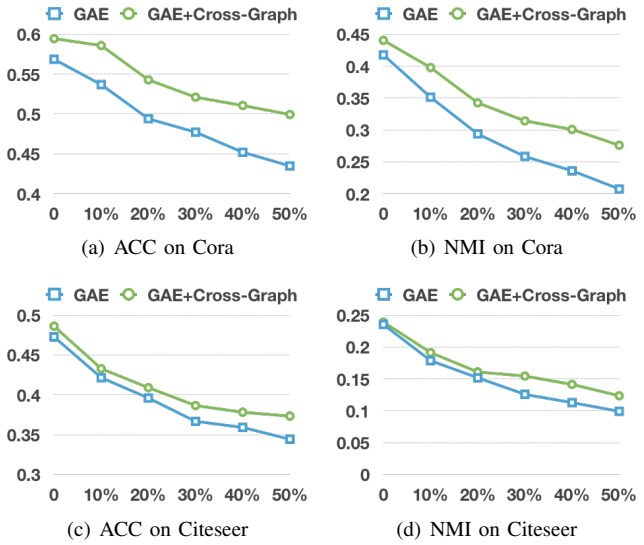
Fig. 4. The devalue of two types of edges (original edges and redundancy edges we added) along with the Cross-Graph training process.

datasets by continuously adding random edges to the original graph structure, until the total number of added edges reaches 50% of the original number. We report the clustering results in Fig. 5.

It can be observed that, our Cross-Graph can continuously help improve the performance of the original algorithm, no matter how many edges we add to the graph. Even on the original graph without any redundancy edges, GAE + Cross-Graph also slightly outperforms GAE in the clustering task. Generally, the more serious the corruption is, the more effective our Cross-Graph is. This is because our Cross-Graph is designed for robust learning and can retain performance

TABLE IV
CLUSTERING RESULTS ON CORRUPTED **WIKI** DATASET

	ACC(↑)	NMI(↑)	F-score(↑)	Precision(↑)	Recall(↑)	ARI(↑)
Spectral Clustering	30.06%	23.01%	19.06%	19.71%	18.55%	10.56%
DeepWalk	36.70%	28.77%	24.41%	25.71%	23.31%	16.64%
DNGR	16.76%	1.46%	17.92%	9.85%	98.99%	0.07%
GAE	19.16%	13.09%	2.96%	3.99%	4.37%	4.55%
GAE+Cross-Graph	22.37%	18.97%	3.23%	5.83%	4.49%	5.96%
ARGA	40.45%	41.40%	37.86%	45.62%	39.66%	21.32%
ARGA+Cross-Graph	43.45%	42.97%	39.46%	47.97%	43.05%	22.35%



(a) ACC on Cora (b) NMI on Cora

(c) ACC on Citeseer (d) NMI on Citeseer

Fig. 5. The clustering performance under different percentage of redundancy edge corruption.

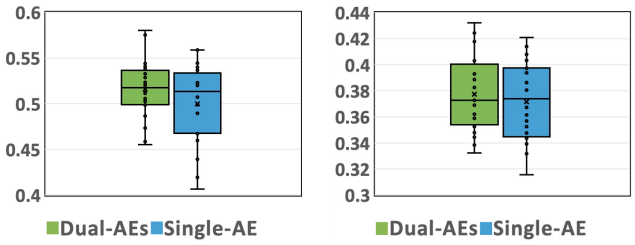
(a) ACC on Cora (b) ACC on Citeseer

Fig. 6. The clustering performance with different Cross-Graph coefficient γ.

under extremely serious corruption.

We also vary the coefficient γ in our Cross-Graph framework. This controls the balance between the input and reconstructed structure in the updated graph. The result is reported in Fig. 6.

The result from both datasets reveals a similar trend: the performance of the learnt embedding is well when the coefficient γ is small, and the performance reaches its peak when γ is around 0.02. As we increase the value of γ from 0.02, the performance fluctuates and deteriorates overall. This is because (1) the change of the input graph structure is too rapid for the neural network to become stabilized; (2) the reconstructed structure is trusted too much and the graph modification is overdone.



(a) Cora (b) Citeseer

Fig. 7. Box plots of the 20 times' clustering accuracy from the Cross-Graph Dual-autoencoders interactive process, compared with a single autoencoder framework. For each box, the five lines from top to bottom represents the maximum, the first quartiles, the sample median, the third quartiles and the minimum of the clustering results.

Effectiveness of the Cross-Graph Dual-autoencoders interactive process.

As discussed above, a single autoencoder can already work to filter out the corruptions. We use two autoencoders to make the framework more stable. To show the effectiveness of this interactive process, we compare our Cross-Graph with a framework which maintains only one autoencoder. The input graph for this autoencoder is updated according to its own reconstruction result. We keep all the other settings the same, and run each framework for 20 times. The performance evaluated by the clustering accuracy is reported in Fig. 7.

We can observe from the box plots that, though the average performance of the single-autoencoder framework is nearly on par with our Cross-Graph, its 20 results are more dispersive. For example for the Cora dataset, all 20 results from our Cross-Graph lie above 0.45, and half of them are concentrated between 0.50 and 0.54. On the other hand, the results from the single-autoencoder framework vary from 0.40 to 0.57. This show our Cross-Graph interactive process can reduce the impact of individual bias and stabilize the framework.

D. Link Prediction on Corrupted Data

a) Evaluation Metrics: We report the results of AUC score and AP score (average precision) follow Pan et al. [31]. A better link prediction result should lead to a higher score for both metrics.

TABLE V
LINK PREDICTION RESULTS ON CORRUPTED DATASETS

	Cora		Citeseer		Wiki	
	AUC(↑)	AP(↑)	AUC(↑)	AP(↑)	AUC(↑)	AP(↑)
GAE	84.03%	86.17%	80.35%	82.88%	75.05%	76.56%
GAE+Cross-Graph	85.72%	87.63%	81.83%	84.07%	76.87%	78.55%
ARGA	86.84%	88.70%	87.29%	89.49%	91.53%	92.71%
ARGA+Cross-Graph	87.74%	89.81%	88.38%	90.49%	92.28%	93.37%

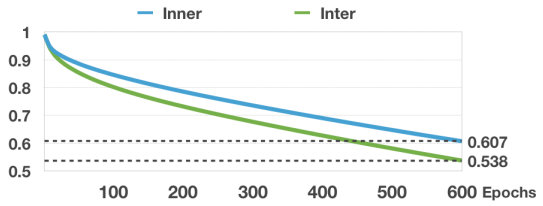


Fig. 8. The devalue of inner-edges and inter-edges comparison along with the Cross-Graph training process on uncorrupted Cora dataset.

b) Experimental Setup: For the link prediction task, we mask 5% edges for hyperparameter optimization and 10% edges for performance test. We mask the valid and test sets before we corrupt the dataset, to make sure our test sets are clean, as we do not see it as a plus in predicting those redundancy edges. For the baselines, we retain the parameter settings described in the original papers.

c) Experimental Results: The detailed results are shown in Table V. We can observe that our Cross-Graph also helps retain the link prediction performance for the based algorithms under structural corruption. For example, the ARGA algorithm originally can achieve both AUC score and AP score as high as 92% on the Cora and Citeseer dataset. Both scores drop to about $86\% - 89\%$ under the corruption. By incorporating our Cross-Graph, we can recover the performance for about 1%. The effect of Cross-Graph is not as strong as in the clustering task, probably because the structural corruption is much more severe for the link prediction task.

E. Experiments on Uncorrupted Data

An interesting observation of our framework is that it can also outperform its base method on uncorrupted dataset. To deeper analyze the principle of our Cross-Graph, we also implement our algorithm incorporating GAE on the original "clean" $Cora$ dataset. The result is summarized in Table VI.

From the node clustering and link prediction task performance, we can see GAE + Cross-Graph can also slightly outperform GAE on the uncorrupted dataset and at least match the performance of GAE along all the evaluation metrics.

Since our Cross-Graph's main ability is to revalue the edges, we also try to monitor two kinds of edges in this "clean" dataset: the edges linking nodes from the same cluster as inner edges, and the edges that link nodes from two different clusters as inter edges. Their average value changes, along with the training process, are simulated in Fig. 8.

Fig. 8 shows that inter-edges connecting nodes from two different clusters devalue much faster than the inner-edges. This is because each cluster of nodes are closely linked with inner-edges. Inter-edges connecting two nodes from different clusters are rare and distant from the clusters. Not surprisingly, these inter-edges are also called weak links in the network and are hard to reconstruct. This will lead our Cross-Graph to mark them as highly abnormal and devalue these inter-edges faster than the inner-edges which, in return, strengthen the structure of each cluster and improve the clustering performance.

F. Network Visualization on Corrupted Data

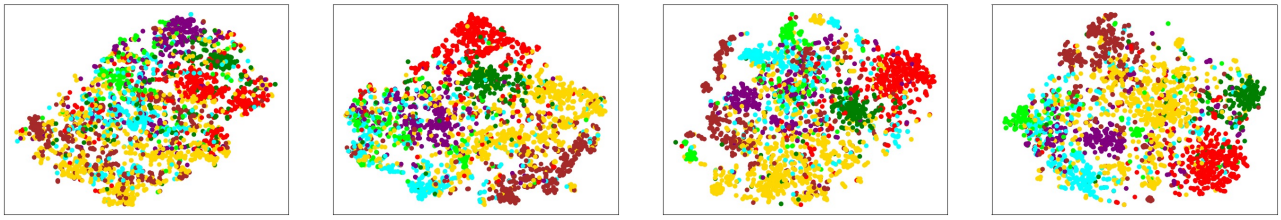
We visualize the corrupted Cora dataset in a two-dimension space by applying the t-SNE algorithm [37] on the embedding learned from different algorithms. The visualizations in Fig. 9 show that, due to the structural corruption of the Cora dataset, GAE already can hardly maintain a meaningful embedding, and ARGA's embedding performance is also severely ruined. Fortunately, by applying Cross-Graph to the original graph autoencoders, the embedding could be effectively recovered, allowing us to obtain a more meaningful visualization layout of the graph data.

VI. CONCLUSION

In this paper, we discuss a problem neglected by predecessors, the corruption in the graph structure. We argue that structural corruption, especially with redundant edges, is commonplace in real-world attributed networks, and can easily ruin the performance of previous graph embedding algorithms. Going a step further, we propose a novel autoencoder-based framework to learn robust attributed graph embedding against redundant edges. Our model is purely unsupervised and can learn efficient graph embedding without access to either clean graph structure or label guidance. We maintain two autoencoders which can learn from each other and frequently filter out those unreliable edges. Experiment results demonstrated that our algorithms can lift up significantly the performance of previous methods against structural corruptions.

ACKNOWLEDGEMENT

BH was supported by the HKRGC Early Career Scheme No. 22200720, NSFC Young Scientists Project No. 62006202, HKBU Tier-1 Start-up Grant and HKBU CSD Start-up Grant.



(a) Embedding from GAE (b) Embedding from GAE+Cross-Graph (c) Embedding from ARGA (d) Embedding from ARGA+Cross-Graph

Fig. 9. Visualizations of the corrupted Cora dataset, based on the embedding learned from various algorithms. The dots represent nodes and the seven different colors represent the ground-truth clusters the nodes belongs to.

TABLE VI
CLUSTERING & LINK PREDICTION RESULTS ON UNCORRUPTED **CORA** DATASET

	ACC(↑)	NMI(↑)	F-score(↑)	Precision(↑)	Recall(↑)	ARI(↑)	AUC(↑)	AP(↑)
GAE	56.85%	41.76%	13.48%	14.55%	14.23%	31.71%	90.93%	92.00%
GAE+Cross-Graph	59.45%	44.04%	13.69%	14.69%	14.64%	33.61%	91.39%	92.43%

REFERENCES

[1] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *TNNLS*, 2020.

[2] S. Ji, S. Pan, E. Cambria, P. Marttinen, and P. S. Yu, "A survey on knowledge graphs: Representation, acquisition and applications," *arXiv preprint arXiv:2002.00388*, 2020.

[3] H. Cai, V. W. Zheng, and K. C.-C. Chang, "A comprehensive survey of graph embedding: Problems, techniques, and applications," *TKDE*, vol. 30, no. 9, pp. 1616–1637, 2018.

[4] D. Zhang, J. Yin, X. Zhu, and C. Zhang, "Network representation learning: A survey," *IEEE transactions on Big Data*, 2018.

[5] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *ICLR*, 2017.

[6] C. Wang, S. Pan, G. Long, X. Zhu, and J. Jiang, "Mgae: Marginalized graph autoencoder for graph clustering," in *Proc. of CIKM*. ACM, 2017, pp. 889–898.

[7] C. Wang, S. Pan, R. Hu, G. Long, J. Jiang, and C. Zhang, "Attributed graph clustering: a deep attentional embedding approach," in *IJCAI*. AAAI Press, 2019, pp. 3670–3676.

[8] Z. Wang, C. Chen, and W. Li, "Predictive network representation learning for link prediction," in *SIGIR*, 2017, pp. 969–972.

[9] J. Li, H. Dani, X. Hu, and H. Liu, "Radar: Residual analysis for anomaly detection in attributed networks." in *IJCAI*, 2017, pp. 2152–2158.

[10] N. Liu, X. Huang, and X. Hu, "Accelerated local anomaly detection via resolving attributed networks." in *IJCAI*, 2017, pp. 2337–2343.

[11] M. Al Hasan and M. J. Zaki, "A survey of link prediction in social networks," in *Social network data analytics*. Springer, 2011, pp. 243–275.

[12] H. Zhang, S. Kiranyaz, and M. Gabbouj, "Outlier edge detection using random graph generation models and applications," *JBD*, vol. 4, no. 1, p. 11, 2017.

[13] D. Zügner, A. Akbarnejad, and S. Günnemann, "Adversarial attacks on neural networks for graph data," in *KDD*, 2018, pp. 2847–2856.

[14] D. Zhu, Z. Zhang, P. Cui, and W. Zhu, "Robust graph convolutional networks against adversarial attacks," in *KDD*, 2019, pp. 1399–1407.

[15] A. Blum and T. Mitchell, "Combining labeled and unlabeled data with co-training," in *COLT*. ACM, 1998, pp. 92–100.

[16] D. Arpit, S. Jastrzebski, N. Ballas, D. Krueger, E. Bengio, M. S. Kanwal, T. Maharaj, A. Fischer, A. Courville, Y. Bengio *et al.*, "A closer look at memorization in deep networks," in *ICML*. JMLR. org, 2017, pp. 233–242.

[17] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning requires rethinking generalization," *arXiv:1611.03530*, 2016.

[18] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *KDD*, 2014.

[19] X. Wang, P. Cui, J. Wang, J. Pei, W. Zhu, and S. Yang, "Community preserving network embedding," in *AAAI*, 2017.

[20] M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu, "Asymmetric transitivity preserving graph embedding," in *KDD*, 2016, pp. 1105–1114.

[21] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *WWW*, 2015, pp. 1067–1077.

[22] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *KDD*, 2016, pp. 855–864.

[23] C. Yang, Z. Liu, D. Zhao, M. Sun, and E. Chang, "Network representation learning with rich text information," in *IJCAI*, 2015.

[24] S. Zhu, L. Zhou, S. Pan, C. Zhou, G. Yan, and B. Wang, "Gssnn: Graph smoothing splines neural networks." in *AAAI*, 2020, pp. 7007–7014.

[25] G. Wan, S. Pan, C. Gong, C. Zhou, and G. Haffari, "Reasoning like human: Hierarchical reinforcement learning for knowledge graph reasoning," in *IJCAI*, 2020, pp. 1926–1932.

[26] S. Zhu, C. Zhou, S. Pan, X. Zhu, and B. Wang, "Relation structure-aware heterogeneous graph neural network," in *ICDM*. IEEE, 2019, pp. 1534–1539.

[27] Z. Wu, S. Pan, G. Long, J. Jiang, X. Chang, and C. Zhang, "Connecting the dots: Multivariate time series forecasting with graph neural networks," in *KDD*, 2020.

[28] S. Cao, W. Lu, and Q. Xu, "Deep neural networks for learning graph representations," 2016.

[29] F. Tian, B. Gao, Q. Cui, E. Chen, and T.-Y. Liu, "Learning deep representations for graph clustering," in *AAAI*, 2014.

[30] T. N. Kipf and M. Welling, "Variational graph auto-encoders," *NIPS Workshop*, 2016.

[31] S. Pan, R. Hu, G. Long, J. Jiang, L. Yao, and C. Zhang, "Adversarially regularized graph autoencoder for graph embedding." in *IJCAI*, 2018, pp. 2609–2615.

[32] L. Jiang, Z. Zhou, T. Leung, L.-J. Li, and L. Fei-Fei, "Mentornet: Learning data-driven curriculum for very deep neural networks on corrupted labels," in *ICML*, 2018.

[33] E. Malach and S. Shalev-Shwartz, "Decoupling" when to update" from" how to update"," in *NeurIPS*, 2017, pp. 960–970.

[34] B. Han, Q. Yao, X. Yu, G. Niu, M. Xu, W. Hu, I. Tsang, and M. Sugiyama, "Co-teaching: Robust training of deep neural networks with extremely noisy labels," in *NeurIPS*, 2018, pp. 8527–8537.

[35] M. Wu, S. Pan, C. Zhou, X. Chang, and X. Zhu, "Unsupervised domain adaptive graph convolutional networks," in *Proceedings of The Web Conference*, 2020, pp. 1457–1467.

[36] R. Xia, Y. Pan, L. Du, and J. Yin, "Robust multi-view spectral clustering via low-rank and sparse decomposition," in *AAAI*, 2014.

[37] L. Van Der Maaten, "Accelerating t-sne using tree-based algorithms," *JMLR*, vol. 15, no. 1, pp. 3221–3245, 2014.