SAGES: Scalable Attributed Graph Embedding With Sampling for Unsupervised Learning

Jialin Wang, Xiaoru Qu, Jinze Bai, Zhao Li, Ji Zhang, *Senior Member, IEEE*, and Jun Gao

Abstract—Unsupervised graph embedding method generates node embeddings to preserve structural and content features in a graph without human labeling burden. However, most unsupervised graph representation learning methods suffer issues like poor scalability or limited utilization of content/structural relationships, especially on attributed graphs. In this paper, we propose SAGES, a graph sampling based autoencoder framework, which can promote both the performance and scalability of unsupervised learning on attributed graphs. Specifically, we propose a graph sampler that considers both the node connections and node attributes, thus nodes having a high influence on each other will be sampled in the same subgraph. After that, an unbiased Graph Autoencoder (GAE) with structure-level, content-level, and community-level reconstruction loss is built on the properly-sampled subgraphs in each epoch. The time and space complexity analysis is carried out to show the scalability of SAGES. We conducted experiments on three medium-size attributed graphs and three large attributed graphs. Experimental results illustrate that SAGES achieves the competitive performance in unsupervised attributed graph learning on a variety of node classification benchmarks and node clustering benchmarks.

Index Terms—Machine learning, unsupervised graph learning, graph neural network

1 Introduction

Graphs are powerful to model complex relationships in different real-life applications, such as citation networks, economic graphs, and social networks. Among the various graphs, the attributed graph has attracted much attention in recent years [1], [2]. As shown in Fig. 1, unlike the plain graphs where only the topological structure is available, nodes in the attributed graph have rich features and attributes associated with them. For example, nodes (articles) in an academic citation graph have substantial text information about the article's topic, and the nodes (users) in a social network post their profiles as the attribute. These informative attributes can benefit graph analysis [3]. Therefore, the utilization of node attributes information is important to study the attributed graph.

The representation learning for attributed graphs, which generates low-dimensional embeddings of nodes to preserve graph topology structure and attributes, is shown to be effective for various graph-based tasks [1], [4]. According to the need for data labeling, these methods are roughly categorized into supervised, semi-supervised [5] and unsupervised learning [6]. This paper focuses on unsupervised learning. Compared with supervised learning, unsupervised learning does not incur a substantial labeling burden. Besides, the node embeddings learned by unsupervised ways can be reused for different downstream applications, such as node classification and node clustering.

The existing graph representation learning can be roughly divided into three categories: factorization based [7], random walk based [8], and graph autoencoder (GAE) based models [6]. The first two models (factorization based and random walk based) are limited by their shallow architecture, while GAE based models can capture both non-linear graph structures and content by using deeper architectures in neural networks. Therefore, a large amount of GAE based models, such as AGE [9], GALA [10], and DGI [11], are proposed to strengthen the representation power of graph embedding models. Although these GAE based models show their power in various tasks (link prediction, node clustering, etc.), they are hard to scale because of the full-batch training. To make the GAE scalable and applicable to large graphs, some sampling-based approaches are developed. GraphSAGE [12] adopts uniform sampling to obtain a fixed size neighborhood in the encoder, and reconstructs the structural information by negative sampling in the decoder. A scalable GAE is proposed in [13], which trains GAE on the smaller k-core version of the graph, and then propagates node representations to other nodes via a fast heuristics. FastGAE [14] performs importance sampling in encoder like FastGCN [15], and reconstruct structural information of the sampled subgraph in decoder. Nevertheless, these scalable GAE based methods have three major drawbacks on large attributed graph embeddings.

First, the graph structure and node attribute information are not well utilized by the sampling strategy. The existing

- *Jialin Wang, Xiaoru Qu, Jinze Bai, and Jun Gao are with the Key Laboratory of High Confidence Software Technologies, Ministry of Education & School of EECS, Peking University, Beijing 100871, China. E-mail: {wangjialin, quxiaoru, baijinze, gaojun}@pku.edu.cn.*
- *Zhao Li is with Alibaba Group, Hangzhou, Zhejiang 310027, China, and also with the Zhejiang University, Hangzhou, Zhejiang 310027, China. E-mail: zhao_li@zju.edu.cn.*
- *Ji Zhang is with the University of Southern Queensland, Toowoomba 4350, Australia, and also with Zhejiang Lab, Hangzhou, Zhejiang 310027, China. E-mail: zhangji77@gmail.com.*

Manuscript received 3 September 2020; revised 4 December 2021; accepted 16 January 2022. Date of publication 4 February 2022; date of current version 3 April 2023.
This work was supported in part by NSFC under Grants 61832001 and 62172372, and in part by Zhejiang Provincial Natural Science Foundation under Grant LZ21F030001.
(Corresponding authors: Zhao Li and Jun Gao.)
Recommended for acceptance by A. Khan.
Digital Object Identifier no. 10.1109/TKDE.2022.3148272

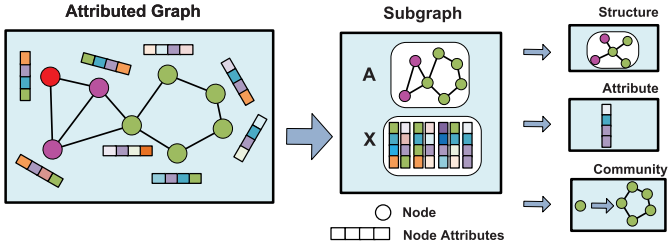

Fig. 1. An illustration of the attributed graph. Nodes with different colors have different labels. $\mathbf{A}$ and $\mathbf{X}$ represent the adjacency matrix and the node attribute matrix of the graph, respectively. A good unsupervised attributed graph embedding method should consider the graph structure, node attribute, and community information.

scalable GAE methods either simply sample nodes uniformly across layers [12] or sample subgraph based on graph topology (k-core number or degree of nodes) [13], [14], but ignore node attributes. However, the utilization of node attributes in sampling proved to be essential [16].

Second, different sampling strategies result in non-identical node and edge sampling probability, and introduce bias in the forward propagation in GAE each minibatch. This bias should be considered by models.

Third, the most existing GAE based methods only consider simple local structural information in the decoder. In the experiment, we find that the performance of GAE, which only reconstruct adjacency information, is seriously degraded after subgraph sampling. There is no such problem in supervised scalable GNNs, such as Cluster-GCN [4], because they have a clear label after subgraph sampling. Intuitively, structural information may be disturbed after subgraph sampling while node attribute information is still intact. Besides, the incorporation of community structure in network embedding can make the node representations more discriminative [2]. Therefore, we argue that both the node attributes and community relationships are crucial on the attributed graph.

In this paper, aiming at the following question: *Can we design an efficient and effective graph embedding framework for attributed graphs, such that all of the above three challenges can be tackled?* We propose our framework Scalable Attributed Graph Embedding with Sampling (SAGES) to efficiently train GAE over large attributed graphs. To make the model scalable, SAGES first samples the training subgraphs, and then builds a full GAE on each subgraph. We apply the following three measures to overcome the issues of scalable GAE methods: i) To better utilize graph structure and node attributes during sampling, SAGES utilizes a light-weight subgraph-sampling algorithm, which considers both the node attributes and graph topological information. Thus, nodes with greater influence on each other have a higher probability of forming subgraphs. ii) To eliminate biases caused by the sampler, we propose normalization techniques so that the feature learning does not give preference to nodes more frequently sampled. iii) To overcome the difficulty in yielding high-quality node embeddings after sampling, we focus on the graph structure, node content, and community information in the decoder. Note that we naturally have communities based on our graph sampler, which offers an obvious advantage for SAGES to capture community-level information without extra cost.

Our contributions can be summarized as follows:

- We develop SAGES, an unsupervised graph learning framework, to learn node representations over large attributed graphs. SAGES samples subgraphs according to the similarity about the node attributes and graph structure. Then SAGES applies the unbiased graph autoencoder models into the subgraphs with highly correlated nodes. The time and space complexity analysis illustrate the scalability of SAGES.
- To learn meaningful node representations capturing local/global information of the attributed graph, SAGES introduces the reconstruction loss of node attributes and the mutual information between node and community as another two important guides in the decoder, as well as the reconstruction loss of graph structure.
- We conduct the experimental studies on three medium-size attributed graphs and three large attributed graphs under both transductive and inductive learning setups. The results show that SAGES is competitive on both node classification and link prediction tasks. Specially, we observe improvements of 2.5%, 1.2%, and 7.5% respectively compared with STOA unsupervised method DGI [11] on three large graphs in node classification tasks. Our code is available on https://github.com/SAGESAlgorithm/SAGES.

2 Related Work

2.1 Attributed Graph Embedding

Nodes in a graph are often affiliated with various content, such as abstract text in the citation graph, user attribute in the social network, and item description information in the e-commerce network. Such graphs are called attributed graphs, and the rich attributed information of nodes can benefit graph analysis. For example, Marsden [17] has proved that node attributes can reflect and affect the community structures in social science. Thus utilizing node attributes information is critical to study attributed graphs.

Most existing graph embedding methods mainly focus on graph topological information. For example, Deepwalk [8], and node2vec [18] only leverage graph structure and learn node representation by extracting different patterns through different random walk strategies. M-NMF [2] generates node representation through modularized nonnegative matrix factorization. PME [19] learn both first-order and second-order proximities in heterogenous graph. However, the above methods ignore the node attributes. To learn node embeddings which can preserve various proximities in both node attributes and topological structure, a wide variety of attributed graph embedding methods are proposed. TADW [20] regards DeepWalk as an inductive matrix factorization method and adds node attributes (text) for representation learning. LANE [1] leverages spectral techniques to project the adjacency matrix, node feature matrix, and node label matrix into a common vector space. Planetoid [21] trains node embedding by jointly predicting the class label and the neighborhood context in the graph.

However, most attributed graph embedding methods are either non-deep learning methods (matrix factorization and

random walks) or shallow models, which are difficult to use deep architectures of neural networks to capture the highly non-linear and high-order property of graphs.

2.2 Supervised Graph Neural Network

Graph neural networks (GNNs) [22] are deep neural networks that capture the dependence of graphs via message passing between the nodes of graphs, and show their power in graph representation learning. Early GNN focused on model structure rather than scalability. Graph Convolutional Networks (GCN) [5], which extends convolution operation to the graph domain, is a powerful model for attributed graph representation learning. Graph Attention Networks (GAT) [23] introduces an attention mechanism to better capture neighbor features by dynamically adjusting edge weights. Despite effective performance achieved, GCN and GAT suffer poor scalability because they train in a full batch manner.

Various methods are proposed to scale GNNs. They can be roughly separated into layer sampling and graph sampling methods.

Layer sampling methods first construct a complete GCN and then sample nodes or edges in each layer to form minibatchs. GraphSAGE [12] reduces receptive field size through uniform node sampling. PinSage [24] develops efficient random walks to sample neighborhoods. GraphCSC [25] utilizes centrality-biased random walks and centrality-based negative sampling approaches to scale algorithm. S-GCN [26] further restricts the receptive field by utilizing the historical activations in the previous layer to avoid redundant re-evaluation. AS-GCN [27] leverages an adaptive sampling strategy to restrict the neighbor expansion factor. FastGCN [15] applies importance sampling to reduce variance and ensures constant sample size for each layer.

Graph sampling methods first sample subgraphs each minibatch and then train a complete GCN on each subgraph. LouvainNE [28] learns node embeddings by a hierarchical clustering approach. COSINE [29] utilizes graph partitioning methods to graph and builds parameter sharing dependency of nodes based on the result of partitioning. Cluster-GCN [4] first partitions the training graph into densely connected clusters, samples subgraphs by randomly combining clusters, and then performs GCN on each sampled subgraph. GraphSAINT [30] samples subgraph based on the random walk, and proposes a normalization technique to eliminate the bias of graph sampling method. Graph sampling methods are more flexible and efficient especially when we use deep GNNs models [4].

2.3 Unsupervised Graph Neural Networks

Unsupervised graph neural networks aim to generate feature-preserving node representations without explicit user labeled data. Unlike some structure-based graph embedding methods, unsupervised GNNs exploit both topological information and node features simultaneously through deep neural networks. Most of these methods follow the standard autoencoder framework [31] and involve the combination of two stacked models. First, they use the encoder based on GNNs to map the node features into the latent embedding space, and then the decoder tries to reconstruct the information of original graph from the nodes latent embeddings. For example, GAE and VGAE [6] use Graph Convolutional Networks (GCN) as the encoder and simply reconstruct the adjacency matrix. ARGA [32] enforces node embedding to match a prior distribution via an adversarial training scheme. DGI [11] maximizes mutual information between node and graph summary representations in the decoder. GALA [10] proposes Laplacian sharpening as a decoder to prevent over-smoothing when reconstructing node features. MVGRL [33] learns graph representation by contrasting structural views of graphs. AGE [9] applies a Laplacian smoothing filter as the encoder and utilizes adaptive learning to train embeddings. These unsupervised GNNs have been widely adopted to tackle challenging problems in the graph, such as node classification [3] node clustering [32], [34], link prediction [6], and graph generation [35].

Although the above GAE based methods perform well in medium-size graph datasets, they are hard to use in large graph because of the scalability problems. Some scalable GAE frameworks [13], [14] are proposed to solve this issue, while they focus more on graph structure but neglect the utilization of attribute in sampling and training stage. In addition, most of the existing GAE based methods guide the training using the reconstruction loss for the local relationships, while we argue that community relationships play key roles especially when there is no explicit label signal.

TABLE 1
Notations Frequently Used in the Paper

Symbol	Definition
$\mathcal{G}$	Undirected attributed graph
N, E, L	The number of nodes, edges and layers in the graph
$\mathcal{V}, \mathcal{E}$	Vetrex and Edge set of the graph $\mathcal{G}$
F	The dimension of node attributes
F'	The dimension of hidden node embeddings
$f^{(\ell)}$	The number of node embedding dimensions in the l-th encoder/decoder layer
$\mathbf{A}$	Adjacency matrix of the graph $\mathcal{G}$
$\mathbf{X}$	Node attributed matrix of the graph $\mathcal{G}$
$\mathbf{Y}$	Node labels of the graph $\mathcal{G}$
$\mathbf{Z}$	Node representation matrix learned by GAE
$\mathbf{H}^{(l)}$	Node representation matrix of the l-th encoder layer
$\hat{\mathbf{H}}^{(1)}$	Node representation matrix of the l-th decoder layer
$\mathbf{x}_i$	The node representation of node i in $\mathbf{X}$
$\mathcal{N}_i$	The neighborhood of node i

3 Problem Statement

3.1 Notation

We use bold upper-case letters to denote matrices, bold lower-case letters to represent vectors, and non-bold italicized letters to denote scalars. Moreover, we use $\mathbf{A}^T$ to denote the transpose of a matrix $\mathbf{A}$. $\mathbf{A}_{i,j}$ denotes the entry of matrix $\mathbf{A}$ at the $i-$th row and the $j-$th column. Table 1 summarizes the frequently used notations in this paper.

3.2 Problem Definition

Suppose an unsupervised attributed graph representation learning setup: An attributed graph is represented as $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$, where $\mathcal{V} = \{v_i\}$ contains N nodes, and $\mathcal{E} = \{e_{ij}\}$ consists of a set of edges. The structural information between nodes can be represented by an adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$.

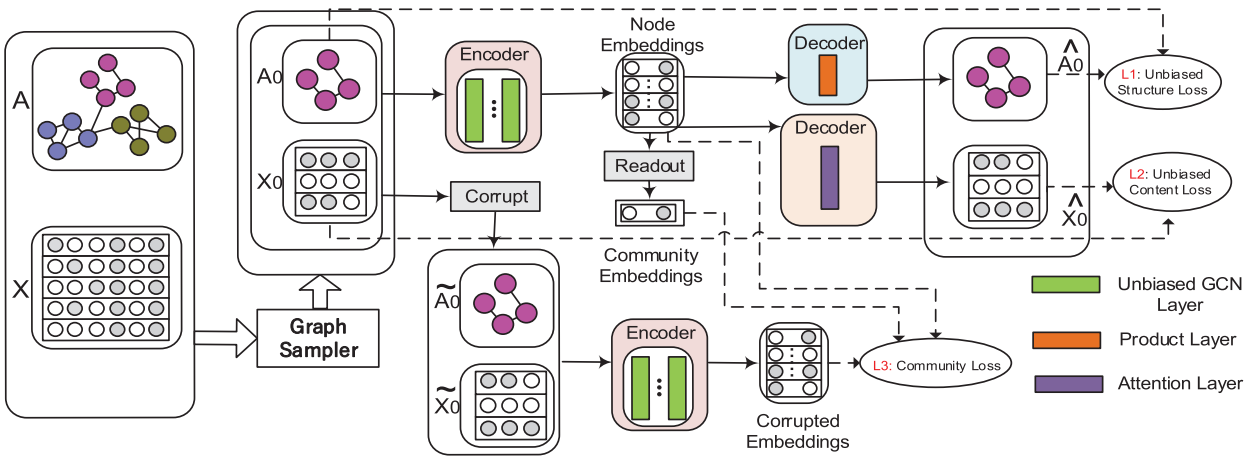

Fig. 2. The architecture of SAGES. Given a graph, we sample subgraphs by graph sampler and train an unbiased graph autoencoder, which contains structure, content and community loss in the decoder, on each minibatch.

We assume the graph to be unweighted in this paper, which means $\mathbf{A}_{i,j} = 1$ if $(v_i, v_j) \in E$; otherwise $\mathbf{A}_{i,j} = 0$. $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$, and $\mathbf{x}_i \in \mathbb{R}^F$ represents the features of node i. Our objective is to learn node representations $\mathbf{Z} = \{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_N\}$, where $\mathbf{z}_i \in \mathbb{R}^{F'}$ denotes the latent representation of node i. These representations can be used for downstream tasks.

4 Our Approach

In order to improve the scalability of model and ensure the quality of generated node embeddings, we present SAGES framework. In this section, we first show the overall architecture of SAGES framework. Then each part of SAGES is described in detail. Finally, the time complexity and space complexity of SAGES algorithm are analyzed.

4.1 Overall Framework

The overall training algorithm of SAGES is shown in Algorithm 1. The core idea behind SAGES is to scale graph autoencoder to handle large attributed graphs by exploiting sampling subgraphs. Roughly, SAGES first extracts appropriately connected subgraphs by graph sampler considering the information of node connections and attributes. Then SAGES employs an attribute sensitive graph autoencoder on each sampled subgraphs. GAE here can capture local/community relationships (both content and structure) between nodes. We use a normalization technique to eliminate bias caused by graph sampling in SAGES.

The overview of the GAE in SAGES is illustrated in Fig. 2. The encoder of SAGES is flexible and can be many GNN architecture variants, such as GCN [5], GAT [23], and JK-Networks [36]. The decoder of SAGES consists of three loss functions: structure loss, content loss, and community loss. Structure loss relies on a simple inner product decoder to reconstruct edges the subgraph. Content loss reconstructs node features of the subgraph. Community-level loss is based on local-community mutual information maximization to capture community information of the subgraph. Through the training process in Algorithm 1, SAGES can combine information of extracted subgraphs together so that the training process overall learns a good representation of the full graph.

Algorithm 1. SAGES Training Algorithm

Input: Graph $\mathcal{G}(\mathcal{V}, \mathcal{E}) = (\mathbf{A}, \mathbf{X})$; Graph Sampler SAMPLE.
Output: The GAE model with trained weights;
$\mathcal{G} \leftarrow$ construct training graph of $\mathcal{G}$. ▷ Inductive setting
Preprocessing: Caculating influence matrix $\tilde{\mathbf{X}}$;
Setup the SAMPLE parameters;
Compute normalization coefficients α, λ.
for each minibatch **do**
  $\mathcal{G}_t(\mathcal{V}_t, \mathcal{E}_t) \leftarrow$ SAMPLE$(\mathcal{G})$.
  GAE construction on $\mathcal{G}_t$.
  $\mathbf{Z_t} \leftarrow$ Forward propagation of $\{\mathbf{A_t}, \mathbf{X_t}\}$ by the encoder of SAGES, normalized by α.
  $\hat{\mathbf{A}}_\mathbf{t}, \hat{\mathbf{X}}_\mathbf{t} \leftarrow$ Reconstruct $\mathbf{A_t}, \mathbf{X_t}$ based on $\mathbf{Z_t}$ by attribute-sensitive decoder.
  $\mathcal{L}_A \leftarrow$ Compute $\lambda_{u,v}$-normalized structure loss$(\mathbf{A_t}, \hat{\mathbf{A}}_\mathbf{t})$.
  $\mathcal{L}_X \leftarrow$ Compute λ_v-normalized content loss$(\mathbf{X_t}, \hat{\mathbf{X}}_\mathbf{t})$.
  $\mathcal{L}_c \leftarrow$ Compute community-level loss.
  Backward propagation according to $\mathcal{L}_A, \mathcal{L}_X$ and $\mathcal{L}_c$.
  Update weights.
end for

4.2 Proposed Method: SAGES

4.2.1 *Structure & Content Sensitive Graph Sampler*

The full-batch GAE is not scalable on the large attributed graph because its training requires the calculation of all node representations per layer. As discussed in Section 2.2, different sampling-based methods have been proposed to train supervised GNNs. Similar to [30], we choose the subgraph sampling based method instead of layer sampling methods to reduce the sampling complexity.

In our model, we first sample subgraphs using the proposed sampler. Then we train our model on the self-supportive subgraphs to generate unsupervised representations for each node. The goal of our proposed sampler is to extract subgraphs with losing little structure and content information. Intuitively, nodes with higher influence with each other should have a higher probability of being sampled into the same subgraph. According to [37], [38], for attributed graphs, ignoring node attributes when sampling nodes may omit informative samples and introduce noises if neighbors with

inconsistent contents are sampled, which reduces the ability of GNN-based models for downstream tasks. Our intuition is that a good sampler should consider both the node attributes and the graph structure.

Previous works such as VR-GCN [39], LADIES [40] and GraphSAINT [30] have shown that reducing the variance of estimations of the node embeddings caused by sampling in mini-batch GCN training leads to higher model accuracy and higher convergence speed. We prove that the sampler SAGES we proposed in this section is a near optimal solution of minimizing the estimation variance under Equation 8. Details are in Appendix A, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2022.3148272.

Specifically, based on the calculation formula of the node embeddings in each layer of our model presented in Equation (8), we can derive the variance of estimations of the node embeddings in the complete multi-layer GCN model following GraphSAINT [30]. We consider a sampling strategy where we sample neighbors for the current node in each step. The proportion of the probabilities of neighbors v_1 and v_2 being sampled conditioned on the current node u can be calculated by Equation (27). We derive that $p_{v_k|u} \propto \cos\theta_k$, where $p_{v_k|u}$ is the probability of the node v_k being sampled conditioned on the current node u, and θ_k is the angle between the node embeddings of u and v_k. Compared to neighbor v_j, v_i is more likely to influence u if the hidden embeddings of neighbor v_i is more similar to the current node u's embedding. The embeddings here are related to node attributes and the graph structure according to the definition in Equation (25), which agrees with our intuition that the optimal sampler should consider both the node attributes and the graph structure. We perform a simple linear K-hop graph convolution transformation: $\tilde{\mathbf{X}} = \mathbf{S}^K\mathbf{X}$. We use $\tilde{\mathbf{X}}$ to approximate the node embeddings in each layer. Then we derive the near optimal sampling probability $p_{v_k|u} \propto \cos < \tilde{\mathbf{x}}_u, \tilde{\mathbf{x}}_{v_k} >$ to minimize the variance of estimations of the node embeddings calculated by Equation (8).

We present the simple linear K-hop graph convolution transformation to generated $\tilde{\mathbf{X}}$:

$$\tilde{\mathbf{X}} = \mathbf{S}^K\mathbf{X}, \quad \mathbf{S} = \widetilde{\mathbf{D}}^{-\frac{1}{2}}\widetilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-\frac{1}{2}} \tag{1}$$

where $\widetilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$, $\mathbf{I}$ is identity matrix, and $\widetilde{\mathbf{D}}$ is the degree matrix of $\widetilde{\mathbf{A}}$. Notably, [41] has proved that the simple linear graph convolution transformation can capture the information of graph structure and node attributes. This is because the expressive power of GCN originates primarily from the repeated graph propagation rather than the nonlinear feature extraction.

We also emphasize that the renormalization trick (add self-loops to original graph) [5] is very helpful here. The simple linear graph convolutional transformation with renormalization trick acts as a lowpass filter in graph spectrum perspective that produces smooth features over the graph. This can lead nearby nodes with similar content tend to share similar representations.

To show the effectiveness of the simple linear graph convolutional transformation, we visualize the Cora dataset in a two-dimensional space by applying the t-SNE [42] algorithm on raw node feature matrix $\mathbf{X}$ and the node embedding matrix $\tilde{\mathbf{X}}$, respectively. We use a linear 2-hop graph convolution transformation to generate $\tilde{\mathbf{X}}$ here. The result in Fig. 3 demonstrates that the node representation after linear graph convolution transformation is more meaningful than the raw node feature. Compared with the raw node feature, our node embeddings' 2D projections exhibit discernible clustering. Thus, we can sample subgraphs with highly correlated nodes, according to the similarity of node embeddings in $\tilde{\mathbf{X}}$.

Fig. 3. The t-SNE visualizations of the node embeddings on the Cora dataset. The *left* visualization illustrates the node embeddings from the raw node features matrix $\mathbf{X}$, and the *right* visualization illustrates the node embeddings from node influence representation matrix $\tilde{\mathbf{X}}$. The clusters of the right visualization based on node influence representation matrix $\tilde{\mathbf{X}}$ are clearly defined.

Algorithm 2. Graph Sampling Algorithms SAMPLE

Input: Graph $\mathcal{G}$; Node representation matrix $\tilde{\mathbf{X}}$;
Sample Measure $\mathcal{M}$; Batch size b; Sample depth h;
Output: Sampled graph $\mathcal{G}_s(\mathcal{V}_s, \mathcal{E}_s)$;
1: $\mathcal{V}_{root} \leftarrow b$ root nodes sampled randomly according to
2: Sample Measure $\mathcal{M}$ from V of $\mathcal{G}$
3: $\mathcal{V}_s \leftarrow \mathcal{V}_{root}$
4: **for** $v \in \mathcal{V}_{\text{root}}$ **do** ▷ Be able to run in parallel
5: $u \leftarrow v$
6: **for** $d = 1$ to h **do**
7: $P_{nei}(r) := \text{softmax}(\frac{\cos < \tilde{\mathbf{x}}_\mathbf{u}, \tilde{\mathbf{x}}_\mathbf{r} >}{\text{temp}}), \quad r \in \mathcal{N}_u$
8: $u \leftarrow$ node sampled from $\mathcal{N}_u$ according to $P_{\text{nei}}(u)$
9: $\mathcal{V}_s \leftarrow \mathcal{V}_s \cup \{u\}$
10: **end for**
11: **end for**
12: **return** $\mathcal{G}_s \leftarrow$ Node induced subgraph of $\mathcal{G}$ from $\mathcal{V}_s$

Algorithm 2 outlines the graph sampling method used in SAGES. Its basic idea is to sample subgraphs that meet the above three requirements by selecting similar nodes in random walk. In Algorithm 2, we first sample b root nodes for random walk by sample measure $\mathcal{M}$. There are two alternative measures for $\mathcal{M}$: *Uniform Sampling*, which samples nodes from $\mathcal{V}$ uniformly. *Degree Sampling*, which is inspired by [15], [30], samples nodes according to a node probability distribution $P(u) \propto \left\|\tilde{\mathbf{A}}_{:,u}\right\|^2$. Here, $\tilde{\mathbf{A}}_{:,u} = \sum_{i=1}^{N} \tilde{\mathbf{A}}_{i,u}$. In practice, we find that *Uniform Sampling* is good enough most of the time. Starting from getting the root nodes, we walk $h-$hops for each walker. In each hop of the walker, we randomly sample one neighbor node of the current node u according to P_{nei} defined in line 7 of Algorithm 2. P_{nei} can measure influence between node and its neighbors, $temp$ is temperature in softmax to control the smoothness of the distribution [43]. Thus, P_{nei} can adapt to different node representation space distribution by controlling $temp$. In the end,

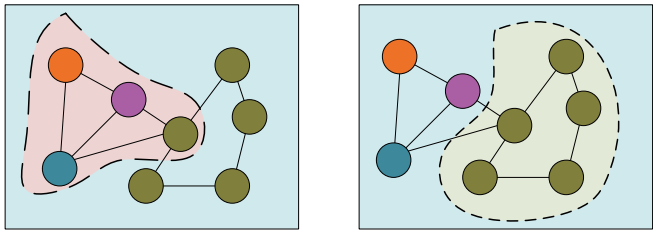

Fig. 4. An illustration of different sampling methods. Nodes with different colors have different labels. The dotted line delineated area represents a sampled subgraph. For the **left** sampler that only depends on graph topology, the nodes in the subgraph sampled may differ relatively from each other in attribute, which makes the model difficult to learn useful information in the subgraph.

we induce a well-connected subgraph whose nodes are highly related according to $\mathcal{V}_s$.

As shown in Fig. 4, compared with the graph sampler which only considers graph structure, our graph sampler considering both graph structure and node features, is a better choice. Note that our graph sampler will not cost too much time. The process of sampling each subgraph is independent, so we can sample subgraphs in parallel. For preprocessing part, we will give a detailed explanation in Section 4.3.

4.2.2 Unbiased Attributed Graph Encoder on Subgraph

SAGES framework is relatively flexible, and many GNNs architecture variants are suitable to be our encoder. For simplicity, we will introduce GCN as our encoder. GCN incorporates spectral convolutions into neural networks to learn node representations. We denote encoder GCN as Φ. Given the subgraph $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t, \mathbf{X_t})$ with adjacency matrix $\mathbf{A_t}$, the l-th layer of encoder Φ is as follows:

$$\mathbf{H_t}^{(l+1)} = \sigma\Big(\mathbf{S_t}\mathbf{H_t}^{(l)}\mathbf{W}^{(l)}\Big), \quad \mathbf{S_t} = \widetilde{\mathbf{D}_\mathbf{t}}^{-\frac{1}{2}}\widetilde{\mathbf{A}_\mathbf{t}}\widetilde{\mathbf{D}_\mathbf{t}}^{-\frac{1}{2}} \tag{2}$$

where $\mathbf{H_t}^{(l)}$ is the node embedding of the l-th layer, $\mathbf{W}^{(l)} \in \mathbb{R}^{f^{(\ell)} \times f^{(\ell+1)}}$ are trainable parameters of node transformation, and $\mathbf{S_t}$ is the symmetric normalization matrix of $\mathbf{A_t}$ with self-loops. σ is the activation function here, it can be Identity Function : $f(x) = x$ or RELU Function : $f(x) = \max(x, 0)$. For layer 0 to $L-1$, we use Identity Function. For layer L, we use RELU. The initial node representations are just the original input features, which means $\mathbf{H_t}^{(0)} = \mathbf{X_t}$. After applying L encoder layers, we consider the output of the last layer as the final node representations matrix $\mathbf{Z_t}$ learned by GAE. Thus for GCN encoder Φ we have

$$\Phi(\mathbf{A}_t, \mathbf{X}_t; \Theta_\Phi) = \mathbf{Z_t}, \quad \mathbf{Z_t} = \mathbf{H_t}^{(L)} \tag{3}$$

where all trainable parameters of encoder Φ can be expressed as

$$\Theta_\Phi = \Big\{\mathbf{W}^{(l)}\Big\}_{l=0,\cdots,L-1} \tag{4}$$

Note that, to sample appropriate subgraph, our graph sampler preserves connectivity characteristics of G and pushes the nodes with high impact to gather together. This will result in a skewed sampling of nodes and edges. Therefore, it is inevitable that this sampler will introduce bias into the mini-batch estimation. So we propose a normalization technique to eliminate bias of forward propagation and back propagation in training GAE, and we focus on encoder in this part.

For simplicity, we only analyze GCN here, and other GCN architecture variants can be extended from it. Analysis of the complete multi-layer GCN with nonlinear activation is difficult, and sometimes removing the nonlinear activation function will preserve or even improve performances. Thus, we analyze feature propagation of each layer independently. Given node $v \in \mathcal{V}_t$, and $u \in \mathcal{V}$, we can compute the aggregated feature of v in the $(l+1)$-th layer as:

$$\eta_v^{(l+1)} = \sum_{u \in \mathcal{N}_v} \mathbf{S}_{v,u}\Big(\mathbf{W}^{(l)}\Big)^T \mathbf{h}_u^{(l)} \mathbb{1}_{u|v} \tag{5}$$

where node u is the neighbor of node v, $\mathbf{h}_u^{(l)}$ is the node u's representation in the $l-$th layer, $\mathbf{S}_{v,u}$ is the element of row v and column u in $\mathbf{S}$. And $\mathbb{1}_{u|v} \in \{0, 1\}$ is the indicator function indicating whether u is in the subgraph (i.e., $\mathbb{1}_{u|v} = 0$ if $(u, v) \notin \mathcal{E}_s$; $\mathbb{1}_{u|v} = 1$ if $(u, v) \in \mathcal{E}_s$).

Define p_v as the probability of the node v being sampled in a subgraph, and $p_{u,v} = p_{v,u}$ as the probability of the edge $(u, v) \in \mathcal{E}$ being sampled in the subgraph. Then, we have

$$\mathbb{E}\big(\mathbb{1}_{u|v}\big) = P((u, v) \text{ sampled } | v \text{ sampled}) = \frac{p_{u,v}}{p_v} \tag{6}$$

Equation (6) (conditional edge probability) holds is due to the initial condition that v is sampled in a subgraph. Then, because the linearity of expectation, the estimation of aggregator in a minibatch is as follow:

$$\mathbb{E}\Big(\eta_v^{(l+1)}\Big) = \sum_{u \in \mathcal{V}} \frac{\mathbf{S}_{v,u}}{\alpha_{u,v}}\Big(\mathbf{W}^{(l)}\Big)^T \mathbf{h}_u^{(l)}, \quad \alpha_{u,v} = \frac{p_{u,v}}{p_v} \tag{7}$$

So we can get an unbiased estimator of the aggregation of node v in full GCN of $\mathcal{G}$ each minibatch by calculating node embedding as:

$$\mathbf{h}_v^{(l+1)} = \sum_{u \in \mathcal{N}_v} \frac{\mathbf{S}_{v,u}}{\alpha_{u,v}}\Big(\mathbf{W}^{(l)}\Big)^T \mathbf{h}_u^{(l)} \mathbb{1}_{u|v} \tag{8}$$

where $\mathbf{h}_v^{(l+1)}$ is the node v representation in the $(l+1)-$th layer. $\alpha_{u,v}$ is called *aggregator normalization coefficients*, it is used to normalize eliminate bias.

4.2.3 Joint Loss Function in Decoder

In this section, we carefully design three regulators to restrict the generated unsupervised node embeddings from different aspects. Specifically, $\mathcal{L}_A$ restricts the generated embeddings to preserve the connectivity information hidden in the graph; $\mathcal{L}_X$ restricts the generated embeddings to preserve the own content information of each node; and $\mathcal{L}_c$ restricts the node embeddings to preserve the community information, which is relevant to which subgraph each node belongs to (similar to DGI[11]). We describe them respectively in the following.

Sample-Aware Attribute Sensitive Decoder. Most existing GAE based methods only focus on local structure in decoder and use inner product decoder to reconstruct $\mathbf{A_t}$ as follows:

$$\hat{\mathbf{A}}_\mathbf{t} = \text{sigmoid}\big(\mathbf{Z_t}\mathbf{Z_t}^\top\big), \quad \mathcal{L}_A = \text{loss}(\mathbf{A_t}, \hat{\mathbf{A}}_\mathbf{t}) \tag{9}$$

where $\mathcal{L}_A$ is structure reconstruction loss. Specially, for edge $(u, v) \in \mathcal{E}_t$, the loss function is as follow:

$$\mathcal{L}_{u,v} = -\log\left(\frac{1}{1+\exp\big(-\mathbf{z}_u^T\mathbf{z}_v\big)}\right) \tag{10}$$

Although $\mathcal{L}_A$ is flexible enough when training on the entire graph, it is not sufficient on attributed graphs to focus only on structure loss. With the graph sampling, structural information may be distorted due to the missing of partial edges, while the node content information is still intact. Therefore we also reconstruct node feature matrix in the decoder. To gain latent representations that better preserve node attributes on $\mathcal{G}_t$, we need to generate $\hat{\mathbf{X}}_\mathbf{t}$ from the nodes latent embeddings $\mathbf{Z_t}$ and make $\hat{\mathbf{X}}_\mathbf{t}$ as close as possible to original input $\mathbf{X_t}$, so we design attribute-sensitive decoder $\Psi(\mathbf{A_t}, \mathbf{Z_t}; \Theta_\Psi) = \hat{\mathbf{X}}_\mathbf{t}$. Here, the decoder Ψ is a single graph attention layer. We first compute the attention matrix $\mathbf{C_t} \in \mathbb{R}^{N_t \times N_t}$ as follows:

$$\mathbf{C_t} = \text{Softmax}\Big(\text{Sigmoid}\Big(\widehat{\mathbf{M}}_s + \widehat{\mathbf{M}}_r\Big)\Big) \tag{11}$$

$$\mathbf{M}_s = \mathbf{A_t} \odot (\mathbf{v}_s\sigma(\mathbf{W_{att}}\mathbf{Z_t})) \tag{12}$$

$$\mathbf{M}_r = \mathbf{A_t} \odot (\mathbf{v}_r\sigma(\mathbf{W_{att}}\mathbf{Z_t})) \tag{13}$$

where $\odot$ is element-wise multiplication with broadcasting capability and σ is RELU. The trainable parameters include $\mathbf{W_{att}} \in \mathbb{R}^{F' \times F'}$, $\mathbf{v}_s \in \mathbb{R}^{1 \times d^{(k)}}$, and $\mathbf{v}_r \in \mathbb{R}^{1 \times d^{(k)}}$. Then the decoder Ψ is as follow:

$$\Psi(\mathbf{A_t}, \mathbf{Z_t}) = \sigma(\mathbf{C_t}\mathbf{Z_t}) \tag{14}$$

For node $v \in \mathcal{V}_t$, and the node feature matrix $\mathbf{X}$, the content reconstruction loss is defined as:

$$\mathcal{L}_v = \|\mathbf{x}_v - \hat{\mathbf{x}}_v\|_2, \quad \mathcal{L}_X = \text{loss}\Big(\mathbf{X_t}, \hat{\mathbf{X_t}}\Big) \tag{15}$$

Further, sampling strategies lead to the non-identical node/edge probabilities, which introduces biases into mini-batch estimation [30]. So we introduce $\lambda_v = \frac{1}{p_v}$ as *content normalization coefficients* and $\lambda_{u,v} = \frac{1}{p_{u,v}}$ as *edge normalization coefficients* to eliminate biases. We can calculate the unbiased structure reconstruction loss $\mathcal{L}_X$ as

$$\mathcal{L}_X = \frac{1}{N_t}\sum_{v \in \mathcal{V}_t} \lambda_v \mathcal{L}_v \tag{16}$$

where N_t is the number of nodes in $\mathcal{G}_t$. And the unbiased structure reconstruction loss $\mathcal{L}_A$ can be calculate as

$$\mathcal{L}_A = \frac{1}{E_t}\sum_{(u,v) \in \mathcal{E}_t} \lambda_{u,v} \mathcal{L}_{u,v} \tag{17}$$

where E_t is the number of edges in $\mathcal{G}_t$.

To learn better node latent representations which capture both node features and graph structure, we minimize the reconstruction error of $\mathbf{A_t}$ and $\mathbf{X_t}$ of the subgraph as follows:

$$\mathcal{L}_r = \mathcal{L}_X + \beta_r \mathcal{L}_\mathcal{A} \tag{18}$$

where $\beta_r \geq 0$ controls the balance between structure and content. $\mathcal{L}_\mathcal{A}$ is as Formula (9).

Feature propagation and calculating loss function within subgraphs require normalization factors $\alpha_{u,v}$, λ_v and $\lambda_{u,v}$, and these factors depend on p_v and $p_{u,v}$. In general, p_v and $p_{u,v}$ is hard to be derived analytically. Thus, we perform pre-processing for estimation. We repeatedly sample N_{sa} subgraphs by utilizing our sampler, and use counter C_v and $C_{u,v}$ to count times the node or edge appears. Then, we set $p_v = \frac{C_v}{N_{sa}}$ and $p_{u,v} = \frac{C_{u,v}}{N_{sa}}$. Note that these sampled subgraphs can be reused as training minibatches. Therefore, the overhead of pre-processing is small.

Local-Community Mutual Information Maximization. We can naturally own the community structure without extra cost, because nodes within the same subgraph have a high influence on each other both in connection and information perspective. Thus, we design a variant of DGI [11] to incorporate community-level information into the node latent representations. The original DGI focuses on local and global relationships in the entire graph, while we focus on local and community relationships. Here, we maximize the mutual information between the node-level embedding and community-level embedding. We can get the node latent representations $\mathbf{Z_t}$ by the encoder on $\mathcal{G}_t$. Then, we compute the community-level summary representation $\mathbf{s_t}$ by leveraging a *Readout Function* $\mathcal{R} : \mathbb{R}^{N_t \times F'} \rightarrow \mathbb{R}^{F'}$

$$\mathbf{s_t} = \mathcal{R}(\mathbf{Z_t}) = \sigma\left(\frac{1}{N_t}\sum_{i=1}^{N_t} \mathbf{z_i}^{(t)}\right) \tag{19}$$

where $\mathbf{z_i}^{(t)}$ is the embedding of node i in $\mathbf{Z_t}$, and $\mathbf{s_t}$ summarizes the node latent embeddings into a *community-level* embedding. σ is a logistic sigmoid nonlinearity function. Then we employ a *Discriminator Function* $\mathcal{D} : \mathbb{R}^{F'} \times \mathbb{R}^{F'} \rightarrow \mathbb{R}$ to measure the probability scores for node-community pair (should be higher for nodes and their corresponding community). The definition of $\mathcal{D}$ is as follows:

$$\mathcal{D}\Big(\mathbf{z}_i^{(t)}, \mathbf{s_t}\Big) = \sigma\Big(\mathbf{s_t}^T\mathbf{W_d}\mathbf{z}_i^{(t)}\Big) \tag{20}$$

where σ is the logistic sigmoid nonlinearity, and $\mathbf{W_d} \in \mathbb{R}^{F' \times F'}$ is trainable parameters. Then, we generate negative examples for $\mathcal{D}$ in community $\mathcal{G}_t$ by utilizing a *Corruption Function* $\mathcal{C}$:

$$\mathcal{C}(\mathbf{A_t}, \mathbf{X_t}) = (\tilde{\mathbf{A}}_t, \tilde{\mathbf{X}}_t)$$

where we keep graph structure $\mathbf{A_t}$ unchanged, and corrupt the attribute matrix $\mathbf{X_t}$ by shuffling it in the row-wise manner like [11]. Then we use the encoder Φ to generate negative node latent representations $\tilde{\mathbf{Z}}_t = \Phi\big(\tilde{\mathbf{A}}_t, \tilde{\mathbf{X}}_t; \Theta_\Phi\big)$. Next, given $\mathbf{Z}_t, \tilde{\mathbf{Z}}_t$ and $\mathbf{s_t}$, we compute a noise-contrastive community-specific cross entropy

$$\mathcal{L}_c = \sum_{i=1}^{N_t} \log \mathcal{D}\Big(\mathbf{z}_i^{(t)}, \mathbf{s_t}\Big) + \sum_{j=1}^{N_t} \log\Big(1 - \mathcal{D}\Big(\tilde{\mathbf{z}}_j^{(t)}, \mathbf{s_t}\Big)\Big) \tag{21}$$

Joint Learning. To enable the model to learn both node-level and community-level information, the loss function of the whole model is as follows:

TABLE 2
The Statistics of the Benchmark Datasets

Dataset	Task	Nodes	Edges	Features	Classes	Degree	Train/Val/Test Nodes
Cora	Transductive	2,708	5,429	1,433	7	2.0	140/500/1,000
Citeseer	Transductive	3,327	4,732	3,703	6	1.4	120/500/1,000
Pubmed	Transductive	19,717	44,338	500	3	2.2	151,708/23,699/55,334
Flickr	Inductive	89,250	899,756	500	7	10.1	44,625/22,312/22,312
Reddit	Inductive	232,965	11,606,919	602	41	50.0	153,932/23,699/55,334
Amazon	Inductive	1008,606	116,196,671	200	47	115.2	857,315/50,431/400,860

Task indicates experimental settings of node classification.

$$\mathcal{J} = \mathcal{L}_r + \beta \mathcal{L}_c \tag{22}$$

where β controls the balance between local and community relationships.

Remark. We expand Equation 22 and explain the settings of hyper-parameters as follows:

$$\begin{aligned}\mathcal{J} = & \frac{1}{N_t}\sum_{v\in\mathcal{V}_t}\lambda_v\|\mathbf{x}_v - \hat{\mathbf{x}}_v\|_2 \\ & - \beta_r \frac{1}{E_t}\sum_{(u,v)\in\mathcal{E}_t}\lambda_{u,v}\log\left(\frac{1}{1+\exp(-\mathbf{z}_u^T\mathbf{z}_v)}\right) \\ & + \beta\left(\sum_{i=1}^{N_t}\log\mathcal{D}\left(\mathbf{z}_i^{(t)},\mathbf{s_t}\right) + \sum_{j=1}^{N_t}\log\left(1-\mathcal{D}\left(\tilde{\mathbf{z}}_j^{(t)},\mathbf{s_t}\right)\right)\right)\end{aligned} \tag{23}$$

There are four parameters, λ_v, $\lambda_{u,v}$, β_r and β. First, λ_v and $\lambda_{u,v}$ can be computed as $\lambda_v = \frac{1}{p_v}$ and $\lambda_{u,v} = \frac{1}{p_{u,v}}$ given an input graph, where p_v and $p_{u,v}$ can be estimated in the preprocessing stage — In the preprocessing stage, we run the sampler repeatedly to obtain a set of N_{sa} subgraphs, where N_{sa} is the number of subgraphs. Then p_v and $p_{u,v}$ can be computed as $p_v = \frac{C_v}{N_{sa}}$, $p_{u,v} = \frac{C_{u,v}}{N_{sa}}$, where $C_{u,v}$ is the number of times that the edge (u, v) appears in the subgraphs, and C_v is the number of times that the node v appears in the subgraphs. Moreover, users can tune β_r and β to set different weights for different regulators for a given graph to get the best performance.

4.3 Complexity Analysis

In this section, we discuss the preprocessing and analyze the time and space complexity of SAGES to show its scalability. The discussion is divided into two parts: graph sampling and model training.

In graph sampling, we will calculate node influence representation matrix $\tilde{\mathbf{X}}$ mentioned in Formula (1). Note that, the dimension of input feature of graph will not be large, because the raw feature is usually processed by dimension reduction methods [44], [45]. In addition, $\mathbf{A}$ is typically sparse, and K is usually small. Thus, we can exploit fast sparse-dense matrix multiplication [46] to compute $\tilde{\mathbf{X}}$. In our experiment, it takes only 80 seconds to compute the influence representation matrix of a graph (Amazon) with one million of nodes and hundreds of millions of edges.

For model training, we only focus on the complexity of encoder, because the cost of calculating content loss in decoder is the same as that in encoder, and the calculation of structure loss and community loss is not the bottleneck. Here, $\|A_t\|_0$ and $\|A\|_0$ are the number of non-zero elements in $\mathbf{A_t}$ and $\mathbf{A}$, respectively. D is the maximum dimension of all layers, and b is the number of nodes of the largest subgraph. Since we only train the encoder within one subgraph per batch, the computation will purely be matrix products as Formula 2. We need $O(bLD^2)$ time to transform node embeddings and $O(L\|A\|_0 D)$ time to perform the node aggregation function per batch. Therefore, the overall time complexity per batch becomes $O\big(bLD^2 + L\|A_t\|_0 D\big)$, and the overall time complexity per epoch is $O\big(NLD^2 + L\|A\|_0 D\big)$. On average, we only require computing $O(bL)$ embeddings each batch, which is linear instead of exponential to L. In terms of space complexity, in each batch, we only need to load b nodes and store their embeddings, resulting in $O(bLF + LF^2)$ memory complexity.

5 Experiments

In this section, we evaluate the proposed SAGES framework on three medium-size attributed graphs and three large attributed graph datasets. To test the effectiveness and generality of latent representation generated by SAGES encoder, node classification, link prediction and node clustering tasks are conducted. We also take ablation studies to demonstrate that each component of the loss in the decoder is important. All experiments are conducted on a machine with two GPUs (NVIDIA RTX 2080 Ti GPU 12G) and 64 CPUs (Intel Xeon Gold 5218 CPU 2.30GHz). We implement SAGES via the PyTorch Geometric (PyG) [47] package.

5.1 Datasets

SAGES has been studied in six real-world attributed graphs. The details about graph datasets can be found in Table 2. The three medium-size attributed graph datasets used in our experiment include Cora, Citeseer, and Pubmed [21]. They are citation networks that consist of scientific publications as nodes and citation relationships as edges. The node features of these citation networks are unique words in each document. They are widely used for assessment of attributed graph analysis [5].

Besides, there are three large attributed graph datasets, including Flickr, Reddit, and Amazon. Flickr dataset originates from the SNAP website[1], and it forms links between images sharing common metadata from Flickr, the node in Flickr represents one uploaded image, and we create the edge when two images share some common properties. The Reddit dataset is constructed by Reddit posts. The node label is the community that a post belongs to, and node features

1. https://snap.stanford.edu/data/web-flickr.html

are word embeddings of the post. More details of Reddit can be found in [12]. Amazon dataset is collected from [30]. Nodes of the Amazon dataset are products on the website and the edges of the Amazon dataset are created if two products are bought by the same customer. The node feature is the word embeddings of the text reviews, and the node label represents the product category (e.g., movies, books, shoes). Note that, we remove the unlabeled nodes from the original Amazon dataset in [30], select the most important category (the category that contains the most products) as the label, and reconstruct the graph on the remaining nodes.

5.2 Node Classification

We compare our SAGES against the state-of-the-art algorithms on a variety of node classification tasks (transductive as well as inductive) and obtain competitive results. Inductive tasks require generalization to the unseen graph, while transductive can see the whole graph during training, and the detailed difference between them is given by [12]. Like the experiment setting in [11], in each case, SAGES and other unsupervised algorithms will learn node representations in a fully unsupervised manner. Then we will use a simple linear (logistic regression) classifier on learned embeddings, and evaluate different representation methods on the node-level classification results.

5.2.1 Baslines

- Competitors on Medium-size graphs: *DeepWalk* [8] only uses structure based on short random walks. *Node2vec* [18] relies on a biased random walk generation. *Graph Auto-Encoder (GAE)* [6] utilizes GCN as encoder and reconstructs edges. *VGAE* is a variational version of GAE. *STNE* [48] is a deep model that capture both first-order and second-order proximity of nodes in embedding with only structural information being considered. *DGI* [11] maximizes the mutual information between node and graph representations. **GALA** [10] utilizes Laplacian sharpening to reconstruct node features in decoder. *MVGRL* [33] learns graph representation by contrasting structural views of graphs. *AGE* [9] uses a non-parametric Laplacian smoothing filter as the encoder and utilizes adaptive learning in the decoder. **LP** [49] uses label propagation. *Planetoid* [21] learns node representations by predicting available class labels and the neighborhood context. *GCN* [5] incorporates spectral convolutions into graph domain.
- Competitors on large attributed graphs: *GraphSAGE* [12] scales GNNs by sampling uniformly and aggregating features from a node's local neighborhood. As our setup is unsupervised, we compare against the unsupervised GraphSAGE version. *FastGCN* [15] scales GNNs by subsampling the receptive field for each layer. *AS-GCN* [27] uses adaptive sampling to scale graph neural network.

5.2.2 Experiment Setting

Following the general settings in GNNs like [11], [23], we conduct transductive tasks on three medium-size graphs (Cora, Citeseer, and Pubmed), and perform inductive tasks on three large graphs (Flickr, Reddit, and Amazon). We use mean classification accuracy on transductive tasks and use the micro-averaged F1 score on inductive tasks. We follow the same settings (i.e., the train/validation/test split) as [11] on the transductive tasks and take the same setting as [30] on the inductive tasks.

For transductive tasks, we reuse the metrics already reported in [5], [11], [21], [49] for the performance of DeepWalk, GCN, Label Propagation (LP), Planetoid, and DGI. For all GAE based models and the SAGES model, we use a 2-layer GCN with 512 node embedding dimensions as the encoder. Other hyperparameter settings are shown in Table 5.

For inductive tasks, we reuse the metrics reported in [11] for the performance of GraphSAGE, DGI on Reddit, as well as the FastGCN and AS-GCN [30] on Flickr and Reddit. For SAGES, we also use a 2-layer GCN as encoder, the dimension of hidden layer is set to 256 for Flickr and Reddit, and 128 for Amazon. Other hyperparameter settings are shown in Table 5.

For all competitors, we adopt the early stopping, which wait 20 steps until no progress on the validation set. Adam optimizer [50] is used here. We perform a hyperparameter sweep on initial learning rates $\{10^{-3}, 10^{-4}, 10^{-5}\}$ for medium-size graphs and $\{10^{-1}, 10^{-2}, 10^{-3}\}$ for large graphs respectively

5.2.3 Experiment Results

Table 3 shows the evaluation results on transductive tasks. The results demonstrate that SAGES outperforms all unsupervised methods. Particularly, we observe an improvement of 1.2%, 0.7%, and 3.7% over the state-of-the-art unsupervised model DGI (a full-batch training method) on Cora, Citeseer, and Pubmed, respectively. This indicates that learning node representation will benefit from considering content information and community-level information of the subgraph composed of closely related nodes in the decoder.

Table 4 shows the evaluation results on inductive tasks. We compare SAGES with the state-of-the-art unsupervised scalable GNN models and some supervised scalable GNNs models. The results show the scalability and effectiveness of our framework. Notice that the Reddit dataset is the large attributed graph widely used for assessment of scalable GNNs, and the Amazon dataset has one million of nodes and hundreds of millions of edges. It is worth noting that GALA, MVGRL, and AGE are not scalable to train in Reddit and Amazon. We further observe that SAGES outperforms the previous unsupervised sampling-based GCN variants DGI on three datasets by more than 2.4%, 1.0%, and 7.0%, respectively. To our best knowledge, our unsupervised SAGES achieves state-of-the-art test F1 score 52.1 on the Flickr dataset. It confirms the empirical superiority of our graph sampling and three loss function.

5.3 Node Clustering

We compare SAGES with various graph embedding based approaches for node clustering on Cora and Citeseer. Nodes of two datasets are clustered in 6 and 7 topic classes respectively, acting as ground-truth communities. Pubmed is not

TABLE 3
Node Classification Accuracies on Cora, Citeseer, and Pubmed

Available Data	Train Graph	Approach	Cora	Citeseer	Pubmed
X	-	Raw features	$47.9 \pm 0.4\%$	$49.3 \pm 0.2\%$	$69.1 \pm 0.3\%$
A, **Y**	Full Graph	LP	68.0%	45.3%	63.0%
A	Full Graph	DeepWalk	67.2%	43.2%	65.3%
A	Full Graph	Node2vec	68.2%	47.2%	69.3%
A, **X**	Full Graph	DeepWalk + features	$70.7 \pm 0.6\%$	$51.4 \pm 0.5\%$	$74.3 \pm 0.9\%$
A, **X**	Full Graph	GAE	$80.7 \pm 0.5\%$	$69.0 \pm 0.7\%$	$76.1 \pm 0.6\%$
A, **X**	Full Graph	STNE	$79.8 \pm 0.5\%$	$68.2 \pm 0.7\%$	$74.1 \pm 0.6\%$
A, **X**	Full Graph	DGI	$82.3 \pm 0.6\%$	$71.8 \pm 0.7\%$	$76.8 \pm 0.6\%$
A, **X**	Full Graph	GALA	$81.0 \pm 0.3\%$	$72.1 \pm 0.2\%$	$76.5 \pm 0.4\%$
A, **X**	Full Graph	MVGRL	$82.0 \pm 0.1\%$	$71.6 \pm 0.2\%$	$75.8 \pm 0.4\%$
A, **X**	Full Graph	AGE	$80.0 \pm 0.2\%$	$72.0 \pm 0.1\%$	$76.8 \pm 0.5\%$
A, **X**	Subgraph	**SAGES (ours)**	$\mathbf{84.5} \pm 0.3\%$	$\mathbf{73.2} \pm 0.3\%$	$\mathbf{80.5} \pm 0.4\%$
A, **X**, **Y**	Full Graph	Planetoid	75.7%	64.7%	77.2%
A, **X**, **Y**	Full Graph	GCN	81.5%	70.3%	79.0%

Available Data shows the type of data used during training for each method. Data types can be found in Table 1. Y here means labels. Train Graph refers to the graph used in training.

TABLE 4
Node Classification Results on Large Graphs (Micro-F1)

Available Data	Approach	Flickr	Reddit	Amazon
A, **X**	GraphSAGE-GCN	0.489 ± 0.001	0.908 ± 0.002	0.703 ± 0.001
A, **X**	GraphSAGE-mean	0.487 ± 0.002	0.897 ± 0.001	0.725 ± 0.003
A, **X**	GraphSAGE-LSTM	0.451 ± 0.003	0.907 ± 0.004	0.687 ± 0.002
A, **X**	GraphSAGE-pool	0.456 ± 0.001	0.892 ± 0.001	0.692 ± 0.002
A, **X**	AGE	0.493 ± 0.002	-	-
A, **X**	DGI	0.497 ± 0.002	0.940 ± 0.001	0.751 ± 0.002
A, **X**	**SAGES** (ours)	$\mathbf{0.522} \pm 0.001$	$\mathbf{0.952} \pm 0.013$	$\mathbf{0.823} \pm 0.013$
A, **X**, **Y**	FastGCN	0.504 ± 0.001	0.924 ± 0.001	0.549 ± 0.013
A, **X**, **Y**	AS-GCN	0.504 ± 0.002	0.958 ± 0.001	-

Available Data is the same meaning as Table 3. The symbol '-' denotes that the competitor is not scalable on the datasets.

suitable for node clustering tasks here because it has only three communities.

5.3.1 Baselines

In addition to the methods listed above, some baselines which are designed for clustering are also included. *K-means* is a classical clustering method. *Graph Encoder* [51] trains stacked autoencoder to get node embedding. *DNGR* [52] utilizes stacked denoising autoencoder for graph embedding. To learn node representations, *RTM* [53] considers both text and citation, *RMSC* [54] employs a multi-view method, *TADW* [20] applies matrix factorization. *ARGA* [32] uses adversarially regularized graph autoencoders to learn graph embedding, and *ARVGA* is a variational version of ARGA. *SAGES* is our method.

TABLE 5
Hyperparameter Settings

Dataset	$temp$	K	Subgraph Size	β_r	β
Cora	0.5	3	600	5.0	3.0
Citeseer	0.5	6	600	20.0	1.0
Pubmed	1.0	2	2000	5.0	1.0
Flickr	0.5	2	5000	10.0	1.0
Reddit	0.3	2	8000	10.0	1.0
Amazon	1.0	2	8000	10.0	1.0

5.3.2 Experiment Setting

For node clustering tasks, we have the same experiment settings as [9]. For representation learning methods, we apply Spectral Clustering on the generated embeddings, and select the best epoch by DaviesBouldin index (DBI) [55]. In particular, the node embeddings here are scaled to the $[0, 1]$ interval by min-max scaler for variance reduction. For other works that specify on the node clustering task, we reuse the metrics already reported in [32]. For metrics, we follow [54], and employ three metrics to validate the results: accuracy (ACC), normalized mutual information (NMI), and average rand index (ARI). All metrics used in node clustering are related to the labels. A better result should lead to higher values for all the metrics. For SAGES, some hyperparameter settings are shown in Table 5. Other parameters and experiment settings are the same as the node classification task.

5.3.3 Experiment Results

Table 6 shows the results of the node clustering tasks. SAGES outperform other methods across most of the evaluation metrics. We can observe that the models using both

TABLE 6
Clustering Results on Cora and Citeseer

	Cora			Citeseer		
	ACC	NMI	ARI	ACC	NMI	ARI
K-means	0.492	0.321	0.230	0.540	0.305	0.279
Spectral	0.367	0.127	0.031	0.239	0.056	0.010
GraphEncoder	0.325	0.109	0.006	0.225	0.033	0.010
DeepWalk	0.484	0.327	0.243	0.337	0.088	0.092
RTM	0.440	0.230	0.169	0.451	0.239	0.023
RMSC	0.407	0.255	0.090	0.295	0.139	0.049
TADW	0.560	0.441	0.332	0.455	0.291	0.228
GAE&VGAE	0.609	0.436	0.346	0.408	0.176	0.124
ARGA	0.640	0.449	0.352	0.573	0.350	0.341
ARVGA	0.638	0.450	0.374	0.544	0.261	0.245
GALA	0.746	0.577	0.532	0.693	0.441	0.446
MVGRL	0.732	0.567	0.523	0.658	0.409	0.402
AGE	0.768	0.607	0.565	0.702	0.448	0.457
SAGES(ours)	**0.770**	**0.612**	**0.566**	**0.703**	**0.449**	**0.460**

content and structure information generally perform better than other autoencoder methods. For example, GALA, AGE, and our methods outperform all the baselines without GNNs encoder. In particular, AGE benefits from adaptive learning that performs well in this task, but it is not scalable and performs mediocre in node classification, while SAGES are competitive in all kinds of tasks. The reasons for this are that i) the attribute-sensitive decoder can better capture node feature and graph structure than a simple inner-product decoder. ii) Maximizing the local mutual information of the community is beneficial to capture the relevant information of clustering.

5.4 Link Prediction

We compare SAGES with various graph representation methods for link prediction on Cora and Citeseer. The comparision algorithms are listed in 5.2.1.

5.4.1 Experiment Setting

We partition the datasets following the experimental settings of AGE [9], removing 5% edges for validation and 10% edges for test. To predict whether there is a potential edge existing between two nodes, we use $\hat{\mathbf{A}} = \text{sigmoid}(\mathbf{Z}\mathbf{Z}^{\top})$ for the node latent embeddings $\mathbf{Z}$. For metrics, we follow [9], and report Area Under Curve (AUC) and Average Precision (AP) scores. A higher value indicates better performance. The training procedures and hyperparameters are consistent with the node classification tasks.

TABLE 7
Link Prediction Results on Cora and Citeseer

	Cora		Citeseer	
	AUC	AP	AUC	AP
GAE	0.910	0.920	0.895	0.899
VGAE	0.914	0.926	0.908	0.920
ARGA	0.924	0.932	0.919	0.930
ARVGA	0.924	0.926	0.924	0.930
GALA	0.921	0.922	0.944	0.948
MVGRL	0.918	0.920	0.933	0.945
AGE	0.942	0.943	0.963	0.965
SAGES(ours)	**0.955**	**0.963**	**0.969**	**0.974**

5.4.2 Experiment Results

Table 7 shows the results of the link prediction tasks. We report mean scores of Area Under Curve (AUC) and Average Precision (AP) with 10 random initializations. Compared with state-of-the-art methods, SAGES outperform them on both AUC and AP.

5.5 In-Depth Analysis

5.5.1 Network Visulization

To qualitatively investigate the effectiveness of the node embeddings learned by SAGES, we utilize t-SNE [42] to project the learned node embeddings of different models into a two-dimensional space. We focus our analysis exclusively on the Cora dataset, because the Cora dataset has the smallest number of nodes, and this significantly aiding clarity.

In Fig. 5, we give the t-SNE visualizations of four kinds of node embeddings in Cora, including node embeddings from raw features, node embeddings learned from a GAE model, node embeddings learned from a DGI model, and node embeddings learned from our SAGES model. Here, all three unsupervised GNNs models follow the graph autoencoder framework, they all use a 2-layer GCN layer as the encoder, and the dimension of node embeddings in all hidden layers is set as 512. In the decoder, the GAE model utilizes inner-product layer to reconstruct edge information in the graph, the DGI model maximizes the mutual information between local patches of a graph and the global representation of the entire graph, and our SAGES model reconstruct structure-level, content-level, and community-level information from properly-sampled subgraphs.

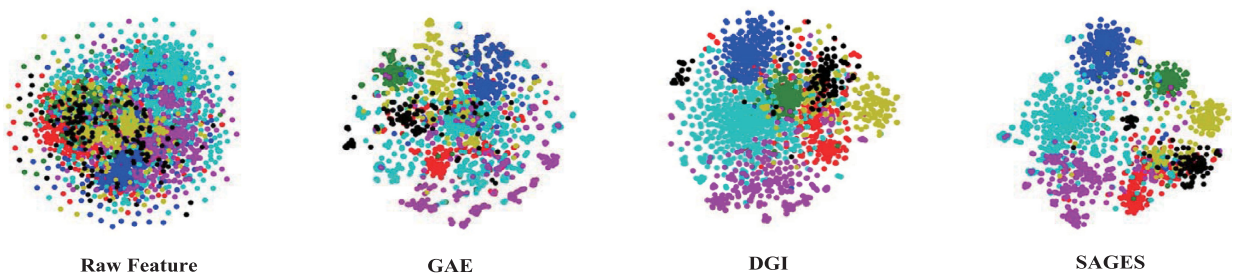

Fig. 5. The t-SNE visualizations of the node representations in the Cora dataset from the raw features (*Raw Feature*), a learned Graph Autoencoder model (*GAE*), a learned Deep Graph Infomax model (*DGI*), and a learned SAGES model (*SAGES*). Node colors denote classes. The clusters of the learned *SAGES* model's embeddings are clearly defined.

TABLE 8
Ablation Study of Loss Function

	Cora	Reddit
	ACC	F1
SAGES(C)	81.5 %	0.910
SAGES(X)	80.7 %	0.894
SAGES(S)	81.9 %	0.920
SAGES(S,X)	82.8 %	0.926
SAGES(S,C)	83.0 %	0.925
SAGES(X,C)	82.5 %	0.912
SAGES	**84.5**%	**0.95**

S means structure loss, X means content loss, C means community loss.

TABLE 9
Sampling Method in Cora and Citeseer

	Cora	Reddit
	ACC	F1
Uniform Sampling	78.5 %	0.890
Degree Sampling	80.8 %	0.910
Random Walk Sampling	76.0 %	0.921
Core Sampling	79.5 %	0.905
SAGES Sampling	**84.5**%	**0.950**

By analyzing the t-SNE visualizations in Fig. 5, we have the following observations: i) Compared to the raw features, all three unsupervised GNN models exhibit discernible clustering in the 2D projected space. ii) The node embeddings of GAE model are more meaningless than the other two models, we can see more overlap and confusion in visualizations of GAE, which shows only reconstructing structural information is not enough to learn meaningful node embeddings. iii) The node embeddings of DGI model are already meaningful. We can observe seven discernible clusters, although the red, black, and green parts of figure are somewhat overlapped. iv) By combining reconstruction loss of structure, content, and community information, the node embeddings become more evident in SAGES model, with less overlapping and each group of nodes gathered together, especially in the green and black parts.

5.5.2 Ablation Study

To measure the impact of each component of the decoder in SAGES, we conduct an ablation study of SAGES on Cora and Reddit dataset. The experimental results are shown in Table 8. The approach SAGES(S) means that SAGES only reconstructs structural information in the decoder, and so on in other cases. The hyperparameter setting here is the same as the above experimental part. By analyzing the table, we have the following observations: i) SAGES outperforms other simplified versions, indicating that each component contributes to the overall performance of our architecture. ii) Even in the case of using one loss function, SAGES still maintains good performance, which shows the effectiveness and robustness of our graph sampler. iii) Graph structural information is crucial for representation learning of nodes, while reasonably combining node content information and community-level information can have a better impact on node embedding learning.

Then, we conduct ablation study on Cora and Reddit to manifest the efficacy of graph sampling methods of SAGES. The sampling method of comparison is as follows: *Uniform Sampling* method samples nodes uniformly to form a subgraph in each minibatch. *Degree Sampling* method samples nodes according to *Degree Sampling* mentioned in Section 4.2.1. *Core Sampling* method sample nodes according to core number [14]. The core number of a node corresponds to the largest value of k for which node is in the $k-$core [56]. *Random Walk Sampling* method use a regular random walk sampler, which selects r root nodes uniformly in random and goes h hops. The experimental results are shown in Table 9. We can observe that our sampling method is significantly better than other methods. In addition, degree-based sampling and core-based sampling are empirically more effective than uniform sampling, while the effect of the regular random walker is not stable. The results give verification to the rationality of the graph sampler of SAGES.

We also investigate the influence of K value in the Formula (1) on the performance of SAGES and report the results in Fig. 6. It shows that the accuracy of SAGES increases at first and then decreases with the increase of K. This is as expected, as nodes usually get information from their neighbors at the beginning. However, as K increases, although linear graph convolutional transformation has the renormalization trick, it also suffers over-smoothing [57] problem. Besides, considering the efficiency and effectiveness of the graph sampling algorithm, we choose a smaller K value for large datasets in the experiment.

5.5.3 Runtime

We also report the running time of SAGES and GraphSAGE on the largest attributed graph Amazon. The time cost is shown in Table 10. The GraphSAGE here is GraphSAGE-LSTM, and we sample 10 and 25 neighbors at the first and second level respectively to build receptive field. The preprocessing of GraphSAGE refers to the process of generating negative sampling by random walk. The size of subgraph in SAGES is 8000. The preprocessing of SAGES refers to the process of calculating influence matrix and sampling subgraphs to setup normalized parameters. We can find that SAGES improves the performance of the model without

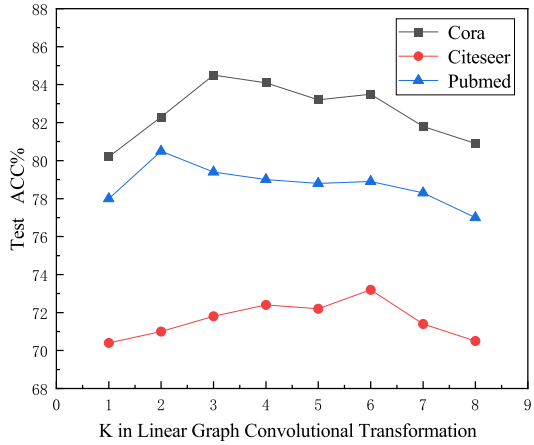

Fig. 6. K value of SAGES.

TABLE 10
Comparisons of Running Time on Amazon Dataset

	Prerpocessing	Training	Total time cost
GraphSAGE	1490s	2.32s	2,054s
SAGES	1780s	1.94s	2,218s

Training here means training time per epoch.

consuming too much time. We can also observe that the training time of SAGES is shorter than that of GraphSAGE. It agrees with the proposition [30], [39], [40] that reducing the variance of estimations of the node embeddings caused by sampling in mini-batch GCN training leads to higher model accuracy and higher convergence speed.

6 Conclusion

This paper proposes SAGES, a scalable unsupervised representation learning framework on attributed graphs. SAGES utilizes a structure and content sensitive sampler to locate highly related nodes into subgraphs, on which an unbiased graph autoencoder is applied to learn the latent node representation with guide of structure, content and community loss. Thus, SAGES promotes both the effectiveness and scalability in unsupervised graph learning. Experimental results show that SAGES outperforms the competitors. For example, SAGES achieves new state-of-the-art F1 scores for Reddit (0.950) and Flickr (0.522).

Acknowledgments

Jialin Wang and Xiaoru Qu contributed equally to this work.

References

[1] X. Huang, J. Li, and X. Hu, "Label informed attributed network embedding," in *Proc. 10th ACM Int. Conf. Web Search Data Mining*, 2017, pp. 731–739.
[2] X. Wang, P. Cui, J. Wang, J. Pei, W. Zhu, and S. Yang, "Community preserving network embedding," in *Proc. 31st AAAI Conf. Artif. Intell.*, 2017, pp. 203–209.
[3] H. Gao and H. Huang, "Deep attributed network embedding," in *Proc. 27th Int. Joint Conf. Artif. Intell.*, 2018, pp. 3364–3370.
[4] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh, "Cluster-GCN: An efficient algorithm for training deep and large graph convolutional networks," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2019, pp. 4–8.
[5] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. Int. Conf. Learn. Representations*, 2017.
[6] T. N. Kipf and M. Welling, "Variational graph auto-encoders," in *Proc. Conf. Neural Informat. Process. Syst. Workshop Bayesian Deep Learn.*, 2016.
[7] A. Ahmed, N. Shervashidze, S. Narayanamurthy, V. Josifovski, and A. J. Smola, "Distributed large-scale natural graph factorization," in *Proc. 22nd Int. Conf. World Wide Web*, 2013, pp. 37–48.
[8] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proc. 20th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2014, pp. 701–710.
[9] G. Cui, J. Zhou, C. Yang, and Z. Liu, "Adaptive graph encoder for attributed graph embedding," in *Proc. 26th ACM Int. Conf. Knowl. Discov. Data Mining*, 2020, pp. 976–985.
[10] J. Park, M. Lee, H. J. Chang, K. Lee, and J. Y. Choi, "Symmetric graph convolutional autoencoder for unsupervised graph representation learning," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2019, pp. 6519–6528.
[11] P. Veličković, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm, "Deep graph infomax," in *Proc. Int. Conf. Learn. Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=rklz9iAcKQ
[12] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. Adv. Neural Informat. Process. Syst.*, 2017, pp. 1024–1034.
[13] G. Salha, R. Hennequin, V. A. Tran, and M. Vazirgiannis, "A degeneracy framework for scalable graph autoencoders," in *Proc. 28th Int. Joint Conf. Artif. Intell.*, 2019, pp. 3353–3359.
[14] G. Salha, R. Hennequin, J.-B. Remy, M. Moussallam, and M. Vazirgiannis, "FastGAE: Fast, scalable and effective graph autoencoders with stochastic subgraph decoding," 2020, *arXiv:2002.01910*.
[15] J. Chen, T. Ma, and C. Xiao, "FastGCN: Fast learning with graph convolutional networks via importance sampling," *CoRR*, vol. abs/1801.10247, 2018. [Online]. Available: http://arxiv.org/abs/1801.10247
[16] Z. Xu, Y. Ke, Y. Wang, H. Cheng, and J. Cheng, "A model-based approach to attributed graph clustering," in *Proc. ACM SIGMOD Int. Conf. Manag. Data*, 2012, pp. 505–516.
[17] P. V. Marsden and N. E. Friedkin, "Network studies of social influence," *Sociol. Methods Res.*, vol. 22, no. 1, pp. 127–151, 1993.
[18] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2016, pp. 855–864.
[19] H. Chen, H. Yin, W. Wang, H. Wang, Q. V. H. Nguyen, and X. Li, "PME: Projected metric embedding on heterogeneous networks for link prediction," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2018, pp. 1177–1186.
[20] C. Yang, Z. Liu, D. Zhao, M. Sun, and E. Chang, "Network representation learning with rich text information," in *Proc. 24th Int. Joint Conf. Artif. Intell.*, 2015, pp. 2111–2117.
[21] Z. Yang, W. W. Cohen, and R. Salakhutdinov, "Revisiting semi-supervised learning with graph embeddings," 2016, *arXiv:1603.08861*.
[22] P. Liang and N. Bose, *Neural Network Fundamentals With Graphs, Algorithms, and Applications*. New York, NY, USA: McGraw-Hill, 1996.
[23] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *Proc. Int. Conf. Learn. Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=rJXMpikCZ
[24] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2018, pp. 974–983.
[25] H. Chen, H. Yin, T. Chen, Q. V. H. Nguyen, W.-C. Peng, and X. Li, "Exploiting centrality information with graph convolutions for network representation learning," in *Proc. IEEE 35th Int. Conf. Data Eng.*, 2019, pp. 590–601.
[26] J. Chen, J. Zhu, and L. Song, "Stochastic training of graph convolutional networks with variance reduction," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 941–949.
[27] W. Huang, T. Zhang, Y. Rong, and J. Huang, "Adaptive sampling towards fast graph representation learning," in *Proc. Adv. Neural Informat. Process. Syst.*, 2018, pp. 4558–4567.
[28] A. K. Bhowmick, K. Meneni, M. Danisch, J.-L. Guillaume, and B. Mitra, "Louvainne: Hierarchical louvain method for high quality and scalable network embedding," in *Proc. 13th Int. Conf. Web Search Data Mining*, 2020, pp. 43–51.
[29] Z. Zhang, C. Yang, Z. Liu, M. Sun, Z. Fang, B. Zhang, and L. Lin, "COSINE: Compressive network embedding on large-scale information networks," *IEEE Trans. Knowl. Data Eng.*, early access, Oct. 13, 2020, doi: 10.1109/TKDE.2020.3030539.
[30] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. Prasanna, "GraphSAINT: Graph sampling based inductive learning method," in *Proc. Int. Conf. Learn. Representations*, 2020.
[31] P. Baldi, "Autoencoders, unsupervised learning, and deep architectures," in *Proc. Int. Conf. Mach. Learn. Workshop Unsupervised Transfer Learn.*, 2012, pp. 37–49.
[32] S. Pan, R. Hu, G. Long, J. Jiang, L. Yao, and C. Zhang, "Adversarially regularized graph autoencoder for graph embedding," in *Proc. 27th Int. Joint Conf. Artif. Intell.*, 2018, pp. 2609–2615.
[33] K. Hassani and A. H. Khasahmadi, "Contrastive multi-view representation learning on graphs," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 3451–3461.
[34] C. Wang, S. Pan, G. Long, X. Zhu, and J. Jiang, "MGAE: Marginalized graph autoencoder for graph clustering," in *Proc. ACM Conf. Informat. Knowl. Manage.*, 2017, pp. 889–898.
[35] W. Jin, R. Barzilay, and T. Jaakkola, "Junction tree variational autoencoder for molecular graph generation," 2018, *arXiv:1802.04364*.

[36] K. Xu, C. Li, Y. Tian, T. Sonobe, K.-I. Kawarabayashi, and S. Jegelka, "Representation learning on graphs with jumping knowledge networks," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 5453–5462.
[37] L. Yang, Z. Liu, Y. Dou, J. Ma, and P. S. Yu, "ConsisRec: Enhancing GNN for social recommendation via consistent neighbor aggregation," in *Proc. Annu. Conf. Assoc. Comput. Machinery Special Int. Group Informat. Retrieval*, 2021, pp. 2141–2145.
[38] S. Kumar and H. Sundaram, "Attribute-guided network sampling mechanisms," *ACM Trans. Knowl. Discov. Data*, vol. 15, no. 4, pp. 69:1–69:24, 2021.
[39] J. Chen, J. Zhu, and L. Song, "Stochastic training of graph convolutional networks with variance reduction," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 941–949.
[40] D. Zou, Z. Hu, Y. Wang, S. Jiang, Y. Sun, and Q. Gu, "Layer-dependent importance sampling for training deep and large graph convolutional networks," in *Proc. Conf. Neural Informat. Process. Syst.*, 2019, pp. 11247–11256.
[41] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger, "Simplifying graph convolutional networks," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 6861–6871.
[42] L. V. D. Maaten and G. Hinton, "Visualizing data using t-SNE," *J. Mach. Learn. Res.*, vol. 9, no. Nov, pp. 2579–2605, 2008.
[43] J. Schmidt, M. R. Marques, S. Botti, and M. A. Marques, "Recent advances and applications of machine learning in solid-state materials science," *npj Comput. Mater.*, vol. 5, no. 1, pp. 1–36, 2019.
[44] M. E. Wall, A. Rechtsteiner, and L. M. Rocha, "Singular value decomposition and principal component analysis," in *A Practical Approach to Microarray Data Analysis*, Berlin, Germany: Springer, 2003, pp. 91–109.
[45] M. Partridge and R. A. Calvo, "Fast dimensionality reduction and simple PCA," *Intell. Data Anal.*, vol. 2, no. 3, pp. 203–214, 1998.
[46] P. Koanantakool *et al.* "Communication-avoiding parallel sparse-dense matrix-matrix multiplication," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2016, pp. 842–853.
[47] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *Proc. Workshop Representation Learn. Graphs Manifolds*, 2019.
[48] J. Liu, Z. He, L. Wei, and Y. Huang, "Content to node: Self-translation network embedding," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2018, pp. 1794–1802.
[49] X. Zhu, Z. Ghahramani, and J. D. Lafferty, "Semi-supervised learning using gaussian fields and harmonic functions," in *Proc. 20th Int. Conf. Mach. Learn.*, 2003, pp. 912–919.
[50] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.
[51] F. Tian, B. Gao, Q. Cui, E. Chen, and T.-Y. Liu, "Learning deep representations for graph clustering," in *Proc. 28th AAAI Conf. Artif. Intell.*, 2014, pp. 1293–1299.
[52] S. Cao, W. Lu, and Q. Xu, "Deep neural networks for learning graph representations," in *Proc. 30th AAAI Conf. Artif. Intell.*, 2016, pp. 1145–1152.
[53] J. Chang and D. Blei, "Relational topic models for document networks," in *Artif. Intell. Statist.*, 2009, pp. 81–88.
[54] R. Xia, Y. Pan, L. Du, and J. Yin, "Robust multi-view spectral clustering via low-rank and sparse decomposition," in *Proc. 28th AAAI Conf. Artif. Intell.*, 2014, pp. 2149–2155.
[55] D. L. Davies and D. W. Bouldin, "A cluster separation measure," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-1, no. 2, pp. 224–227, Apr. 1979.
[56] F. D. Malliaros, C. Giatsidis, A. N. Papadopoulos, and M. Vazirgiannis, "The core decomposition of networks: Theory, algorithms and applications," *VLDB J.*, vol. 29, no. 1, pp. 61–92, 2020.
[57] Z. W. Ming Chen, B. D. Zengfeng Huang, and Y. Li, "Simple and deep graph convolutional networks," in *Proc. 37th Int. Conf. Mach. Learn.*, 2020, pp. 1725–1735.

Jialin Wang received the BS and MS degrees in computer science, from Peking University. He is currently an researcher with WeChat Search Application Department, Tencent. His current research interests include graph neural networks, recommendation system, and data mining. He has published articles in WWW, CIKM, and DASFAA.

Xiaoru Qu received the BS degree from Peking University in 2018. She is currently working toward the PhD degree with Computer Science Department, Peking University. Her research interests include graph data mining, information retrieval and text mining. She has published articles in CIKM and WWW.

Jinze Bai is currently working toward the PhD degree with Computer Science Department, Peking University. His current research interests include graph neural networks, heterogeneous information networks, and neural architecture search. He has published multiple articles in conferences, including AAAI, WWW, and DASFAA. He is also a contributor to some open-source projects such as Tensorflow and Pytorch.

Zhao Li received the PhD degree (Hons.) from Computer Science Department, University of Vermont. He is currently a senior staff scientist with Alibaba Group, specializing in e-commerce ranking and recommender systems. He has published more than 50 articles in prestigious conferences and journals, including NeurIPS, AAAI, IJCAI, and DMKD. His current research interests include adversarial machine learning, network representation learning, knowledge graphs, multi-agent reinforcement learning, and big data-driven security. He is also a technical committee member of the China Computer Federation on Database.

Ji Zhang (Senior Member, IEEE) received the PhD degree from the Faculty of Computer Science, Dalhousie University, Canada in 2008. He is currently an associate professor in computer science with the University of Southern Queensland (USQ), Australia. He is an ACM member, Australian Endeavour fellow, Queensland fellow (Australia) and Izaak Walton Killam Scholar (Canada). His research interests are big data analytics, knowledge discovery and data mining (KDD), information privacy, and security. He has published more than 160 papers in major peer-reviewed international journals and conferences.

Jun Gao received the BE and ME degrees in computer science from Shandong University, China, in 1997, 2000, and the PhD degree in computer science from Peking University in 2003. Currently he is a professor with the School of Electronics Engineering and Computer Science, Peking University, China. His major research interests include web data management and graph data management.