Network Embedding for Community Detection in Attributed Networks

HELI SUN and FANG HE, Xi'an Jiaotong University
JIANBIN HUANG, Xidian University
YIZHOU SUN, University of California
YANG LI, CHENYU WANG, LIANG HE, ZHONGBIN SUN, and XIAOLIN JIA,
Xi'an Jiaotong University

Community detection aims to partition network nodes into a set of clusters, such that nodes are more densely connected to each other within the same cluster than other clusters. For attributed networks, apart from the denseness requirement of topology structure, the attributes of nodes in the same community should also be homogeneous. Network embedding has been proved extremely useful in a variety of tasks, such as node classification, link prediction, and graph visualization, but few works dedicated to unsupervised embedding of node features specified for clustering task, which is vital for community detection and graph clustering. By post-processing with clustering algorithms like k-means, most existing network embedding methods can be applied to clustering tasks. However, the learned embeddings are not designed for clustering task, they only learn topological and attributed information of networks, and no clustering-oriented information is explored. In this article, we propose an algorithm named Network Embedding for node Clustering (NEC) to learn network embedding for node clustering in attributed graphs. Specifically, the presented work introduces a framework that simultaneously learns graph structure-based representations and clustering-oriented representations together. The framework consists of the following three modules: graph convolutional autoencoder module, soft modularity maximization module, and self-clustering module. Graph convolutional autoencoder module learns node embeddings based on topological structure and node attributes. We introduce soft modularity, which can be easily optimized using gradient descent algorithms, to exploit the community structure of networks. By integrating clustering loss and embedding loss, NEC can jointly optimize node cluster labels assignment and learn representations that keep local structure of network. This model can be effectively optimized using stochastic gradient algorithm. Empirical experiments on real-world networks and synthetic networks validate the feasibility and effectiveness of our algorithm on community detection task compared with network embedding based methods and traditional community detection methods.

CCS Concepts: • **Computing methodologies** → **Knowledge representation and reasoning**; • **Information systems** → *Information integration*;

This work was supported in part by the Science and Technology Foundation of Shenzhen City (Grant: JCYJ20170816100845994), the National Science Foundation of China (Grants: 61672417, 61876138, 61772091, and 61802035), and the Fundamental Research Funds for the Central Universities of China.

Authors' addresses: H. Sun and F. He, Xi'an, Jiaotong University, Department of Computer Science and Technology, West Xian Ning Road, Xi'an, Shanxi, 710000, China; emails: hlsun@mail.xjtu.edu.cn, fanghe.xjtu@gmail.com; J. Huang, Xidian University, School of Software, South Tai Bai Road, Xi'an, Shanxi, 710000, China; email: jbhuang@xidian.edu.cn; Y. Sun, University of California, Los Angeles, Department of Computer Science, Westwood, Los Angeles, CA 90095; email: yzsun@cs.ucla.edu; Y. Li, C. Wang, L. He, Z. Sun, and X. Jia, Xi'an Jiaotong University, Department of Computer Science and Technology, West Xian Ning Road, Xi'an, Shanxi, 710000, China; emails: {ly3117034007, chenyu0823}@stu.xjtu.edu.cn, {lhe, zhongbin725, xlinjia}@mail.xjtu.edu.cn.

1556-4681/2020/05-ART36 $15.00
https://doi.org/10.1145/3385415

Additional Key Words and Phrases: Community detection, network embedding, attributed network, representation learning

ACM Reference format:
Heli Sun, Fang He, Jianbin Huang, Yizhou Sun, Yang Li, Chenyu Wang, Liang He, Zhongbin Sun, and Xiaolin Jia. 2020. Network Embedding for Community Detection in Attributed Networks. *ACM Trans. Knowl. Discov. Data* 14, 3, Article 36 (May 2020), 25 pages.
https://doi.org/10.1145/3385415

1 INTRODUCTION

Network embedding aims to learn a vector representation for each node in the graph, such that two nodes closed to each other in the graph have similar node representation in a low-dimensional embedding space. Such node embedding has been proved very useful as feature inputs for a wide variety of tasks, including node classification [Perozzi et al. 2014; Cao et al. 2015; Kipf and Welling 2016a], link prediction [Grover and Leskovec 2016; Kipf and Welling 2016b], graph visualization [Tang et al. 2015; Wang et al. 2016], and recommendation [Yu et al. 2014; Zhang et al. 2016].

However, few works focus on unsupervised node representation for node clustering task. Real network often consists of network modules or communities in which nodes more tight connected than the rest of the networks. Finding network communities is critical for analyzing the organizational structures and understanding complex systems. Community detection has been intensively studied and many effective methods have been proposed [Raghavan et al. 2007; Xu et al. 2007; Blondel et al. 2008; Rosvall and Bergstrom 2008; Fortunato 2010; Zhou et al. 2009; Cheng et al. 2011; Xu et al. 2012; Yang et al. 2013; Xu et al. 2014; Newman and Clauset 2016; Bojchevski and Günnemann 2018] for complex networks and attributed networks. To learn a good embedding for nodes in attributed graph for clustering task, we have the following two goals to achieve: (a) local structure of the graph must be kept, i.e., nodes which are strongly connected with each other should be similar in the embedding space, and nodes with similar attributes should also be closed to each other in low-dimensional space; and (b) the embeddings in low-dimensional vector space fall in well-organized clusters thus the clustering algorithm like k-means can be applied to.

Utilizing deep learning in spectral clustering has solid theoretical foundation [Tian et al. 2014]. The aims of autoencoder and spectral clustering are similar, to get the low-order representations of input data and keep important information through reconstruction. Besides, autoencoder is more efficient and can easily get sparse representations of nodes. Recently some researches focus on this method. [Di et al. 2017] proposes a deep integration representation (DIR) algorithm which combines graph topology and attribution information to get deep embedding for community discovery. Deep attributes residue graph (DARG) algorithm [Hu et al. 2017a] feeds relevant attribute pairs to a deep neural network to reconstruct new representations for clustering. However, these methods are not designed for community detection. The network embedding and clustering are optimized separately, so the existing method may neglect the higher-order proximity of network and lose some structure information. The research in [Wang et al. 2013] uses a generative model for network to detect overlapping community. [Bhatia and Rani 2018] proposed a learning model based on autoencoder. These methods are effective on overlapping community detection. However, they just consider edge information and leaves attributions and community structures out, which are also vital to community detection.

Recent network representation learning methods, e.g., DeepWalk [Perozzi et al. 2014], Node2Vec [Grover and Leskovec 2016], GraRep [Cao et al. 2015], and DNGR [Cao et al. 2016] use the clustering performance to evaluate the quality of the learned node representations. These works follow the approach that first applies an unsupervised network embedding algorithm to learn embeddings, and then performs k-means clustering on the learned representations to cluster nodes to communities. However, these approaches are not designed for clustering tasks, the community structure is not extensively explored and node representations are not guaranteed to meet the requirements of clustering task. Recently, [Cavallari et al. 2017] introduced the concept of community embedding and manage to close the loop of community detection, community embedding, and node embedding by iteratively optimizing the first-order loss, second-order loss [Tang et al. 2015], and Gaussian Mixture Model [Bishop 2006]. Though extensive experiments have proven the effectiveness of the model, we find some important factors have been neglected. The model optimizes node embedding loss and clustering loss separately, this may corrupt feature space hence lead to meaningless features. Exactly speaking, node embedding procedure cannot make use of clustering result of last iteration, thus may destroy the clustering-oriented features of former iterations. Moreover, the model takes advantage of the upper bound of likelihood of Gaussian Mixture Model as higher-order loss, and optimizes it with respect to the embeddings, which may not promise to achieve the optimal result.

In order to preserve community structure of networks, we try to optimize modularity metric of the graph. Directly integrating modularity maximization module with graph convolution autoencoder module and optimizing with gradient descent algorithm is hard, because modularity maximization problem is believed to be NP-hard [Newman and Girvan 2004]. Considering modularity matrix as a kind of laplacian matrix, spectral clustering is a good way to solve this problem using singular value decomposition technique with some relaxation, aka. allowing community membership indicator matrix to take any real value [Yang et al. 2016]. Based on this relaxation, we can transform the node representation of graph to community membership indicator matrix, such that modularity of graph can be optimized together with graph convolutional autoencoder. We name the relaxed modularity as soft modularity.

Unlike previous work, we consider combing node embedding procedure and clustering procedure together and optimize a unified objective function. In this way, the learning procedure of graph structure and node attribute can be conducted along with clustering procedure harmoniously. Thus, the learned embeddings are suitable for clustering task in attributed network. To learn representation of graph local structure and node attributes, we design a graph convolutional autoencoders base on the graph convolutional networks [Kipf and Welling 2016a]. Together with local structure and node attribute information, the community structure of graphs is also preserved by using the proposed soft modularity, which can be optimized simultaneously with the reconstruction loss of graph convolutional autoencoders as a kind of higher-order regularizer of hidden layers. And then, the central point is how to ensure node embeddings in low-dimensional space falling into well-organized clusters. We define a centroid-based probability distribution and minimize its Kullback–Leibler (KL) divergence to an auxiliary target distribution to simultaneously improve clustering assignment and feature representation in a self-learning manner.

The proposed Network Embedding for node Clustering (NEC) consists of the following two stages: (a) pretraining of the node embedding, including graph convolutional autoencoders and soft modularity maximization; and (b) union training of node embedding loss with clustering loss. In this way, the proposed framework can jointly perform learning features of local structure and clustering. The optimization of NEC can be directly performed with mini-batch SGD. Extensive experiments are carefully designed and conducted, and the results show the effectiveness of our algorithm.

The contributions of our work are summarized as follows:

— We propose a network embedding algorithm NEC for attributed community detection, which can learn representation vectors and community assignment for nodes in attributed

graphs. We are among the early works that dedicate to unsupervised embedding of node features for community detection in attributed networks.
— We introduce a novel relaxed soft modularity method that can efficiently learn embeddings of nodes by exploiting the higher-order proximity of nodes. We manage to fuse the network embedding procedure and community preserving procedure and form a unified model which can be easily optimized to learn features from attributed graphs.
— We combine the attribute graph embedding procedure with the community structure preserving procedure and optimize them simultaneously by forming a unified objective function. By integrating network embedding and clustering together rather than separating them, the performance of community detection can be significantly improved.
— Empirical experiments on various real-world datasets and synthetic datasets demonstrate the superior effectiveness of the proposed algorithm on community detection task.

The rest of this article is organized as follows. In section *Related Work*, we briefly survey related work in community detection and network embedding. We present the detailed techniques of our model in section *Model Description*. In section *Experiments*, we empirically evaluate NEC on community detection task on various datasets and assess the graph visualization and parameter sensitivity aspect of our algorithm. We conclude with a discussion of NEC framework and highlight some directions for future work in section *Conclusion*.

2 RELATED WORK

Traditional Community Detection Methods. Community detection is one of the most popular and well-studied problem of network data analysis. By partitioning the network nodes into clusters, community detection can often help reveal interesting structure of graphs. Vertices within the same community usually have higher probability of being connected to each other than to members of other communities. According to whether there is additional information of graphs that can be used, such as node attributes or edge labels, graph data can be divided into complex networks and attributed networks. Different methods are designed for different kinds of networks. We briefly introduce some popular community detection methods for complex networks and attributed networks.

There are many classical community detection methods developed for complex networks. Spectral graph clustering [Ng et al. 2002; Von Luxburg 2007] is a method to detect communities using spectral properties of networks. Spectral clustering projects the graph nodes into a low-dimensional vector space. The resulting points can be grouped in clusters by using standard partitional clustering techniques like k-means. Louvain method [Blondel et al. 2008] performs a greedy optimization of modularity in a hierarchical manner. Infomap [Rosvall and Bergstrom 2008] is an efficient community method based on map equation, which yields the description length of an infinite random walk, expressing the shannon entropy of the walk within and between clusters. The best partition is the one yielding the minimum description length. It can be applied to directed or weighted networks. Probabilistic generative models like stochastic blockmodel [Holland et al. 1983; Goldenberg et al. 2010] and degree corrected stochastic blockmodel [Karrer and Newman 2011], which fall in the general class of random graph model, are also important approaches for detecting community structure. By fitting blockmodel to empirical network data, we can discover the community structure of graphs [Psorakis et al. 2011; He et al. 2015; Jin et al. 2015].

Attributed networks are ubiquitous in real life and good abstraction for many applications, such as social networks [Khan et al. 2019], coauthor networks [Orman et al. 2015], and protein interaction networks. By combining topology structure and attribution together, we can acquire the deep interrelation information from attributed graphs. Because of this, a lot of research studies focus on the direction in recent years, and some algorithms are proposed for attributed graph clustering and community detection [Xu et al. 2017; Guo et al. 2017b]. SA-Cluster [Zhou et al. 2009] regards node attributes as virtual nodes and constructs an attribute-augmented graph, and performs random walk on the attribute-augmented graph to obtain a unified distance. It adopts the K-mediods algorithm to cluster the nodes based on learned pairwise distance. Inc-Cluster [Cheng et al. 2011] is a faster version of SA-Cluster. Parameter-free identification of cohesive subgroups (PICS) [Akoglu et al. 2012] is a parameter-free community detection algorithm for attributed graphs. Communities from Edge Structure and Node Attributes (CESNA) [Yang et al. 2013] takes advantage of the probabilistic model of BigCLAM [Yang and Leskovec 2013] for generating edges and the logistic model of attributes together to infer the distribution of community assignments. CohsMix [Zanghi et al. 2010] embeds numerical attributes of nodes into the MixNet model [Daudin et al. 2008] for generating link classes. Bayesian attributed graph clustering (BAGC) [Xu et al. 2012] and General bayesian attributed graph clustering (GBAGC) [Xu et al. 2014] extend the cohsMix model to process categorical attributes and weighted graphs. Structure and inference in annotated networks (SIAN) [Newman and Clauset 2016] is a bayesian statistical inference model which constructs a generative network model possessing correlation between community structure and node metadata. By fitting the model, communities and accompanying metadata can be efficiently observed. Partial anomaly identification and clustering in attributed networks (PAICAN) [Bojchevski and Günnemann 2018] is a probabilistic generative model that explicitly models community structure and partial anomalies by generalizing ideas of degree corrected stochastic blockmodel.

Network Embedding Methods. Network embedding, a popular graph representation framework, has been attracting increasing attention in recent years. It aims to project a graph into a low-dimensional space for further applications.

The earliest network embedding approach is based on matrix factorization methods. For example, locally linear embedding (LLE) [Roweis and Saul 2000], IsoMap [Tenenbaum et al. 2000] and Laplacian Eigenmap [Belkin and Niyogi 2002] typically aim to solve the leading eigenvectors of graph affinity matrices as node embedding. However, the complexity of solving eigenvector of matrices is at least quadratic to the number nodes.

In recent years, random walk-based methods are popular because of the success of word embedding in natural language processing. DeepWalk [Perozzi et al. 2014] deploys a truncated random walk and then learns node embeddings by skip-gram [Mikolov et al. 2013]. Node2Vec [Grover and Leskovec 2016] designs a biased-random walk that provides a trade-off between breadth-first and depth-first graph searches, and hence produces more informative embeddings than DeepWalk. GraphSAGE [Hamilton et al. 2017] proposes a framework that learns inductive representation by sampling and aggregating features from a node's local neighborhood. Text-associated DeepWalk (TADW) [Yang et al. 2015] is a network embedding method for representation learning of text-attributed networks. Large-scale information network embedding (LINE) [Tang et al. 2015] defines the first-order proximity and second-order proximity and minimize the KL divergence between distributions of adjacency matrix and embedding. It can be effectively applied to directed and weight networks in linear time complexity.

The growing research on deep learning has led to deep neural networks based methods applied to graphs. Structural deep network embedding (SDNE) [Wang et al. 2016] proposes using deep autoencoders to preserve local network proximities. Deep neural networks for graph representations (DNGR) [Cao et al. 2016] combines random surfing with deep autoencoder to learn representations for nodes. Graph convolutional networks (GCN) [Kipf and Welling 2016a] proposes a semi-supervised framework which defined a convolution operation on graph to iteratively aggregate the embedding of neighbors for a node. Graph auto-encoders (GAE) [Kipf and Welling 2016b] is a kind of variational graph autoencoder that replaces linear layer with graph convolutional layer, which is designed for link prediction task. However, most of the above works are not intended for node clustering task.

Network Embedding for Clustering Methods. Clustering is a fundamental task in data mining. In recent years, many research studies use deep-learning methods to improve clustering results. The deep clustering methods can be classified into four categories according to the network architectrue [Min et al. 2018]. Deep Embedded Clustering (DEC)[Xie et al. 2016] is a representative algorithm. It pre-trains an autoencoder based on reconstruction loss, then uses clusering loss to fine-tune the network. Information Maximizeing Self-Augmented Training (IMSAT) [Hu et al. 2017b] is an unsupervised algorithm to obtain a function which can get discrete representations of data and it can be used for clustering task. Joint Unsupervised Learning (JULE) [Yang et al. 2016] utilizes convolutional neural network to learn representation and a hierarchical clustering to cluster. Deep Adaptive Image Clustering (DAC) [Chang et al. 2017] is an algorithm based on a single-stage convolutional network for images clustering. [Tian et al. 2014] directly applies a sparse autoencoder on the adjacency matrix to learn node embeddings for clustering task. Similarly, [Yang et al. 2016] adapts stacked autoencoder on modularity matrix to learn the representation of nodes for community detection. But the effectiveness of their low-dimensional embedding is limited because it only learns the second-order proximity. Furthermore, the autoencoder-based model is hard to be extended to large-scale data sets because the sparsity of adjacency matrix and the complexity of neural network.

Recently, Cavallari et al. [2017] introduce the concept of community embedding and try to close the loop of community detection, community embedding and node embedding. Based on LINE [Tang et al. 2015] and Gaussian Mixture Model, their framework manages to combine node embedding and clustering together by iteratively optimizing first-order loss, second-order loss, higher-order loss, and clustering loss. However, the Gaussian Mixture Model can't be optimized together with the network embedding procedure because it needs the embedding to fit the model. So the framework has to conduct these two procedures separately hence the embedding learned may not be optimal. In contrast, NEC can conduct node embedding and clustering procedure simultaneously by forming a united objective function.

3 MODEL DESCRIPTION

Given a graph $G = (V, E, X)$, where V is the set of nodes, E is the set of edges between nodes. A denotes the adjacency matrix of graph G. N denotes the the number of nodes, aka. $|V|$. $X \in \mathbb{R}^{N \times D}$ is the node attribute matrix, and D is the number of attributes. The problem of network embedding aims to represent each node $v \in V$ into a low-dimensional space $\mathbb{R}^h$, i.e., learning a function $f_G : V \rightarrow \mathbb{R}^h$, where h is usually far less than $|V|$. We denote the embedding of each node v_i as $\vec{z}_i$. In the space of $\mathbb{R}^h$, the network information and cluster information must be preserved. In this article, we consider the network embedding for clustering task[1]. Suppose there are K communities in graph G. For each node v_i, we denote its community assignment as $s_i \in 1, \ldots, K$. The notations in this article is summarized in Table 1. Given node embedding, a straightforward way to uncover communities is to run a clustering algorithm, such as k-means or Gaussian Mixture Model. However, such an approach lacks a unified objective function, which makes the optimization difficult. We present the NEC model to find communities in a graph by a unified objective function and optimize cluster assignment in a self-learning manner.

Our model consists of the following three parts: graph convolutional autoencoder, modularity maximization module, and clustering module. The architecture of NEC is shown in Figure 1. In

[1]We simply consider node clustering in graph and community detection as the same task, which means we try to find communities in graphs by an embedding-then-clustering way.

Table 1. Notations

symbols	Descriptions
A	graph adjacency matrix
X	graph attribute matrix
N	number of nodes in graph
Z	representations of nodes
h	hidden dimensions
A'	reconstructed graph adjacency matrix
F	auxiliary matrix
K	number of communities in graph
H	temporary community assignment matrix
Q	assignment distribution matrix
P	auxiliary target distribution

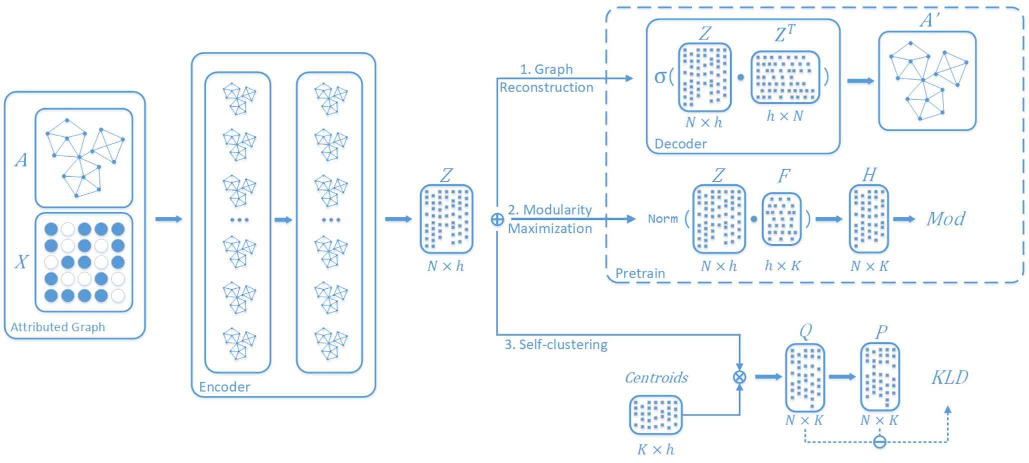

Fig. 1. Architecture of NEC. Left part is the graph convolutional encoder that learns node representation from graph structure and node attributes. Right part consists of three modules. Right-top is graph convolutional decoder that reconstructs graph structure. Right-middle is modularity maximization module that preserves the community structure. Right-bottom is the self-clustering module that learns cluster-oriented embeddings together with community assignments.

the following part of this section, we will introduce the three parts respectively and then combine them together to form the NEC model. Afterwards, we inference the details of how to optimize the model. Lastly, we reveal the pseudocode of the whole model and conduct the complexity analysis.

3.1 Graph Convolutional Autoencoder

The graph convolutional autoencoder aims to embed an attributed graph G into a low-dimensional space. To form an convolutional graph autoencoder, we need to integrate both graph structure and node content in the encoder and choose proper information to be reconstructed via the decoder.

Encoder model: To represent both topological information and node attribute, we choose multi-layer graph convolutional network [Kipf and Welling 2016a]. Graph convolutional network learns a layer-wise transformation by a spectral convolution function $f(Z^{(l)}, A|W^{(l)})$:

$$Z^{(l+1)} = f_\phi(Z^{(l)}, A|W^{(l)}) \tag{1}$$

where $Z^{(l)}$ and $Z^{(l+1)}$ are input and output of convolution. We have $Z^0 = X$. W is a weight matrix that we need to learn. The convolutional network is defined as follows:

$$f_\phi(Z^{(l)}, A|W^{(l)}) = \phi(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-1/2}Z^{(l)}W^{(l)}) \tag{2}$$

where $\tilde{A} = A + I$ and $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$. I is identity matrix with the same size as A. ϕ is an activation function, such as $\text{sigmoid}(t) = \frac{1}{1+e^t}$, $\text{relu}(t) = \max(0, t)$ or $\text{linear}(t) = t$.

In our article, the encoder is constructed with a two-layer GCN as follows:

$$Z^{(1)} = f_{\text{relu}}(X, A|W^{(0)}) \tag{3}$$
$$Z^{(2)} = f_{\text{linear}}(Z^{(1)}, A|W^{(1)}) \tag{4}$$

Note that the activation function of the first and the second layer are relu(·) and linear(·), respectively. The graph encoder can be denoted as $f(Z, A) = \Phi(Z|X, A)$, it encodes both structure information and node attributes into a representation $Z = \Phi(Z|X, A) = Z^{(2)}$.

Decoder Model: The graph decoder aims to reconstruct the graph data. Both graph structure and node attribute can be reconstructed. Here we choose to reconstruct only graph adjacency matrix A because it makes sure that our method works even if there are no node attributes to learn(In that case, we simply let $X = I$). When we say reconstructing the adjacency matrix, it actually means to predict the edges between nodes. Specifically, let the reconstructed adjacency matrix be A', the decoder is defined as an inner product between representation of nodes:

$$p(A'|Z) = \prod_{i=1}^{N}\prod_{j=1}^{N} p(A'_{ij}|\vec{z}_i, \vec{z}_j), \text{with } p(A'_{ij} = 1|\vec{z}_i, \vec{z}_j) = \sigma(\vec{z}_i^T \vec{z}_j) \tag{5}$$

where A'_{ij} are elements of A', $\sigma(\cdot)$ is sigmoid function.

In summary, for the graph convolutional autoencoder, the node embedding and the reconstructed graph can be presented as:

$$Z = f_{\text{linear}}(f_{\text{relu}}(X, A|W^{(0)}), A|W^{(1)}) \tag{6}$$
$$A' = \text{sigmoid}(ZZ^T) \tag{7}$$

The loss of graph autoencoder is defined as the reconstruction error between A and A':

$$L_{\text{gae}} = \mathbb{E}_{\Phi(Z|X,A)}[\log p(A'|Z)] \tag{8}$$

Specifically, L is the cross entropy loss:

$$L_{\text{gae}} = -\sum_i \sum_j A_{ij} \log(A'_{ij}) \tag{9}$$

The neural network wights $W^{(0)}$ and $W^{(1)}$ are trained using gradient descent. In this work, we perform batch gradient descent using the full graph for every training iteration. To reduce memory requirement, we use sparse representation of A. So the memory requirement is $O(|E|)$.

3.2 Modularity Maximization

To preserve the higher-order proximity of network, we design a relaxed modularity maximization approach. Modularity is defined as the difference between the number of edges with communities and the expected number of such edges over all pairs of nodes [Newman 2006]. The modularity of graph G is defined as:

$$\text{Mod} = \frac{1}{4m}\text{Tr}(H^T BH) \tag{10}$$

where $B \in \mathbb{R}^{N\times N}$ is the so-called modularity matrix, which has elements $B_{ij} = A_{ij} - \frac{d_i d_j}{2m}$. $\mathrm{Tr}(\cdot)$ is the trace of a matrix. A is the adjacency matrix. d_i is the degree of node v_i, and $m = \frac{1}{2}\sum_i k_i$ is the number of edges. $H \in \mathbb{R}^{N\times K}$ denotes the community assignment matrix.

$$H_{ik} = \begin{cases} 1 & \text{if node } i \text{ belongs to community } k \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

Notice that the original definition of modularity requires hard assignment of community label of node. i.e., H_{ik} equals 1 if node v_i belongs to community k and zero otherwise. The modularity can be relaxed and allow soft assignment of nodes to communities, aka. replace hard community assignment with $Tr(H^T H) = N$ [Yang et al. 2016]. It's easy to get that $Tr(H^T H) = N$ is equivalent to $\sum_i \sum_k H_{ij}^2 = N$. To meet the constraint, we can normalize H as follows:

$$H_{ik} = \frac{\sqrt{N} \cdot \sqrt{H_{ik}}}{\sum_i \sum_k \sqrt{H_{ik}}} \tag{12}$$

If we set the embedding dimension to be K, then the representation matrix Z can be regraded as the community assignment matrix, i.e., $H = Z$. We can get node embedding by optimizing the modularity. However, the number of communities is not always suitable for embedding. For example, the Citeseer dataset, which has more than 3,200 nodes and 4,632 edges, only has six distinct labels, the representation power of node embedding will be very weak if we set the embedding dimension to be 5. To address this issue, we introduce an auxiliary full connected embedding layer. Suppose we want to embed the nodes to an h dimension space. Then we need a full connected layer $F \in \mathbb{R}^{h\times K}$ to map the embedding to community assignment. The eventually relaxed modularity is defined as:

$$L_{\mathrm{Mod}} = \frac{1}{4m}\mathrm{Tr}(H^T BH) \text{ with } H = norm(ZF) \tag{13}$$

where *norm* is the normalization procedure of (12).

The relaxed modularity (Equation (13)) can be optimized using gradient descent algorithm. The gradients of L_{Mod} with respect to H are computed as:

$$\frac{\partial L_{\mathrm{Mod}}}{\partial H} = \frac{1}{2m} H^T B \tag{14}$$

3.3 Clustering

Most previous network embedding methods are not designed for clustering tasks. To learn a clustering-specific node representation and ensure the node representation in low-dimension space falling in well-organized clusters, a reasonable choice is to jointly optimize network embedding and clustering in one objective. In the work of DEC [Xie et al. 2016; Guo et al. 2017a], they propose a model that clusters data by simultaneously learning a set of K cluster centers in the embedding space and the weights of DNN that maps data points into low-dimensional representations. The clustering loss is defined as KL divergence between the soft assignment distributions Q and the auxiliary distribution P as follows:

$$L_{\mathrm{cluster}} = KL(P||Q) = \sum_i \sum_k p_{ik} \log \frac{p_{ik}}{q_{ik}} \tag{15}$$

where q_{ik} is the probability of assigning node v_i to cluster k measured by Student's t-distribution:

$$q_{ik} = \frac{(1 + \|\vec{z}_i - \mu_k\|^2/\alpha)^{-\frac{\alpha+1}{2}}}{\sum_{k'}(1 + \|\vec{z}_i - \mu_{k'}\|^2/\alpha)^{-\frac{\alpha+1}{2}}} \tag{16}$$

where α is the degree of freedom of the Student's t-distribution. We set $\alpha = 1$ for our experiments. p_{ik} is the auxiliary target distribution defined as:

$$p_{ik} = \frac{q_{ik}^2 / \sum_i q_{ik}}{\sum_{k'} (q_{ik'}^2 / \sum_i q_{ik})} \tag{17}$$

The clustering loss works by using the samples with high confidence as supervision to help improving the samples with low confidence and to make samples in each cluster distribute more densely.

The clustering loss (Equation (15)) can be optimized using gradient descent algorithm. During backpropagation, the gradients of L_{cluster} with respect to each node embedding $\vec{z}_i$ and each cluster centroid μ_k are computed as:

$$\frac{\partial L_{\text{cluster}}}{\partial \vec{z}_i} = \frac{\alpha + 1}{\alpha} \sum_{k=1}^{K} \left(1 + \frac{||\vec{z}_i - \mu_k||^2}{\alpha}\right)^{-1} (p_{ik} - q_{ik})(\vec{z}_i - \mu_k) \tag{18}$$

$$\frac{\partial L_{\text{cluster}}}{\partial \mu_k} = \frac{\alpha + 1}{\alpha} \sum_{i=1}^{N} \left(1 + \frac{||\vec{z}_i - \mu_k||^2}{\alpha}\right)^{-1} (q_{ik} - p_{ik})(\vec{z}_i - \mu_k) \tag{19}$$

Then the gradients are passed down to update centroids μ and graph encoder weights.

3.4 Model Training

In order to learn network embedding for node clustering task, we design a unified objective function to optimize node embedding and node assignment simultaneously. In this way, it can be guaranteed to get good representations as well as certified cluster assignments of nodes. Otherwise, if these two parts are carried out separately, i.e., training node embeddings first and then clustering node to different clusters and feed back to node embedding lastly, the result will be not stable and no meaningful embedding will be guaranteed to be learned. The clustering loss (Equation (15)) is a good choice that can be integrated with the embedding loss because this strategy tries to cluster embeddings in a self-learning way [Nigam and Ghani 2000] and does not need to train clusters from raw data.

The proposed unified objective function is:

$$L_{\text{train}} = L_{\text{gae}} - \beta L_{\text{Mod}} + \gamma L_{\text{cluster}} \tag{20}$$

where β and γ are hyper-parameters that trade off the weights of different losses. L_{gae}, L_{Mod}, and L_{cluster} denote losses of graph convolutional autoencoder, modularity maximization module, and clustering module, respectively, shown in Equations (8), (13), and (15).

We jointly optimize the node embedding and cluster centers using Stochastic Gradient Descent (SGD). The graph encoder weights $W^{(0)}$, $W^{(1)}$ and cluster centroids μ_k are updated by:

$$W^{(0)} = W^{(0)} - \frac{\lambda}{N} \sum_{i=1}^{N} \left(\frac{\partial L_{\text{gae}}}{\partial W^{(0)}} - \beta \frac{L_{\text{mod}}}{\partial W^{(0)}} + \gamma \frac{\partial L_{\text{cluster}}}{\partial W^{(0)}} \right) \tag{21}$$

$$W^{(1)} = W^{(1)} - \frac{\lambda}{N} \sum_{i=1}^{N} \left(\frac{\partial L_{\text{gae}}}{\partial W^{(1)}} - \beta \frac{L_{\text{mod}}}{\partial W^{(1)}} + \gamma \frac{\partial L_{\text{cluster}}}{\partial W^{(1)}} \right) \tag{22}$$

$$\mu_k = \mu_k - \frac{\lambda}{b} \frac{\partial L}{\partial \mu_k} \tag{23}$$

Pretraining: Note that in Equation (16), we need cluster centers μ to compute the soft assignment distribution Q that measures similarity between embedding and centroids. To make the algorithm converge faster, a pretraining phase is needed to get initial centroids before training the clustering oriented embeddings. In pretraining phase, we optimize the following objective:

$$L_{\text{pretrain}} = L_{\text{gae}} - \beta L_{\text{Mod}} \tag{24}$$

After pretraining, all embeddings can be extracted from the output of graph encoder. Then k-means algorithm is employed on node representation Z to obtain K initial clustering centroids μ.

In the pretraining phase, the graph encoder's weights are updated by:

$$W^{(0)} = W^{(0)} - \frac{\lambda}{N} \sum_{i=1}^{N} \left(\frac{\partial L_{\text{gae}}}{\partial W^{(0)}} - \beta \frac{L_{\text{mod}}}{\partial W^{(0)}} \right) \tag{25}$$

$$W^{(1)} = W^{(1)} - \frac{\lambda}{N} \sum_{i=1}^{N} \left(\frac{\partial L_{\text{gae}}}{\partial W^{(1)}} - \gamma \frac{L_{\text{mod}}}{\partial W^{(1)}} \right) \tag{26}$$

ALGORITHM 1: NEC

```
Input: graph $G = (V, E)$; number of community $K$; embedding dimension $h$; parameters $\beta, \gamma$; stopping
       threshold $\tau$; max number of epochs for pretraining $T_1$; max number of epochs for training $T_2$; target
       distribtion update interval $T$.
Output: node embedding $Z$; cluster centers $\mu$ and node labels $s$.
for $epoch = 1 : T_1$ do
    update $W^{(0)}, W^{(1)}$ according to (25) and (26).
end
apply $k$-means on $Z$ to get initial cluster centroids $\mu_0$;
for $epoch = 1 : T_2$ do
    if $epoch\%T == 0$ then
        update $P$ using (17);
        compute $s$ by (27);
        if labels changing rate $< \tau$ then
            exit.
        end
    end
    for $A$ and $X$ do
        update $W^{(0)}, W^{(1)}, \mu$ according to (21), (22) and (23);
    end
end
```

Early stopping **:** Notice that the target distribution P serves as "supervised" soft label but depends on Q. When Q varies with the updating of node embedding, P also needs to be updated. However, because of the stochasticity of SGD, P should not be updated too frequently. We update it every T epochs. See Equations (16) and (17) for the update rules. While updating P, each node v_i can get its label by:

$$s_i = \arg\max_k q_{ik} \tag{27}$$

The algorithm will stop training if the changed label assignment percentage between two consecutive epoches is less than a threshold τ.

We summarize NEC in *Algorithm* 1. In lines 1–2, the algorithm optimizes reconstruction loss and modularity loss to get the pretrained node embedding Z. In line 3, the algorithm performs

k-means on Z to get initial cluster centroids. In lines 4–11, the algorithm minimizes the unified objective function to get the ultimate node embeddings and node cluster assignments.

4 EXPERIMENTS

In this section, we assess the effectiveness of our NEC algorithm through experiments. We compare NEC with several baselines on community detection task. These baselines consist of two different sets of algorithms, i.e., network embedding-based algorithms and traditional community detection algorithms. Traditional community detection methods directly find communities based on graph topology and other information while network embedding based methods form clusters through node embeddings. Firstly, we compare NEC with several baselines on community detection task. Then, we evaluate the results of node classification. Finally, we study the parameter sensitivity of our model.

4.1 Experiment Setup

Compared Algorithms: We compare our algorithm with several existing network embedding-based methods. Note that network embedding algorithm only learns the representation of all nodes. To get the community assignments, clustering algorithms like k-means will be applied on these representation vectors.

— *Node2Vec* [Grover and Leskovec 2016]: Node2Vec is an improvement of DeepWalk. It designs a walk strategy that achieves a tradeoff between bread-first-search and depth-first-search to exploit the homophily and structural roles of nodes.
— *ComE* [Cavallari et al. 2017]: ComE learns node embedding by iteratively optimizing first-order, second-order and higher-order loss.
— *TADW* [Yang et al. 2015]: TADW extends DeepWalk to text-associated networks. It incorporates text features of nodes into network representation learning under the framework of matrix factorization.
— *GAE* [Kipf and Welling 2016b]: GAE is autoencoder-based unsupervised framework for network data, which naturally leverage both topological and content information.
— *VGAE* [Kipf and Welling 2016b]: VGAE is a variational version of GAE.

We also compare our algorithm with some traditional community detection methods.

— *Louvain* [Blondel et al. 2008]: Louvain method is a greedy optimization method that attempts to optimize the modularity of a partion of the graph. By iteratively repeat the locally modularity maximization step and the aggregating step, it attains a hierarchy of communities.
— *Infomap* [Rosvall and Bergstrom 2008]: Infomap is a flow-based method that optimizes the map equation, which exploits the information-theoretic duality between the problem of compressing data, and the problem of detecting and extracting significant patterns or structures with those data.
— *BAGC* [Xu et al. 2012]: BAGC is a model-based approach to attributed graph clustering. It is an Bayesian probabilistic model that captures both structural and attribute aspects of networks.
— *PICS* [Akoglu et al. 2012]: PICS is a parameter-free community detection method for attributed graphs. It requires no user-specified parameters such as the number of clusters and similarity functions.
— *PAICAN* [Bojchevski and Günnemann 2018]: PAICAN is a probabilistic model that aims to use the graph structure and node attributes to group similar nodes together. The model

derives meaningful clusters based on the clean source, aka. clean graph from Degree-Corrected Stochastic Block Model and clean binary attributes from Bernoulli Mixture Model.

Among these algorithms, Node2Vec, ComE, Louvain, and Infomap are for plain networks (no attributes). TADW, GAE, VGAE, BAGC, PICS, and PAICAN are designed for attributed networks.

Evaluation Metrics: To assess the quality of the community detection results, we report three supervised metrics, i.e., the *Normalized Mutual Information* (NMI) score, *Accuracy* [Cai et al. 2011], and Adjusted Rand Index (ARI), and two unsupervised metric, i.e., Modularity [Newman 2006] and Attribute Entropy [Sun et al. 2018].

NMI. Given cluster label C in dataset and predicted cluster label $\bar{C}$, the NMI is defined as:

$$NMI(C,\bar{C}) = \frac{2 \cdot MI(C,\bar{C})}{[H(C) + H(\bar{C})]} \tag{28}$$

where $H(C) = -\sum_{i=1}^{|C|} P(i)\log(P(i))$ is entropy, $P(i) = |C_i|/N$ is the probability that an object picked at random from C falls into class C_i, and $MI(C,\bar{C}) = \sum_{i=1}^{|C|}\sum_{j=1}^{|\bar{C}|} \frac{|C_i \cap \bar{C}_j|}{N} \log(\frac{N|C_i \cap \bar{C}_j|}{|C||\bar{C}|})$ is mutual information between C and $\bar{C}$.

Accuracy. The clustering accuracy is defined as:

$$AC(C,\bar{C}) = \frac{\sum_{i=1}^{N} \delta(C_i, map(\bar{C}_i))}{N} \tag{29}$$

where $\delta(x,y)$ is the Kronecker function that equals 1 if $x = y$ and equals 0 otherwise, and $map(\cdot)$ is the permutation function that maps each cluster labels $\bar{C}_i$ to the equivalent label from the dataset. The best mapping can be found by using the Kuhn–Munkres algorithm [Plummer and Lovász 1986].

ARI. The ARI between two cluster partition C and $\bar{C}$ is defined as:

$$ARI(C,\bar{C}) = \frac{\binom{n}{2}(a+d) - [(a+b)(a+c) + (c+d)(b+d)]}{\binom{n}{2}^2 - [(a+b)(a+c) + (c+d)(b+d)]} \tag{30}$$

where a, b, c, and d are defined as:

$$a = \sum_{vw} \delta(C^v, C^w)\delta(\bar{C}^v, \bar{C}^w)$$
$$b = \sum_{vw} (1 - \delta(C^v, C^w))\delta(\bar{C}^v, \bar{C}^w)$$
$$c = \sum_{vw} \delta(C^v, C^w)(1 - \delta(\bar{C}^v, \bar{C}^w))$$
$$d = \sum_{vw} (1 - \delta(C^v, C^w))(1 - \delta(\bar{C}^v, \bar{C}^w))$$

Modularity. The modularity of the resulting communities can be calculated as follows:

$$Mod(C) = \frac{1}{2m}\sum_{ij}\left(A_{ij} - \frac{d_i \cdot d_j}{2m}\right)\delta(c_i, c_j) \tag{31}$$

where c_i is the community assignment of node i, and the other settings are the same as Equation (10).

Table 2. Synthetic Datasets

dataset	#nodes	#edges	#features	mixing parameter	#communities
synthetic_1000_0.50	1,000	6,970	100	0.50	12
synthetic_1000_0.55	1,000	7,276	100	0.55	12
synthetic_1000_0.60	1,000	7,076	100	0.60	14
synthetic_2000_0.50	2,000	14,085	200	0.50	14
synthetic_2000_0.55	2,000	15,006	200	0.55	14
synehttic_2000_0.60	2,000	14,089	200	0.60	13

Table 3. Larger Synthetic Datasets

dataset	#nodes	#edges	#features	mixing parameter	#communities
synthetic_10000_0.50	10,000	66,590	100	0.50	8
synthetic_10000_0.60	10,000	66,320	100	0.60	8
synthetic_2000_0.50	20,000	133,492	100	0.50	16
synehttic_2000_0.60	20,000	133,573	100	0.60	16

Table 4. Real-world Datasets

dataset	#nodes	#edges	#features	#communities
Hvr	307	6,526	6	2
Cora	2,708	5,278	1,433	7
Citeseer	3,264	4,532	3,703	6

Datasets: Since all compared attributed community detection algorithms support categorical attributes or binary attributes, and categorical attributes can be converted to binary attributes, we report the results on binary attributed graphs for all algorithms. Results on both synthetic datasets and real-world datasets are evaluated.

We generate several synthetic networks with binary node attributes, as listed in Table 2. These datasets are generated by the following approach. Firstly, a plain network with ground-truth community assignment is generated using Andrea Lancichinetti, Santo Fortunato, and Filippo Radicchi (LFR) benchmark [Lancichinetti et al. 2008]. Then binary attributes are generated for each node according to the ground-truth label. The same attribute vectors are assigned to nodes in the same community. In this way, nodes can be perfectly divided into ideal clusters in terms of attributes. We also add noise to the attributes, which can be achieved by randomly flip attributes of each node and each dimension with a small probability.

Among the many parameters of LFR benchmark, mixing parameter is the most important one because the difficulty of discovering communities in graph depends on it. The bigger it is, the harder to identify communities. For datasets listed in Table 2, the last underscore separated number is the corresponding mixing parameter. For example, the mixing parameter of synthetic_1000_0.5 is 0.5.

In order to verify the scalability of our method, we also generated several larger synthetic networks using the same method. Details of these datasets are listed in Table 3.

We use three public networks datasets, Hvr, Cora, and Citeseer, as listed in Table 4. The details of these datasets are introduced as follows:

- *Hvr*[2]: The Hvr dataset is a gene network consisting of several networks of highly recombinant malaria parasite genes. The ground-truth clusters are decided according to the Cys-PoLV (CP) labels.
- *Cora*[3]: The Cora dataset consists of 2,708 machine-learning papers classified into one of the seven research areas. The citation network consists of 5,429 links. Each publication in the dataset is described by a 0/1-valued word vector indicating the absence/presence of the corresponding word from the dictionary. After stemming and removing stopwords, the dictionary consists of 1,433 unique words. Each paper has a label indicating the research area it belongs to.
- *Citeseer* [4]: The Citeseer dataset consists of 3,312 scientific publications classified into one of six research areas. The citation network consists of 4,732 links. Each publication in the dataset is described by a 0/1-valued word vector indicating the absence/presence of the corresponding word from the dictionary. The dictionary consists of 3,703 unique words. Similar with *Cora*, the label of each paper represents the research area the paper belongs to.

Parameter Settings: In the experiment of NEC, we set $h = 32, \tau = 0.001, \alpha = 1.0, \beta = 0.1, \gamma = 0.1$, and $T = 1$ for all datasets. For synthetic datasets, we set $T_1 = 350, T_2 = 400$. For Hvr, Citeseer, and Cora datasets, we set $T_1 = 200, T_2 = 100$. We implement our model using Pytorch [Paszke et al. 2017] library. The parameters are optimized by Adam [Kingma and Ba 2014] algorithm with an initial learning rate of 0.001. We set K as the number of unique labels in each dataset. Thirty times of k-means are performed on the resulting node representations and the mean result is reported.

In Node2Vec and ComE, the number of walks for every node is set to 30 and walk length is set to 10, and the window size is set to 5. We set $p = q = 0.25$ for Node2Vec according to [Grover and Leskovec 2016]. In GAE and VGAE, the dimension of hidden layer is set to 32, weight_decay and dropout are set to 0. For fair comparisons, embedding dimensions is set to the same number as that of NEC. For other algorithms, we use the default parameter suggested by the author.

All results are obtained on Linux machines with 4 Intel(R) Core(TM) i7-6700 CPUs with 32 GB memory and NVIDIA Quadro K1200 GPU with 4 GB memory.

4.2 Experiment on Synthetic Datasets

In this subsection, we evaluate the performance of our algorithm on synthetic datasets. We compare NEC with network embedding-based methods and traditional community detection methods, respectively.

4.2.1 Compared with Network Embedding Based Methods. Since nodes of all datasets have single label, we only assign one community label to each node. For all network embedding-based methods, we apply k-means on their node embeddings to predict communities. We evaluate the performance of community detection by NMI, $Accuracy$, ARI, and $Modularity$. We also carry out experiments to verify the effect of the modularity maximization module and the clustering module. NEC-Mod represents the model which removes the clustering module from loss function, and NEC-Clu represents the loss function 20 without modularity maximization module. Due to the sensitivity of k-means to the initial values, all of the results are averaged over 30 different runs, and shown in Tables 5–8.

As we can see, NEC achieves the best results on all datasets. NEC achieves 1.9%~8.4%, 0.8%~5.5%, and 2.1%~12.3% improvement compared to the second best results in terms of NMI, ACC, and

[2]Available at https://github.com/abojchevski/paican/tree/master/data/.
[3]Available at https://linqs.soe.ucsc.edu/data.
[4]Available at https://linqs.soe.ucsc.edu/data.

Table 5. NMI of Community Detection (Bold Numbers Represent the Best Results)

Dataset	Node2Vec	ComE	TADW	GAE	VGAE	NEC	NEC-Mod	NEC-Clu
Synthetic_1000_0.50	0.9021	0.8932	0.8947	0.9711	0.9724	**0.9912**	0.9890	0.9889
Synthetic_1000_0.55	0.6890	0.8024	0.8236	0.9067	0.9267	**0.9519**	0.9512	0.9410
Synthetic_1000_0.60	0.4160	0.6679	0.5323	0.7701	0.8024	**0.8706**	0.8471	0.8373
Synthetic_2000_0.50	0.8946	0.8254	0.8227	0.9589	0.9620	**0.9892**	0.9844	0.9784
Synthetic_2000_0.55	0.8025	0.8145	0.7574	0.8270	0.8370	**0.8742**	0.8733	0.8655
Synthetic_2000_0.60	0.5550	0.5597	0.4770	0.6105	0.6474	**0.6829**	0.6478	0.6462

Table 6. Accuracy (%) of Community Detection (Bold Numbers Represent the Best Results)

Dataset	Node2Vec	ComE	TADW	GAE	VGAE	NEC	NEC-Mod	NEC-Clu
Synthetic_1000_0.50	0.9500	0.9470	0.9490	0.9860	0.9870	**0.9960**	0.9950	0.9949
Synthetic_1000_0.55	0.7520	0.8940	0.9110	0.9549	0.9655	**0.9783**	0.9775	0.9729
Synthetic_1000_0.60	0.4950	0.7920	0.6780	0.8589	0.8830	**0.9319**	0.9178	0.9119
Synthetic_2000_0.50	0.9485	0.8550	0.9120	0.9800	0.9820	**0.9952**	0.9930	0.9904
Synthetic_2000_0.55	0.8995	0.9060	0.8750	0.9085	0.8545	**0.9320**	0.9313	0.9206
Synthetic_2000_0.60	0.7145	0.7425	0.6265	0.7380	0.7605	**0.7752**	0.7520	0.7528

Table 7. ARI of Community Detection (Bold Numbers Represent the Best Results)

Dataset	Node2Vec	ComE	TADW	GAE	VGAE	NEC	NEC-Mod	NEC-Clu
Synthetic_1000_0.50	0.8942	0.8873	0.8970	0.9704	0.9713	**0.9916**	0.9894	0.9891
Synthetic_1000_0.55	0.5976	0.7721	0.8191	0.9074	0.9249	**0.9511**	0.9506	0.9397
Synthetic_1000_0.60	0.2823	0.6012	0.4609	0.7251	0.7632	**0.8567**	0.8318	0.8208
Synthetic_2000_0.50	0.8896	0.7936	0.8200	0.9567	0.9599	**0.9901**	0.9849	0.9791
Synthetic_2000_0.55	0.7932	0.8051	0.7491	0.8151	0.7874	**0.8747**	0.8662	0.8540
Synthetic_2000_0.60	0.5037	0.5100	0.4116	0.5650	0.6066	**0.6423**	0.5951	0.5951

Table 8. Modularity of Community Detection (Bold Numbers Represent the Best Results)

Dataset	Node2Vec	ComE	TADW	GAE	VGAE	NEC	NEC-Mod	NEC-Clu
Synthetic_1000_0.50	0.4067	0.4045	0.3794	0.4096	0.4102	**0.4112**	0.4111	0.4110
Synthetic_1000_0.55	0.3279	0.3511	0.3121	0.3553	0.3579	**0.3602**	0.3601	0.3597
Synthetic_1000_0.60	0.2550	0.2980	0.1697	0.2964	0.3030	**0.3151**	0.3129	0.3116
Synthetic_2000_0.50	0.4216	0.4014	0.3808	0.4089	0.4094	**0.4264**	0.4262	0.4260
Synthetic_2000_0.55	0.3729	**0.3735**	0.3226	0.3631	0.3609	0.3700	0.3693	0.3680
Synthetic_2000_0.60	0.2970	**0.3019**	0.1634	0.2788	0.2906	0.2982	0.2917	0.2904

ARI, respectively, demonstrating the effectiveness of our model. Specifically, NEC has dramatically increased the *ARI* on Synthetic_1000_0.60 from 0.7632 to 0.8567 compared with VGAE, the second best result. Furthermore, notice that NEC consistently shows better performance than GAE and VGAE, and NEC-Mod and NEC-Clu both perform worse in terms of all metrics. Especially in synthetic_1000_0.6 and synthetic_2000_0.6, where NEC-Mod performs 4.1% and 7.3% worse than NEC in terms of ARI, respectively. In contrast NEC achieves 2.6% in terms of modularity compared to NEC-Mod. These results prove that the modularity maximization module and self-clustering

Table 9. Results of Community Detection Compared to Traditional Community Detection Methods

measure	dataset	Infomap	Louvain	PICS	BAGC	PAICAN	NEC	NEC-Mod	NEC-Clu
NMI	Synthetic_1000_0.50	0.8350	0.8827	0.0171	0.0597	0.9655	**0.9912**	0.9890	0.9889
	Synthetic_1000_0.55	-	0.7777	0.0213	0.0666	0.8601	**0.9519**	0.9512	0.9410
	Synthetic_1000_0.60	-	0.4838	0.0183	0.0528	0.8269	**0.8706**	0.8471	0.8373
	Synthetic_2000_0.50	0.8190	0.8625	0.0138	0.0581	0.9277	**0.9892**	0.9844	0.9784
	Synthetic_2000_0.55	0.1964	0.7387	0.0141	0.0338	0.8509	**0.8742**	0.8733	0.8655
	Synthetic_2000_0.60	0.1305	0.5101	0.0155	0.0238	0.6406	**0.6829**	0.6478	0.6462
ACC	Synthetic_1000_0.50	-	-	-	-	0.9880	**0.9960**	0.9950	0.9949
	Synthetic_1000_0.55	-	-	-	-	0.8840	**0.9783**	0.9775	0.9729
	Synthetic_1000_0.60	-	-	-	-	0.8310	**0.9319**	0.9178	0.9119
	Synthetic_2000_0.50	-	-	-	-	**0.9990**	0.9952	0.9930	0.9904
	Synthetic_2000_0.55	-	-	-	-	0.9200	**0.9320**	0.9313	0.9206
	Synthetic_2000_0.60	-	-	-	-	0.7655	**0.7752**	0.7520	0.7528
ARI	Synthetic_1000_0.50	0.7999	0.8173	0.0027	0.0046	0.9905	**0.9916**	0.9894	0.9891
	Synthetic_1000_0.55	-	0.6637	0.0005	0.0016	0.8787	**0.9511**	0.9506	0.9397
	Synthetic_1000_0.60	-	0.3549	0.0015	0.0011	0.8456	**0.8567**	0.9318	0.8208
	Synthetic_2000_0.50	0.8055	0.7908	0.0002	0.0028	0.9878	**0.9901**	0.9849	0.9791
	Synthetic_2000_0.55	0.0143	0.6093	0.0010	0.0004	**0.8944**	0.8747	0.8662	0.8540
	Synthetic_2000_0.60	0.0005	0.4232	0.0018	0.0018	0.6089	**0.6423**	0.5951	0.5951
Modulairty	Synthetic_1000_0.50	0.3931	0.4002	0.0042	0.0388	0.4109	**0.4112**	0.4111	0.4110
	Synthetic_1000_0.55	-	**0.3608**	0.0020	0.0094	0.3525	0.3602	0.3600	0.3597
	Synthetic_1000_0.60	-	0.2808	0.0093	0.0042	0.3140	**0.3151**	0.3129	0.3116
	Synthetic_2000_0.50	0.4102	0.4176	0.0075	0.0242	0.4261	**0.4264**	0.4262	0.4260
	Synthetic_2000_0.55	0.0776	**0.3727**	0.0010	0.0027	0.3698	0.3700	0.3693	0.3680
	Synthetic_2000_0.60	0.0572	**0.3111**	0.0019	0.0073	0.2836	0.2982	0.2917	0.2904

module can significantly improve the performance of community detection. Besides, we can also get that methods that make use of attribute information perform better than those not, which means that attribute information does contribute to the discovery of communities.

4.2.2 Compared with Traditional Community Detection Methods. We compared the results of NEC with traditional community detection methods, including Infomap, Louvain, PICS, BAGC, and PAICAN, as shown in Table 9. Because Infomap, Louvain, PICS, and BAGC does not require the number of communities as a hyper-parameter, so the number of communities that these algorithms will find is non-deterministic. According to Equation (29), the number of communities in C should equal the number of communities in $\bar{C}$, otherwise it is unable to compute *ACC*. We mark *ACC* of results that number of predicted communities does not equal the number of given communities as "-."

As we can see, NEC achieves best results on most datasets in terms of *NMI*, *ACC*, and *ARI*. Especially, on Synthetic_1000_0.55, NEC achieves 10.7% improvement in *NMI* and *ACC*, 8.2% improvement in *ARI* compared to the second best results, demonstrating the superior performance of our model compared to traditional methods. In terms of *Modularity*, it is notable that Louvain method performs very good. This is because Louvain method directly optimizes modularity. It is interesting to see that network embedding based methods can outperform traditional methods that directly make use of graph structure and attribute information. The results validates the effectiveness of network embeddings based methods on community detection and the superior of NEC algorithm.

Table 10. Results of Running Time Compared to Traditional Community Detection Methods

dataset	Infomap	Louvain	PICS	BAGC	PAICAN	NEC
Synthetic_1000_0.50	0.55	0.030	0.09	0.15	0.07	59.79
Synthetic_1000_0.55	0.67	0.025	0.12	0.13	0.08	66.15
Synthetic_1000_0.60	0.29	0.030	0.10	0.15	0.08	77.94
Synthetic_2000_0.50	1.46	0.041	0.19	0.32	0.14	195.09
Synthetic_2000_0.55	2.00	0.049	0.22	0.35	0.15	181.10
Synthetic_2000_0.60	2.27	0.042	0.19	0.41	0.15	196.12

Table 11. Results of Running Time Compared to Network Embedding Based Methods

dataset	Node2Vec	ComE	TADW	GAE	VGAE	NEC
Synthetic_1000_0.50	5.37	13.23	6.47	71.41	61.45	59.79
Synthetic_1000_0.55	5.22	13.38	6.86	72.45	61.57	66.15
Synthetic_1000_0.60	5.29	13.28	6.62	71.40	61.37	77.94
Synthetic_2000_0.50	12.76	28.25	13.09	69.83	61.65	195.09
Synthetic_2000_0.55	14.26	28.48	15.21	61.67	61.31	181.10
Synthetic_2000_0.60	13.88	28.86	15.33	61.45	61.65	196.12

4.2.3 Running Time Comparation. To assess the efficiency of NEC, we compare the running time of NEC with different methods on small synthetic datasets. Results are also averaged over 30 different runs.

Table 10 shows the running time of NEC, compared with different community detection algorithms on several synthetic datasets. And Table 11 shows the result compared with network embedding based methods. We can see that the running time of NEC is raising with respect to the network size and mixing parameter. Since NEC uses modularity maximization module, which would cost more time when the network is large. Also, the clustering module would consume more time when the community structure is harder to find. It can be seen that traditional community detection methods have better efficiency than network embedding based methods since they only aim to detect communities. However, network embedding-based methods including NEC also generates embedding vectors for nodes. NEC is less efficient than other network embedding based methods. It is because NEC needs to consider the effect of modularity and attribution information when embedding and reconstructing the network, rather than simply consider the structural information. Although NEC consume more time, it also achieves best results on these datasets in terms of other metrics.

4.2.4 Evaluate on Larger Scale Datasets. We also perform experiments on larger scale datasets to verify the scalability of our method. We still evaluate the performance of community detection with respect to *NMI*, *Accuracy*, and *Modulairty*. Since we already illustrated the role of two additional modules, NEC-Mod and NEC-Clu are not evaluated this time. All results are averaged over 30 different runs, and are shown in Tables 12 and 13, separately.

It can be seen from Tables 12 and 13 that the results are similar with that on small synthetic datasets. NEC performs best on most datasets in terms of these four metrics. Although VGAE achieves similar results with NEC, it is worse than NEC in all metrics, which illustrates the effectiveness of maximization module and self-clustering module. Compared with traditional community detection method, NEC constantly performs better on most datasets. However, Louvain gets better result than NEC in terms of *Modularity* on synthetic_20000_0.6. As mentioned before,

Table 12. Results of Community Detection Compared to Traditional Community Detection Methods on Larger Scale Datasets

measure	dataset	Infomap	Louvain	PICS	BAGC	PAICAN	NEC
NMI	Synthetic_10000_0.50	0.4965	0.5146	0.8323	0.6865	0.9002	**0.9333**
	Synthetic_10000_0.60	0.1853	0.1810	0.4239	0.3940	0.6609	**0.7801**
	Synthetic_20000_0.50	0.4824	0.4935	0.7518	0.7714	0.8030	**0.9297**
	Synthetic_20000_0.60	0.1209	0.1284	0.2057	0.2441	0.7592	**0.7743**
ACC	Synthetic_10000_0.50	-	-	-	0.9611	0.9660	**0.9790**
	Synthetic_10000_0.60	0.8004	-	0.7135	-	0.9007	**0.9130**
	Synthetic_20000_0.50	-	-	0.8102	0.9298	0.9553	**0.9636**
	Synthetic_20000_0.60	-	-	-	-	0.8922	**0.9008**
ARI	Synthetic_10000_0.50	0.1224	0.2418	0.9228	0.7833	0.9205	**0.9516**
	Synthetic_10000_0.60	0.7576	0.7465	0.7399	0.7559	0.7736	**0.8090**
	Synthetic_20000_0.50	0.1403	0.3219	0.8975	0.8744	0.9003	**0.9491**
	Synthetic_20000_0.60	0.1476	0.1287	0.7582	0.7451	0.7662	0.7884
Modulairty	Synthetic_10000_0.50	0.3513	0.3608	0.0897	0.0448	0.3687	**0.3715**
	Synthetic_10000_0.60	-	0.3010	0.0574	0.0096	0.2702	**0.2724**
	Synthetic_20000_0.50	-	0.3349	0.1008	0.0389	0.3530	**0.3605**
	Synthetic_20000_0.60	0.0544	**0.2820**	0.0057	0.0071	0.2511	0.2541

Table 13. Results of Community Detection Compared to Network Embedding Based Methods on Larger Scale Datasets

measure	dataset	Node2Vec	ComE	TADW	GAE	VGAE	NEC
NMI	Synthetic_10000_0.50	0.8780	0.8827	0.8785	0.9294	0.9300	**0.9333**
	Synthetic_10000_0.60	0.3723	0.5998	0.6894	0.7758	0.7789	**0.7801**
	Synthetic_20000_0.50	0.8510	0.8034	0.8571	0.9197	0.9209	**0.9297**
	Synthetic_20000_0.60	0.3507	0.6002	0.7004	0.7455	0.7330	**0.7743**
ACC	Synthetic_10000_0.50	0.9205	0.9264	0.9190	0.9672	0.9778	**0.9790**
	Synthetic_10000_0.60	0.4396	0.8051	0.8522	0.9088	0.9101	**0.9130**
	Synthetic_20000_0.50	0.9223	0.9217	0.8843	0.9496	0.9597	**0.9636**
	Synthetic_20000_0.60	0.8382	0.8840	0.8062	0.8580	0.8986	**0.9008**
ARI	Synthetic_10000_0.50	0.8520	0.8626	0.9227	0.9501	0.9490	**0.9516**
	Synthetic_10000_0.60	0.7072	0.7222	0.6976	0.7986	0.7912	**0.8090**
	Synthetic_20000_0.50	0.8574	0.8080	0.8946	0.9260	0.9420	**0.9491**
	Synthetic_20000_0.60	0.7060	0.6862	0.7011	0.7799	0.7831	0.7884
Modulairty	Synthetic_10000_0.50	0.3633	0.3641	0.3318	0.3701	0.3711	**0.3715**
	Synthetic_10000_0.60	0.2540	0.2284	0.1903	0.2698	0.2700	**0.2724**
	Synthetic_20000_0.50	0.3413	0.3386	0.2049	0.3509	0.3553	**0.3605**
	Synthetic_20000_0.60	0.2500	0.2423	0.1698	0.2366	0.2522	0.2541

this is because Louvain optimizes modularity directly. The results show that NEC method can also get better results on larger scale datasets.

4.3 Evaluate on Real-world Dataset

We also evaluate the effectiveness of NEC on real-world datasets. Results of all ten baseline algorithms on Hvr, Cora, and Citeseer datasets, including traditional methods and network embedding-based methods, are shown in Tables 14–16.

Table 14. Results on Hvr Dataset (Bold Numbers Represent the Best Results)

Alg.	NMI	ACC	ARI	MOD
Infomap	0.4090	-	0.1124	0.5179
Louvain	0.4305	-	0.1575	**0.5267**
PICS	0.4487	-	0.1972	0.4701
BAGC	0.0385	0.5230	0.0426	0.3267
PAICAN	0.8182	0.9737	0.8957	0.1601
Node2Vec	0.7242	0.9572	0.8329	0.0809
ComE	0.7538	0.9638	0.8572	0.0833
TADW	0.8125	0.9711	0.8025	0.1193
GAE	0.7977	0.9614	0.8731	0.1594
VGAE	0.8072	0.9704	0.8830	0.1013
NEC	**0.8238**	**0.9737**	**0.8956**	0.1627

Table 15. Results on Cora Dataset (Bold Numbers Represent the Best Results)

Alg.	NMI	ACC	ARI	MOD
Infomap	0.4698	-	0.0534	0.7297
Louvain	0.4698	-	0.1074	**0.7658**
PICS	0.0463	-	0.0500	0.2596
BAGC	0.1445	-	0.0159	0.0022
PAICAN	0.3947	0.5011	0.3075	0.6620
Node2Vec	0.2413	0.4239	0.1344	0.6127
ComE	0.4194	0.6185	0.3383	0.1456
TADW	0.3826	0.5572	0.2712	0.4151
GAE	0.4661	0.6414	0.3810	0.7284
VGAE	0.5058	0.6721	0.4402	0.7259
NEC	**0.5584**	**0.7506**	**0.5304**	0.7336

Table 16. Results on Citeseer Dataset (Bold Numbers Represent the Best Results)

Alg.	NMI	ACC	ARI	MOD
Infomap	**0.3880**	-	0.0252	0.8271
Louvain	0.3794	-	0.0364	**0.8522**
PICS	0.0739	-	0.0177	0.4978
BAGC	0.0000	-	0.0000	0.0000
PAICAN	0.2171	0.4556	0.1950	0.6973
Node2Vec	0.1633	0.3664	0.0741	0.6201
ComE	0.1770	0.3676	0.1028	0.0689
TADW	0.2641	0.4996	0.3206	0.5573
GAE	0.2751	0.5100	0.2145	0.7401
VGAE	0.2864	0.5106	0.2353	0.7557
NEC	0.2783	**0.5731**	**0.2982**	0.7321

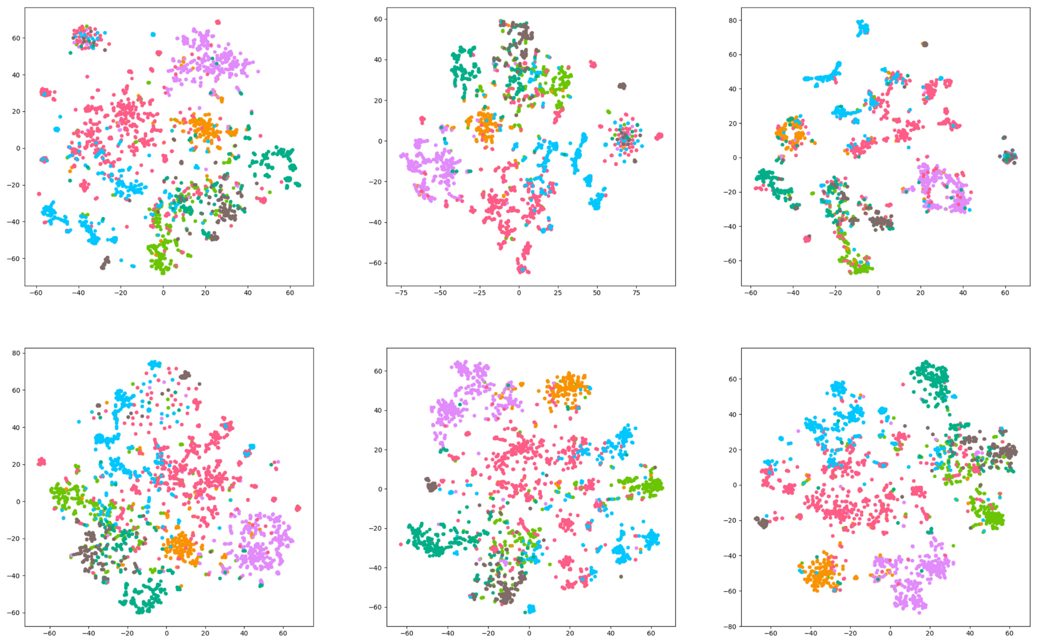

Fig. 2. Visualization of node embeddings on Cora dataset. Different colors represent different communities. From left to right and top to down: embedding from Node2Vec, ComE, TADW, GAE, VGAE, and our NEC.

As we can see, NEC achieves best results on all three datasets on *NMI*, *ACC*, and *ARI* metrics. Especially, on Cora, NEC achieves 5.9% improvement in *NMI*, 9.0% improvement in *ACC*, and 14.4% improvement in *ARI* compared to the second best results. On Citeseer, NEC achieves 12.2% improvement in *ACC* and 26.7% improvement in *ARI*. In terms of *Modularity* metric, Louvain achieves best results on all three datasets because it takes Modularity as the objective and maximizes it directly. Results on these three real-world datasets also validate the effectiveness of NEC compared with traditional methods and network embedding based methods.

4.4 Graph Visualization

We visualize the Cora dataset in a two-dimensional space by applying the t-SNE algorithm [Maaten and Hinton 2008] on the node embeddings of different algorithms, e.g., Node2Vec, ComE, TADW, GAE, VGAE, and NEC. The results are shown in Figure 2. The colors of different points indicate the community labels of corresponding nodes. Compared with other methods, our method makes nodes with the same color (that is, nodes in the same community) are almost clustered together, and that with different color are barely overlapped. Also there does not exist many isolated nodes or small node clusters. Thus we can see that the embeddings results of NEC are much more meaningful. Also these nodes are well-clustered than others, which validates the effectiveness of NEC. Furthermore, compared with GAE algorithm, which has no self-clustering module, the results of NEC are consistently better. It shows that self-clustering does help to improve the performance.

4.5 Parameter Analysis

In this section, we evaluate the effect of parameters β and γ of NEC on the datasets. We only show two datasets (Cora and Citeseer) because the results of different datasets have similar trends. We show the results of NMI and ACC with respect to different β and γ in Figure 3. The default

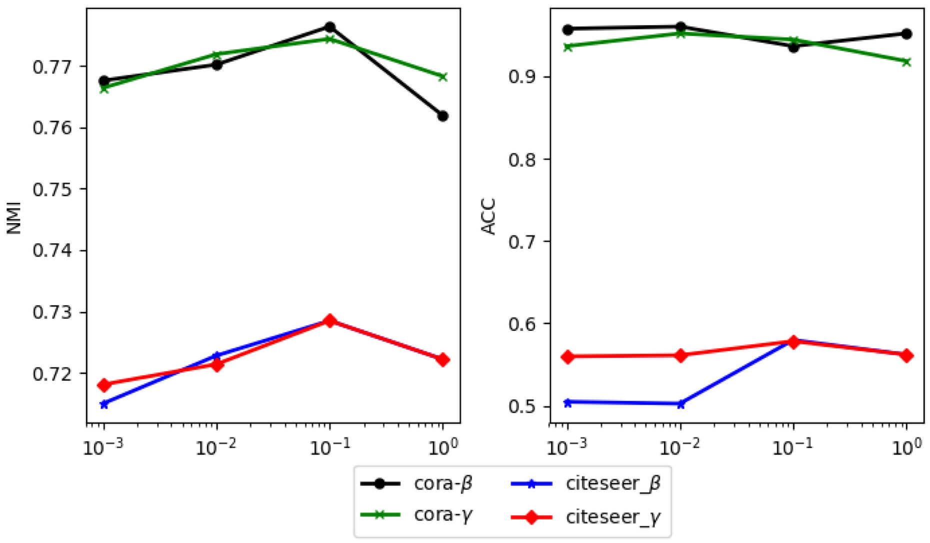

Fig. 3. results of NEC on Cora dataset under different β and γ.

values of β and γ are 0.1. As we can see, the results are quite steady with the variance of β and γ. Generally, the best β is 0.1. This indicates that it is necessary to achieve a trade off between the reconstruction loss, the modularity loss, and the clustering loss. Actually, to prevent the clustering loss from corrupting the embedding space, the coefficient γ should be less than 1. The best value of γ is 0.1.

5 CONCLUSION

We proposed a novel network embedding for clustering model to learn the representations of nodes in attributed graphs. We developed a framework that consist of graph convolutional autoencoder, modularity maximization module, and self-clustering module to embed attributed network structure and apply clustering loss to ensure the embedding is suited for clustering task. We design a unified objective function to combine these losses and optimize it with a two-stage approach. The extensive experiments on community detection demonstrated the superiority of our model. For future work, since computation of modularity and GCN need the whole graph to be loaded in memory, it will be interesting to improve the memory efficiency of our model. Besides, at present our method only considers non-overlapping community detection that one node only belongs to one community. Thus future work may include devising network embedding method for overlapping community detection task and improve its running efficiency.

ACKNOWLEDGMENTS

Any opinions, findings, and conclusions expressed here are those of the authors and do not necessarily reflect the views of the funding agencies.

REFERENCES

Leman Akoglu, Hanghang Tong, Brendan Meeder, and Christos Faloutsos. 2012. Pics: Parameter-free identification of cohesive subgroups in large attributed graphs. In *Proceedings of the 2012 SIAM International Conference on Data Mining*. SIAM, 439–450.

Mikhail Belkin and Partha Niyogi. 2002. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic.* 585–591.

Vandana Bhatia and Rinkle Rani. 2019. A distributed overlapping community detection model for large graphs using autoencoder. *Future Generation Computer Systems* 94 (2019), 16–26.

Christopher M. Bishop. 2006. Machine learning and pattern recognition. In *Information Science and Statistics.* Springer.

Vincent D. Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. 2008. Fast unfolding of communities in large networks. *Journal of statistical mechanics: Theory and experiment* 2008, 10 (2008), P10008.

Aleksandar Bojchevski and Stephan Günnemann. 2018. Bayesian robust attributed graph clustering: Joint learning of partial anomalies and group structure. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence.*

Deng Cai, Xiaofei He, Jiawei Han, and Thomas S. Huang. 2011. Graph regularized nonnegative matrix factorization for data representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33, 8 (2011), 1548–1560.

Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2015. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management.* ACM, 891–900.

Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2016. Deep neural networks for learning graph representations. In *Proceedings of the 13th AAAI Conference on Artificial Intelligence.* 1145–1152.

Sandro Cavallari, Vincent W. Zheng, Hongyun Cai, Kevin Chen-Chuan Chang, and Erik Cambria. 2017. Learning community embedding with community detection and node embedding on graphs. In *Proceedings of the 2017 ACM Conference on Information and Knowledge Management.* ACM, 377–386.

Jianlong Chang, Lingfeng Wang, Gaofeng Meng, Shiming Xiang, and Chunhong Pan. 2017. Deep adaptive image clustering. In *Proceedings of the 2017 IEEE International Conference on Computer Vision.*

Hong Cheng, Yang Zhou, and Jeffrey Xu Yu. 2011. Clustering large attributed graphs: A balance between structural and attribute similarities. *ACM Transactions on Knowledge Discovery from Data* 5, 2 (2011), 12.

J.-J. Daudin, Franck Picard, and Stéphane Robin. 2008. A mixture model for random graphs. *Statistics and Computing* 18, 2 (2008), 173–183.

Jin Di, Ge Meng, Zhixuan Li, Wenhuan Lu, and Francoise Fogelman-Soulie. 2017. Using deep learning for community discovery in social networks. In *Proceedings of the IEEE 29th International Conference on Tools with Artificial Intelligence.*

Santo Fortunato. 2010. Community detection in graphs. *Physics Reports* 486, 3–5 (2010), 75–174.

Anna Goldenberg, Alice X. Zheng, Stephen E. Fienberg, Edoardo M. Airoldi, et al. 2010. A survey of statistical network models. *Foundations and Trends® in Machine Learning* 2, 2 (2010), 129–233.

Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* ACM, 855–864.

Ting Guo, Jia Wu, Xingquan Zhu, and Chengqi Zhang. 2017b. Combining structured node content and topology information for networked graph clustering. *ACM Trans. Knowl. Discov. Data* 11, 3, Article 29 (March 2017), 29. DOI: https://doi.org/10.1145/2996197

Xifeng Guo, Long Gao, Xinwang Liu, and Jianping Yin. 2017a. Improved deep embedded clustering with local structure preservation. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence.* 1753–1759.

Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems.* 1025–1035.

Dongxiao He, Dayou Liu, Di Jin, and Weixiong Zhang. 2015. A stochastic model for detecting heterogeneous link communities in complex networks. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence.*

Paul W. Holland, Kathryn Blackmond Laskey, and Samuel Leinhardt. 1983. Stochastic blockmodels: First steps. *Social Networks* 5, 2 (1983), 109–137.

Pengwei Hu, Keith Chan, and Tiantian He. 2017a. Deep graph clustering in social network. In *Proceedings of the 26th International Conference on World Wide Web Companion.* 1425–1426. DOI: https://doi.org/10.1145/3041021.3051158

Weihua Hu, Takeru Miyato, Seiya Tokui, Eiichi Matsumoto, and Masashi Sugiyama. 2017b. Learning discrete representations via information maximizing self-augmented training. In *Proceedings of the 34th International Conference on Machine Learning.*

Di Jin, Zheng Chen, Dongxiao He, and Weixiong Zhang. 2015. Modeling with node degree preservation can accurately find communities. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence.*

Brian Karrer and Mark E. J. Newman. 2011. Stochastic blockmodels and community structure in networks. *Physical Review E* 83, 1 (2011), 016107.

Abeer Khan, Lukasz Golab, Mehdi Kargar, Jaroslaw Szlichta, and Morteza Zihayat. 2020. Compact group discovery in attributed graphs and social networks. *Information Processing and Management* 57, 2 (2020), 102054. DOI: https://doi.org/10.1016/j.ipm.2019.102054

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. In *Procedings of the 3rd International Conference on Learning Representations (ICLR'15)*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1412.6980

Thomas N. Kipf and Max Welling. 2016a. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

Thomas N. Kipf and Max Welling. 2016b. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308* (2016).

Andrea Lancichinetti, Santo Fortunato, and Filippo Radicchi. 2008. Benchmark graphs for testing community detection algorithms. *Physical Review E* 78, 4 (2008), 046110.

Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of Machine Learning Research* 9, Nov (2008), 2579–2605.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

Erxue Min, Guo Xifeng, Liu Qiang, Zhang Gen, Cui Jianjing, and Long Jun. 2018. A survey of clustering with deep learning: From the perspective of network architecture. *IEEE Access* 6 (2018), 39501–39514.

Mark E. J. Newman. 2006. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences* 103, 23 (2006), 8577–8582.

Mark E. J. Newman and Aaron Clauset. 2016. Structure and inference in annotated networks. *Nature Communications* 7, 1 (2016), 11863.

Mark E. J. Newman and Michelle Girvan. 2004. Finding and evaluating community structure in networks. *Physical Review E* 69, 2 (2004), 026113.

Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. 2002. On spectral clustering: Analysis and an algorithm. In *Proceedings of the 14th International Conference on Neural Information Processing Systems*. 849–856.

Kamal Nigam and Rayid Ghani. 2000. Analyzing the effectiveness and applicability of co-training. In *Proceedings of the 9th International Conference on Information and Knowledge Management*. ACM, 86–93.

Günce Orman, Vincent Labatut, Marc Plantevit, and Jean-François Boulicaut. 2015. Interpreting communities based on the evolution of a dynamic attributed network. *Social Network Analysis and Mining* 5, 1 (2015), 20. DOI: https://doi.org/10.1007/s13278-015-0262-4

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. In *NIPS-W*.

Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 701–710.

Michael D. Plummer and László Lovász. 1986. *Matching Theory*. Vol. 29. Elsevier.

Ioannis Psorakis, Stephen Roberts, Mark Ebden, and Ben Sheldon. 2011. Overlapping community detection using bayesian non-negative matrix factorization. *Physical Review E* 83, 6 (2011), 066114.

Usha Nandini Raghavan, Réka Albert, and Soundar Kumara. 2007. Near linear time algorithm to detect community structures in large-scale networks. *Physical Review E* 76, 3 (2007), 036106.

Martin Rosvall and Carl T. Bergstrom. 2008. Maps of random walks on complex networks reveal community structure. *Proceedings of the National Academy of Sciences* 105, 4 (2008), 1118–1123.

Sam T. Roweis and Lawrence K. Saul. 2000. Nonlinear dimensionality reduction by locally linear embedding. *Science* 290, 5500 (2000), 2323–2326.

Heli Sun, Hongxia Du, Jianbin Huang, Zhongbin Sun, Liang He, Xiaolin Jia, and Zhongmeng Zhao. 2018. Detecting semantic-based communities in node-attributed graphs. *Computational Intelligence* 34, 4 (2018), 1199–1222.

Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 1067–1077.

Joshua B. Tenenbaum, Vin De Silva, and John C. Langford. 2000. A global geometric framework for nonlinear dimensionality reduction. *Science* 290, 5500 (2000), 2319–2323.

Fei Tian, Bin Gao, Qing Cui, Enhong Chen, and Tie-Yan Liu. 2014. Learning deep representations for graph clustering. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence*. 1293–1299.

Ulrike Von Luxburg. 2007. A tutorial on spectral clustering. *Statistics and Computing* 17, 4 (2007), 395–416.

Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1225–1234.

Zhenwen Wang, Yanli Hu, Weidong Xiao, and Bin Ge. 2013. Overlapping community detection using a generative model for networks. *Physica A Statistical Mechanics and Its Applications* 392, 20 (2013), 5218–5230.

Junyuan Xie, Ross Girshick, and Ali Farhadi. 2016. Unsupervised deep embedding for clustering analysis. In *Proceedings of the International Conference on Machine Learning*. 478–487.

Xiaowei Xu, Nurcan Yuruk, Zhidan Feng, and Thomas A. J. Schweiger. 2007. Scan: A structural clustering algorithm for networks. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 824–833.

Xiao Xu, Cheng, Fujimaki, and Muraoka. 2017. Efficient nonparametric and asymptotic Bayesian model selection methods for attributed graph clustering. *Knowledge and Information Systems* 53, 1 (2017), 239–268. DOI:https://doi.org/10.1007/s10115-017-1030-8

Zhiqiang Xu, Yiping Ke, Yi Wang, Hong Cheng, and James Cheng. 2012. A model-based approach to attributed graph clustering. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. ACM, 505–516.

Zhiqiang Xu, Yiping Ke, Yi Wang, Hong Cheng, and James Cheng. 2014. GBAGC: A general bayesian framework for attributed graph clustering. *ACM Transactions on Knowledge Discovery from Data* 9, 1 (2014), 5.

Cheng Yang, Zhiyuan Liu, Deli Zhao, Maosong Sun, and Edward Chang. 2015. Network representation learning with rich text information. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence*.

Jaewon Yang and Jure Leskovec. 2013. Overlapping community detection at scale: A nonnegative matrix factorization approach. In *Proceedings of the 6th ACM International Conference on Web Search and Data Mining*. ACM, 587–596.

Jaewon Yang, Julian McAuley, and Jure Leskovec. 2013. Community detection in networks with node attributes. In *Proceedings of the IEEE 13th International Conference on Data Mining*. IEEE, 1151–1156.

J. Yang, D. Parikh, and D. Batra. 2016. Joint unsupervised learning of deep representations and image clusters. In *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition*. 5147–5156. DOI:https://doi.org/10.1109/CVPR.2016.556

Liang Yang, Xiaochun Cao, Dongxiao He, Chuan Wang, Xiao Wang, and Weixiong Zhang. 2016. Modularity based community detection with deep learning. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence*. 2252–2258.

Xiao Yu, Xiang Ren, Yizhou Sun, Quanquan Gu, Bradley Sturt, Urvashi Khandelwal, Brandon Norick, and Jiawei Han. 2014. Personalized entity recommendation: A heterogeneous information network approach. In *Proceedings of the 7th ACM International Conference on Web Search and Data Mining*. ACM, 283–292.

Hugo Zanghi, Stevenn Volant, and Christophe Ambroise. 2010. Clustering based on random graph model embedding vertex features. *Pattern Recognition Letters* 31, 9 (2010), 830–836.

Fuzheng Zhang, Nicholas Jing Yuan, Defu Lian, Xing Xie, and Wei-Ying Ma. 2016. Collaborative knowledge base embedding for recommender systems. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 353–362.

Yang Zhou, Hong Cheng, and Jeffrey Xu Yu. 2009. Graph clustering based on structural/attribute similarities. *Proceedings of the VLDB Endowment* 2, 1 (2009), 718–729.

Received August 2019; revised December 2019; accepted February 2020