Adaptive Graph Convolution Methods for Attributed Graph Clustering

Xiaotong Zhang, *Member, IEEE,* Han Liu, *Member, IEEE,* Qimai Li, Xiao-Ming Wu, Xianchao Zhang

Abstract—Attributed graph clustering is a challenging task as it requires to jointly model graph structure and node attributes. Although recent advances in graph convolutional networks have shown the effectiveness of graph convolution in combining structural and content information, there is limited understanding of how to properly apply it for attributed graph clustering. Previous methods commonly use a fixed and low order graph convolution, which only aggregates information of few-hop neighbours and hence cannot fully capture the cluster structures of diverse graphs. In this paper, we first propose an adaptive graph convolution method (AGC) for attributed graph clustering, which exploits high-order graph convolutions to capture global cluster structures and adaptively selects an appropriate order k via intra-cluster distance. While AGC can find a reasonable k and avoid over-smoothing, it is not sensitive to the gradual decline of clustering performance as k increases. To search for a better k, we further propose an improved adaptive graph convolution method (IAGC) that not only observes the variation of intra-cluster distance, but also considers the inconsistencies of filtered features with graph structure and raw features, respectively. We establish the validity of our methods by theoretical analysis and extensive experiments on various benchmark datasets.

Index Terms—Attributed graph clustering, adaptive graph convolution, low-pass graph filter

1 Introduction

Attributed graph clustering [1] aims to cluster nodes of an attributed graph where each node is associated with a set of feature attributes. Attributed graphs widely exist in real-world applications such as social networks [2], citation networks [3], and protein-protein interaction networks [4]. Clustering plays an important role in detecting communities and analyzing network structures. The main challenge of attributed graph clustering lies in joint modeling of graph structure and node attributes to properly identify clusters in data.

Some classical clustering methods such as k-means only deal with data features. In contrast, many graph-based clustering methods [5] only leverage graph connectivity patterns, e.g., user friendships in social networks, paper citation links in citation networks, and genetic interactions in protein-protein interaction networks. Typically, these methods learn node embeddings using Laplacian eigenmaps [6], random walks [7], or autoencoder [8]. Nevertheless, they usually fall short in attributed graph clustering, since they do not make use of informative node features such as user profiles in social networks, document contents in citation networks, and protein signatures in protein-protein interaction networks.

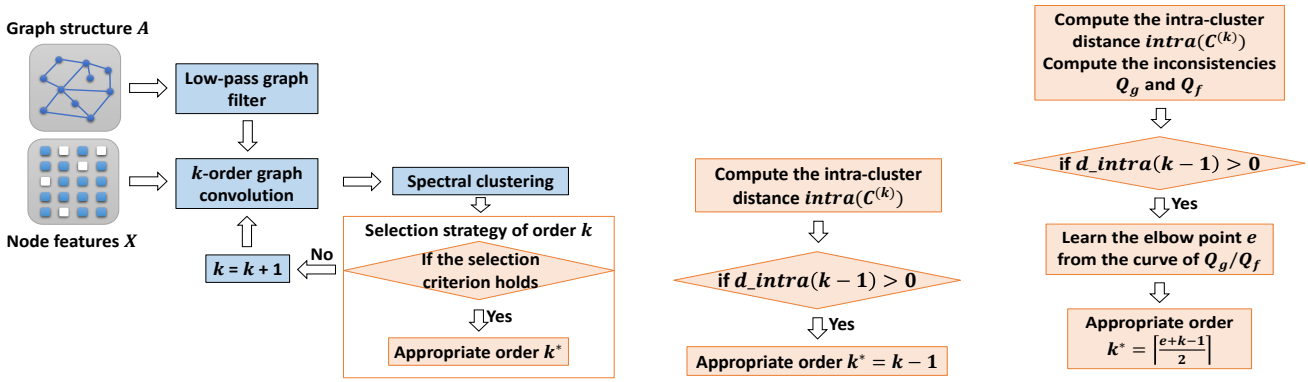
In recent years, various attributed graph clustering methods have been proposed, including methods based on generative models [9], spectral clustering [10], random walks [11], nonnegative matrix factorization [12], and graph convolutional networks (GCN) [13]. In particular, GCN-based methods such as GAE [14], MGAE [15], ARGE [16], DGI [17], SDCN [18] have demonstrated state-of-the-art performance on some attributed graphs.

The key component of GCN-based methods is graph convolution, which nicely integrates structural and feature information in learning node representations. However, there is limited study of how to properly apply it to improve clustering performance. Most existing methods use GCN as a feature extractor straightforwardly, where each convolutional layer is coupled with a projection layer, making it difficult to stack many layers and train a deep model. More importantly, these methods commonly use a shallow two-layer or three-layer GCN, which only aggregates information of neighbours of each node in two or three hops away and hence are inadequate to capture the global cluster structures of large graphs. According to the studies in [19], [20], [21], utilizing local information is not enough for clustering. Further, using a fixed-order graph convolution can only smooth the features of nodes within a fixed distance (i.e., neighbours of each node within the same number of hops), which is clearly not enough for identifying clusters in diverse graphs.

To address these issues, we propose two adaptive graph convolution methods AGC and IAGC for attributed graph clustering. Based on the common assumption that neighbouring nodes tend to be in the same cluster, we aim to learn node representations such that those in the same cluster will have similar representations, which can make clustering much easier. To this end, instead of stacking many layers as in vanilla GCN, we design a new k-order graph convolution that acts as a low-pass graph filter on node features to generate smooth feature representations, where k can be adaptively selected. Our method AGC chooses k by observing the variation of intra-cluster distance, which helps to select a reasonable order k that keeps the clustering performance from declining rapidly due to over-smoothing. However, as observed from our experiments, this strategy is not sensitive to the gradual decline of the clustering performance as k increases. To search for a better k, we further propose IAGC which observes the

- *Xiaotong Zhang, Han Liu and Xianchao Zhang are with School of Software, Dalian University of Technology, Dalian, Ganjingzi Qu 116024, China. Qimai Li and Xiao-Ming Wu are with Department of Computing, The Hong Kong Polytechnic University, HongKong, China. E-mail: zxt.dut@hotmail.com, liu.han.dut@gmail.com, {csqmli,csxmwu}@comp.polyu.edu.hk, xczhang@dlut.edu.cn.*

Manuscript received April 19, 2005; revised August 26, 2015.



(a) The unified framework of AGC and IAGC (b) Selection strategy of AGC (c) Selection strategy of IAGC

Fig. 1: Overview of AGC and IAGC. They first conduct k-order graph convolution with a low-pass graph filter to obtain smooth node features, then perform spectral clustering on the learned features to cluster the nodes. Both AGC and IAGC can search for an appropriate order k for different graphs. AGC selects the order k by finding the first local minimum of intra-cluster distance. IAGC improves the selection strategy by observing the ratio between the inconsistency of filtered features with graph structure Q_g and that with raw features Q_f.

inconsistencies of filtered features with graph structure and raw features respectively, in addition to the variation of intra-cluster distance. The improved selection strategy can identify an order k closer to the best one. Both AGC and IAGC consist of three steps: 1) conducting k-order graph convolution to obtain smooth feature representations; 2) performing spectral clustering on the learned features to cluster the nodes; 3) searching for an appropriate order k with some selection strategy. The overview of AGC and IAGC is shown in Fig. 1. Different from existing work of attributed graph clustering that focus on model design, our work is devoted to studying the graph filter and the neighbourhood size that are suitable for different attributed graphs. In particular, our main contributions are described as follows.

1) We design a new low-pass graph filter based on the theory of graph signal processing, and theoretically show that k-order graph convolution with a low-pass filter produces smoother features as k increases, shedding new light on applying graph convolution for clustering.

2) We propose two attributed graph clustering methods AGC and IAGC based on the proposed k-order graph convolution, which takes into account long-range neighbourhood relations to capture global cluster structures and adaptively selects an appropriate order k for diverse graphs to avoid over-smoothing and optimize clustering performance.

3) We design two strategies to search for an appropriate order k. AGC selects k by observing the variation of intra-cluster distance only, while IAGC also considers the inconsistencies of filtered features with graph structure and raw features respectively, which helps to find a better k.

4) We evaluate the performance of our methods on five benchmark datasets including three citation networks, one webpage network, and one co-purchase network. Experimental results show that AGC and IAGC compare favorably with and in many cases significantly outperform state-of-the-art methods.

A preliminary version of this paper was published in proceedings of the 28th IJCAI conference (IJCAI-19) [22]. Compared with the conference version, the technical contributions of this paper is that we observe the strategy of AGC is not sensitive to the gradual decline of the clustering performance as k increases, so we propose a new strategy to search for a better order k for different attributed graphs. In particular, we extend the preliminary version as 1) We propose an improved adaptive graph convolution method (IAGC) with a new strategy for selecting the order k. 2) We validate the effectiveness of the selection strategies in AGC and IAGC empirically and provide analysis on the time complexity of both methods. 3) We experiment with three additional attributed graph datasets and include seven more baselines (state-of-the-art attributed graph embedding or clustering methods) for comparison. 4) We provide a more detailed description of related works to make the paper self-contained.

2 Related Work

Graph-based clustering methods can be roughly categorized into two branches: structural graph clustering and attributed graph clustering.

Structural graph clustering methods [23] only exploit graph structure [24]. Methods based on graph Laplacian eigenmaps [6], [25] assume that nodes with higher similarity should be mapped closer. Methods based on matrix factorization [26], [27], [28] factorize the node adjacency matrix into node embeddings. Methods based on random walks [7] learn node embeddings by maximizing the probability of the neighbourhood of each node. Autoencoder based methods [8], [29] find low-dimensional node embeddings with the node adjacency matrix and then use the embeddings to reconstruct the adjacency matrix. Some autoencoder based works [30] further exploit generative adversarial net to constrain node embeddings to match a prior distribution.

Attributed graph clustering takes into account both node connectivity and features. Some methods model the interaction between graph connectivity and node features with generative models [9], [31]. Some methods apply nonnegative matrix factorization or spectral clustering on both the underlying graph and node features to get a consistent cluster partition [10], [11]. Some most recent methods integrate node relations and features using GCN [13]. One branch focuses on graph embedding learning, and the learned node embeddings can be used for node clustering. GAE and VGAE [14] learn node representations with a two-layer GCN and then reconstruct the node adjacency matrix with autoencoder and variational autoencoder respectively. MGAE [15] learns node representations with a three-layer GCN and then applies marginalized denoising autoencoder to reconstruct the

given node features. ARGE and ARVGE [16] learn node embeddings by GAE and VGAE respectively and then use generative adversarial networks to enforce the node embeddings to match a prior distribution. AMIL [32] performs adversarial learning on the representation mechanism in encoder to learn a robust two-layer graph convolutional encoder. DGI [17] obtains the patch representation with one-layer GCN or three graph attention layers, then maximizes mutual information between patch representations and corresponding high-level summaries of a graph. GMI [33] employs two discriminators to maximize mutual information between input and representations of both nodes and edges, where the representations are learned from one-layer or two-layer GCN.

The other branch is specifically designed for node clustering. DAEGC [34] stacks two graph attention layers in which attention mechanism is used to adjust weights of existing edges, then adopts graph structure reconstruction loss as well as a self-optimizing clustering loss to update the node embeddings and obtain the clustering results. SDCN [18] first adopts deep neural network model with feature reconstruction loss to get a cluster assignment distribution, then uses this distribution to guide the clustering assignment generated by a two-layer GCN. MvAGC [35] applies a two-order graph filter to obtain smooth node representations, then uses the self-expressiveness property of data and a regularization term to learn node similarity matrix, the method is finally extended to tackle multi-view [36] attributed graphs. SP-DAGC [37] stacks a six-layer GCN by reconstructing both node features and graph structure, then adopts a self-optimizing clustering loss to update the node embeddings and obtain the clustering results. SENet [38] first improves graph structure [39] by leveraging the information of shared neighbours, then stacks a two-layer GCN to learn node embeddings and cluster partitions with the help of a spectral clustering loss. CGC [40] uses a graph neural network (GNN) with a mean aggregator to construct the node encoder, and adopts contrastive learning to learn node embeddings and cluster partitions simultaneously. GCC [41] uses a simple graph convolutional network to learn convolved node representations, and realizes node clustering by minimizing the difference between the convolved node representations and their reconstructed cluster representatives. MAGCN [42] develops a two-layer multi-view attribute graph convolution encoder with attention mechanism for learning node embeddings and a consistent embedding encoder for extracting the consistency information among multiple views, and adopts attribute and graph reconstruction losses as well as a self-optimizing clustering loss to update the node embeddings and obtain the clustering results. SGCMC [43] uses a four-layer graph attention encoder and a consistent representation constraint to learn node representations and the geometric similarity embedded in different views, and learns the predicted labels of node embeddings with the help of pseudo labels from spectral clustering, then employs a self-supervised learning module to minimize the discrepancy of the predicted label matrix and the geometric similarity. MinCutPool [44] constructs a GNN followed by a multi-layer perceptron to compute the cluster assignments for spectral clustering. These methods design clustering-directed GNNs to directly obtain node partitions and get satisfactory results, but they either use a fixed-layer model to combine the graph structure and node features, or manually set different graph convolutional layers for different graphs, which cannot adaptively deal with real-world graphs. Although these methods can use residual connections and attention mechanisms to adaptively select the receptive field for each node, the selected receptive field is only appropriate for minimizing the loss function, not for optimizing the clustering performance. The fundamental reason is that they simply use the results of traditional clustering methods such as *k*-means and spectral clustering to guide the clustering results of node embeddings, not considering how to fully utilize the graph structure and node features to optimize the clustering performance for different graphs. Our proposed methods specifically target this issue by continuously expanding the number of hops of neighbours and designing strategies that can measure the performance of the learned node representations and cluster partitions.

There are also some works that propose low-pass graph filters or select adaptive hops of neighbours for GNNs. In particular, SGCN [45] proposes to simplify GCN [13] by removing nonlinearities and collapsing weight matrices between consecutive layers, and shows that the simplified model corresponds to a fixed low-pass-type graph filter followed by a linear classifier. Geniepath [46] proposes to learn adaptive receptive paths for each node by designing adaptive breadth and depth functions. The adaptive breadth function weights the one-hop neighbours of each node, while the adaptive depth function weights the neighbours of different hops for each node, and the weights can be learned by optimizing the classification losses. However, different from these works that are specifically designed for node classification, our work studies the node clustering problem. Moreover, our work proposes new ways for designing the low-pass graph filter and selection strategies of the order in graph convolution. Specifically, our work first designs a low-pass graph filter based on the theory of graph signal processing, then analyzes that a too low-pass graph filter will cause the over-smoothing issue, and hence proposes two strategies to adaptively select the appropriate order for different graphs to achieve better clustering performance.

3 Problem Formulation and Graph Convolution

3.1 Problem Formulation

Given a non-directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, X)$, where $\mathcal{V} = \{v_1, v_2, ..., v_n\}$ is a set of nodes with $|\mathcal{V}| = n$. $\mathcal{E}$ is a set of edges that can be represented as an adjacency matrix $\boldsymbol{A} = \{a_{ij}\} \in \mathbb{R}^{n\times n}$, where $a_{ij} = 1$ if node v_i and node v_j are connected, otherwise we have $a_{ij} = 0$. $\boldsymbol{X}$ is a feature matrix of all the nodes, i.e., $\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_n]^\top \in \mathbb{R}^{n\times d}$, where $\boldsymbol{x}_i \in \mathbb{R}^d$ is a real-valued feature vector of node v_i, and d is the number of attributes of each node. Our goal is to partition the nodes of the graph $\mathcal{G}$ into m clusters $\mathcal{C} = \{C_1, C_2, \cdots, C_m\}$. Note that we call v_j a k-hop neighbour of v_i, if v_j can reach v_i by traversing k edges.

3.2 Graph Convolution

To formally define graph convolution, we first introduce the notions of graph signal and graph filter [47].

A *graph signal* can be represented as a vector $\boldsymbol{f} = [f(v_1), \cdots, f(v_n)]^\top$, where $f : \mathcal{V} \to \mathbb{R}$ is a real-valued function on the nodes of a graph. In attributed graphs, each column of the feature matrix $\boldsymbol{X}$ can be considered as a graph signal.

Given an adjacency matrix $\boldsymbol{A}$ and the degree matrix $\boldsymbol{D} = \mathrm{diag}(d_1, \cdots, d_n)$, the symmetrically normalized graph Laplacian $\boldsymbol{L}_s = \boldsymbol{I} - \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{-\frac{1}{2}}$ can be eigen-decomposed as $\boldsymbol{L}_s = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^{-1}$, where $\boldsymbol{\Lambda} = \mathrm{diag}(\lambda_1, \cdots, \lambda_n)$ are the eigenvalues in ascending order, and $\boldsymbol{U} = [\boldsymbol{u}_1, \cdots, \boldsymbol{u}_n]$ are the associated orthogonal eigenvectors. A linear *graph filter* can be

represented as a matrix $\boldsymbol{G} = \boldsymbol{U}p(\boldsymbol{\Lambda})\boldsymbol{U}^{-1} \in \mathbb{R}^{n\times n}$, where $p(\boldsymbol{\Lambda}) = \text{diag}(p(\lambda_1), \cdots, p(\lambda_n))$ is called the frequency response function of $\boldsymbol{G}$.

Graph convolution [47] is defined as the multiplication of a graph signal $\boldsymbol{f}$ with a graph filter $\boldsymbol{G}$:

$$\bar{\boldsymbol{f}} = \boldsymbol{G}\boldsymbol{f}, \tag{1}$$

where $\bar{\boldsymbol{f}}$ is the filtered graph signal.

4 Adaptive Graph Convolution Method

We first design a low-pass graph filter based on graph signal processing, then propose an adaptive graph convolution method (AGC) by exploiting k-order graph convolution to capture long-distance data relations and adaptively selecting the order k for different graphs based on intra-cluster distance.

4.1 Low-Pass Graph Filter Design

In graph signal processing [47], the eigenvalues $(\lambda_q)_{1\le q\le n}$ can be taken as frequencies and the associated eigenvectors $(\boldsymbol{u}_q)_{1\le q\le n}$ are considered as Fourier basis of the graph. A graph signal $\boldsymbol{f}$ can be decomposed into a linear combination of the eigenvectors $(\boldsymbol{u}_q)_{1\le q\le n}$ [47], i.e.,

$$\boldsymbol{f} = \boldsymbol{U}\boldsymbol{z} = \sum\nolimits_{q=1}^{n} z_q\boldsymbol{u}_q, \tag{2}$$

where $\boldsymbol{z} = [z_1, \cdots, z_n]^\top$ and z_q is the coefficient of $\boldsymbol{u}_q$. The magnitude of the coefficient $|z_q|$ indicates the strength of the basis signal $\boldsymbol{u}_q$ presented in $\boldsymbol{f}$.

A graph signal is smooth if nearby nodes on the graph have similar feature representations. Since a graph signal is a linear combination of a group of basis signals $(\boldsymbol{u}_q)_{1\le q\le n}$, we analyze the smoothness of basis signals first. The smoothness of a basis signal $\boldsymbol{u}_q$ can be measured by Laplacian-Beltrami operator [48],

$$\begin{aligned}\Omega(\boldsymbol{u}_q) &= \frac{1}{2}\sum_{(v_i,v_j)\in\mathcal{E}} a_{ij}\left\|\frac{\boldsymbol{u}_q(i)}{\sqrt{d_i}} - \frac{\boldsymbol{u}_q(j)}{\sqrt{d_j}}\right\|_2^2 \\ &= \boldsymbol{u}_q{}^\top\boldsymbol{L}_s\boldsymbol{u}_q = \lambda_q,\end{aligned} \tag{3}$$

where $\boldsymbol{u}_q(i)$ denotes the i-th element of the vector $\boldsymbol{u}_q$, and the smaller $\Omega(\boldsymbol{u}_q)$ is, the smoother $\boldsymbol{u}_q$ becomes. Eq. (3) indicates that the basis signals associated with lower frequencies (smaller eigenvalues) are smoother, which means that a smooth graph signal $\boldsymbol{f}$ should contain more low-frequency basis signals than high-frequency ones. This can be achieved by performing graph convolution with a low-pass graph filter $\boldsymbol{G}$, as shown below.

According to Eq. (2), the graph convolution can be written as

$$\bar{\boldsymbol{f}} = \boldsymbol{G}\boldsymbol{f} = \boldsymbol{U}p(\boldsymbol{\Lambda})\boldsymbol{U}^{-1}\cdot\boldsymbol{U}\boldsymbol{z} = \sum\nolimits_{q=1}^{n} p(\lambda_q)z_q\boldsymbol{u}_q. \tag{4}$$

In the filtered signal $\bar{\boldsymbol{f}}$, the coefficient z_q of the basis signal $\boldsymbol{u}_q$ is scaled by $p(\lambda_q)$. As $\boldsymbol{\Lambda} = \text{diag}(\lambda_1, \cdots, \lambda_n)$ are the eigenvalues (frequencies) in ascending order, to preserve the low-frequency basis signals and remove the high-frequency ones in $\boldsymbol{f}$, the frequency response function $p(\cdot)$ should be decreasing and nonnegative, i.e., the graph filter $\boldsymbol{G}$ should be low-pass.

A low-pass graph filter can take on many forms. Here, we design a low-pass graph filter with the frequency response function

$$p(\lambda_q) = 1 - \frac{1}{2}\lambda_q. \tag{5}$$

As shown by the red line in Fig. 2, one can see that $p(\cdot)$ in Eq. (5) is decreasing and nonnegative on $[0, 2]$. Note that all the eigenvalues λ_q of the symmetrically normalized graph Laplacian $\boldsymbol{L}_s$ fall into interval $[0, 2]$ [48], which indicates that the graph filter $\boldsymbol{G}$ with $p(\cdot)$ in Eq. (5) is low-pass.

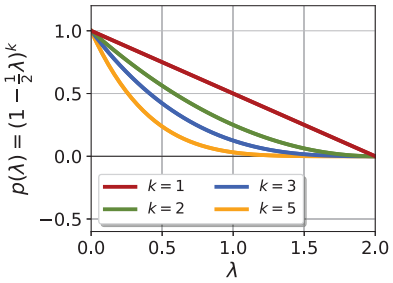


Fig. 2: Frequency response functions.

The graph filter $\boldsymbol{G}$ with $p(\cdot)$ in Eq. (5) as the frequency response function can be formulated as

$$\boldsymbol{G} = \boldsymbol{U}p(\boldsymbol{\Lambda})\boldsymbol{U}^{-1} = \boldsymbol{U}(\boldsymbol{I} - \frac{1}{2}\boldsymbol{\Lambda})\boldsymbol{U}^{-1} = \boldsymbol{I} - \frac{1}{2}\boldsymbol{L}_s. \tag{6}$$

By performing graph convolution on the feature matrix $\boldsymbol{X}$, we obtain the filtered feature matrix:

$$\bar{\boldsymbol{X}} = \boldsymbol{G}\boldsymbol{X}, \tag{7}$$

where $\bar{\boldsymbol{X}} = [\bar{\boldsymbol{x}}_1, \bar{\boldsymbol{x}}_2, \cdots, \bar{\boldsymbol{x}}_n]^\top \in \mathbb{R}^{n\times d}$ is the filtered node features after graph convolution. According to the analysis of Eq. (3) and Eq. (4), applying such a low-pass graph filter on the feature matrix makes adjacent nodes have similar feature values along each dimension, i.e., the graph signals are smooth, and it is also proved by Theorem 1. Based on the cluster assumption that nearby nodes are likely to have the same label [19], performing graph convolution with a low-pass graph filter will make the nodes in the same cluster have similar representations, thus facilitating the downstream clustering task.

Note that the proposed graph filter in Eq. (6) is quite different from the graph filter used in GCN. The graph filter used in GCN is $\boldsymbol{G} = \boldsymbol{I} - \boldsymbol{L}_s$, which is with the frequency response function $p(\lambda_q) = 1 - \lambda_q$ [49]. It is clearly not low-pass as it is negative for $\lambda_q \in (1, 2]$.

4.2 k-Order Graph Convolution

To make clustering easy, it is desired that nodes of the same class should have similar feature representations after graph filtering. However, the first-order graph convolution in Eq. (7) may not be adequate to achieve this, especially for large and sparse graphs, as it updates each node v_i by the aggregation of its 1-hop neighbours only, without considering long-distance neighbourhood relations. To capture global graph structure and facilitate clustering, we propose to use k-order graph convolution.

We define k-order graph convolution as

$$\bar{\boldsymbol{X}}^{(k)} = (\boldsymbol{I} - \frac{1}{2}\boldsymbol{L}_s)^k\boldsymbol{X}, \tag{8}$$

where k is a positive integer, and the corresponding graph filter is

$$\boldsymbol{G} = (\boldsymbol{I} - \frac{1}{2}\boldsymbol{L}_s)^k = \boldsymbol{U}(\boldsymbol{I} - \frac{1}{2}\boldsymbol{\Lambda})^k\boldsymbol{U}^{-1}. \tag{9}$$

The frequency response function of $\boldsymbol{G}$ in Eq. (9) is

$$p(\lambda_q) = (1 - \frac{1}{2}\lambda_q)^k. \tag{10}$$

As shown in Fig. 2, $p(\lambda_q)$ in Eq. (10) becomes more low-pass as k increases, indicating that the filtered node features $\bar{\boldsymbol{X}}^{(k)}$ will be smoother.

(a) Raw features (b) $k = 1$ (c) $k = 12$ (d) $k = 100$

Fig. 3: (a-d) t-SNE visualization of the raw and filtered node features of Cora with different k.

The iterative calculation formula of k-order graph convolution is

$$\bar{\boldsymbol{x}}_i^{(0)} = \boldsymbol{x}_i, \quad \bar{\boldsymbol{x}}_i^{(1)} = \frac{1}{2}\bar{\boldsymbol{x}}_i^{(0)} + \frac{1}{2}\sum_{(v_i,v_j)\in\mathcal{E}} \frac{a_{ij}}{\sqrt{d_i d_j}}\bar{\boldsymbol{x}}_j^{(0)}, \cdots,$$
$$\bar{\boldsymbol{x}}_i^{(k)} = \frac{1}{2}\bar{\boldsymbol{x}}_i^{(k-1)} + \frac{1}{2}\sum_{(v_i,v_j)\in\mathcal{E}} \frac{a_{ij}}{\sqrt{d_i d_j}}\bar{\boldsymbol{x}}_j^{(k-1)}, \tag{11}$$

and the final $\bar{\boldsymbol{x}}_i$ is $\bar{\boldsymbol{x}}_i^{(k)}$. From Eq. (11), one can easily see that k-order graph convolution updates the features of each node v_i by aggregating the features of its k-hop neighbours iteratively. As k-order graph convolution takes into account long-distance data relations, it can be useful for capturing global graph structure to improve clustering performance.

4.3 Theoretical Analysis

As k increases, k-order graph convolution will make the node features smoother on each dimension. In the following, we prove this using the Laplacian-Beltrami operator $\Omega(\cdot)$ defined in Eq. (3). Denote by $\boldsymbol{f}$ a column of the feature matrix $\boldsymbol{X}$, which can be decomposed as $\boldsymbol{f} = \boldsymbol{U}\boldsymbol{z}$. Note that $\Omega(\beta\boldsymbol{f}) = \beta^2\Omega(\boldsymbol{f})$, where β is a scalar. Therefore, to compare the smoothness of different graph signals, we need to put them on a common scale. In what follows, we consider the smoothness of a normalized signal $\frac{\boldsymbol{f}}{\|\boldsymbol{f}\|_2}$, i.e.,

$$\Omega\left(\frac{\boldsymbol{f}}{\|\boldsymbol{f}\|_2}\right) = \frac{\boldsymbol{f}^\top \boldsymbol{L}_s \boldsymbol{f}}{\|\boldsymbol{f}\|_2^2} = \frac{\boldsymbol{z}^\top \boldsymbol{\Lambda} \boldsymbol{z}}{\|\boldsymbol{z}\|_2^2} = \frac{\sum_{i=1}^n \lambda_i z_i^2}{\sum_{i=1}^n z_i^2}. \tag{12}$$

Theorem 1. *If the frequency response function $p(\lambda)$ of a graph filter $\boldsymbol{G}$ is nonincreasing and nonnegative for all λ_i, then for any signal $\boldsymbol{f}$ and the filtered signal $\bar{\boldsymbol{f}} = \boldsymbol{G}\boldsymbol{f}$, we always have*

$$\Omega\left(\frac{\bar{\boldsymbol{f}}}{\|\bar{\boldsymbol{f}}\|_2}\right) \le \Omega\left(\frac{\boldsymbol{f}}{\|\boldsymbol{f}\|_2}\right).$$

Proof. We first prove the following lemma by induction. The following inequality

$$T_c^{(n)} = \frac{\sum_{i=1}^n c_i T_i}{\sum_{i=1}^n c_i} \le \frac{\sum_{i=1}^n b_i T_i}{\sum_{i=1}^n b_i} = T_b^{(n)} \tag{13}$$

holds, if $T_1 \le \cdots \le T_n$ and $\frac{c_1}{b_1} \ge \cdots \ge \frac{c_n}{b_n}$ with $\forall c_i, b_i \ge 0$. It is easy to validate that it holds when $n = 2$.

Now assume that it holds when $n = l-1$, i.e., $T_c^{(l-1)} \le T_b^{(l-1)}$. Then, consider the case of $n = l$ and we have

$$\begin{aligned}\frac{\sum_{i=1}^l c_i T_i}{\sum_{i=1}^l c_i} &= \frac{\sum_{i=1}^{l-1} c_i T_i + c_l T_l}{\sum_{i=1}^{l-1} c_i + c_l} \\ &= \frac{(\sum_{i=1}^{l-1} c_i) T_c^{(l-1)} + c_l T_l}{\sum_{i=1}^{l-1} c_i + c_l} \\ &\le \frac{(\sum_{i=1}^{l-1} c_i) T_b^{(l-1)} + c_l T_l}{\sum_{i=1}^{l-1} c_i + c_l}.\end{aligned} \tag{14}$$

Since $T_b^{(l-1)} = \frac{\sum_{i=1}^{l-1} b_i T_i}{\sum_{i=1}^{l-1} b_i} \le \frac{\sum_{i=1}^{l-1} b_i T_{l-1}}{\sum_{i=1}^{l-1} b_i} = T_{l-1}$, we have $T_b^{(l-1)} \le T_l$. Also note that $\frac{\sum_{i=1}^{l-1} c_i}{\sum_{i=1}^{l-1} b_i} \ge \frac{c_l}{b_l}$. Since the lemma holds when $n = 2$, we have

$$\begin{aligned}\frac{(\sum_{i=1}^{l-1} c_i) T_b^{(l-1)} + c_l T_l}{\sum_{i=1}^{l-1} c_i + c_l} &\le \frac{(\sum_{i=1}^{l-1} b_i) T_b^{(l-1)} + b_l T_l}{\sum_{i=1}^{l-1} b_i + b_l} \\ &= \frac{\sum_{i=1}^{l} b_i T_i}{\sum_{i=1}^{l} b_i},\end{aligned} \tag{15}$$

which shows that the inequality Eq. (13) also holds when $n = l$. By induction, the above lemma holds for all n.

We can now prove Theorem 1 using this lemma. For convenience, we arrange the eigenvalues λ_i of L_s in ascending order such that $0 \le \lambda_1 \le \cdots \le \lambda_n$. Since $p(\lambda)$ is nonincreasing and nonnegative, $p(\lambda_1) \ge \cdots \ge p(\lambda_n) \ge 0$. Theorem 1 can then be proved with the above lemma by letting

$$T_i = \lambda_i, \quad b_i = z_i^2, \quad c_i = p^2(\lambda_i) z_i^2. \tag{16}$$

□

Assuming that $\boldsymbol{f}$ and $\bar{\boldsymbol{f}}$ are obtained by $(k-1)$-order and k-order graph convolution respectively, one can immediately infer from Theorem 1 that $\bar{\boldsymbol{f}}$ is smoother than $\boldsymbol{f}$. In other words, k-order graph convolution will produce smoother features as k increases. Since nodes in the same cluster tend to be densely connected, they are likely to have more similar feature representations with large k, which can be beneficial for clustering.

4.4 Clustering via Adaptive Graph Convolution

Traditional clustering methods could be applied to the filtered features to obtain clustering results. In this paper, we give preference to spectral clustering, which does not make strong assumptions on the form of the clusters [50]. In particular, we perform the classical spectral clustering method [50] on the filtered feature matrix $\bar{\boldsymbol{X}}^{(k)}$ to partition the nodes of $\mathcal{V}$ into m clusters, similar to [15]. Specifically, we first apply the linear kernel $\boldsymbol{K} = \bar{\boldsymbol{X}}^{(k)}(\bar{\boldsymbol{X}}^{(k)})^\top$ to learn pairwise similarity between nodes, and then we calculate $\boldsymbol{W} = \frac{1}{2}(|\boldsymbol{K}| + |\boldsymbol{K}^\top|)$ to make sure that the similarity matrix

is symmetric and nonnegative, where $|\cdot|$ means taking absolute value of each element of the matrix. Finally, we perform spectral clustering on $\boldsymbol{W}$ to obtain clustering results by computing the eigenvectors associated with the m largest eigenvalues of $\boldsymbol{W}$ and then applying the k-means algorithm on the eigenvectors to obtain cluster partitions.

The central issue of k-order graph convolution is how to select an appropriate k. Although k-order graph convolution can make nearby nodes have similar feature representations, k is definitely not the larger the better. k being too large will lead to over-smoothing, i.e., the features of nodes in different clusters are mixed and become indistinguishable. Fig. 3 visualizes the raw and filtered node features of the *Cora* citation network with different k, where the features are projected by t-Distributed Stochastic Neighbor Embedding (t-SNE) and nodes of the same class are indicated by the same colour. t-SNE is a nonlinear dimension reduction tool for visualizing high-dimensional data by preserving data similarity in a low-dimensional space. It can be seen that the node features become similar as k increases. The data exhibits clear cluster structures with $k = 12$. However, with $k = 100$, the features are over-smoothed and nodes from different clusters are mixed together.

To adaptively select the order k, we use the clustering performance metric – internal criteria based on the information intrinsic to the data alone [51]. Here, we consider intra-cluster distance ($intra(\mathcal{C})$ for a given cluster partition $\mathcal{C}$), which represents the compactness of $\mathcal{C}$:

$$intra(\mathcal{C}) = \frac{1}{|\mathcal{C}|}\sum_{C\in\mathcal{C}}\frac{1}{|C|(|C|-1)}\sum_{\substack{v_i,v_j\in C,\\ v_i\neq v_j}}\|\bar{\boldsymbol{x}}_i-\bar{\boldsymbol{x}}_j\|_2. \quad (17)$$

Note that inter-cluster distance can also be used to measure clustering performance given fixed data features, and a good cluster partition should have a large inter-cluster distance and a small intra-cluster distance. However, by Theorem 1, the node features become smoother as k increases, which could significantly reduce both intra-cluster and inter-cluster distances. Hence, inter-cluster distance may not be a reliable metric for measuring the clustering performance with respect to (w.r.t.) different k, and so we propose to observe the variation of intra-cluster distance for choosing k.

Our strategy is to find the first local minimum of $intra(\mathcal{C})$ w.r.t. k. Specifically, we start from $k = 1$ and increment it by 1 iteratively. In each iteration t, we first obtain the cluster partition $\mathcal{C}^{(t)}$ by performing k-order ($k = t$) graph convolution and spectral clustering, then we compute $intra(\mathcal{C}^{(t)})$. Once $intra(\mathcal{C}^{(t)})$ is larger than $intra(\mathcal{C}^{(t-1)})$, we stop the iteration and set the chosen $k = t - 1$. More formally, consider $d\_intra(t-1) = intra(\mathcal{C}^{(t)}) - intra(\mathcal{C}^{(t-1)})$, the criterion for stopping the iteration is $d\_intra(t-1) > 0$, i.e., stops at the first local minimum of $intra(\mathcal{C}^{(t)})$. Therefore, the final choice of cluster partition is $\mathcal{C}^{(t-1)}$. The benefits of this selection strategy are two-fold. First, it ensures finding a local minimum for $intra(\mathcal{C})$ that may indicate a good cluster partition and avoids over-smoothing. Second, it is time efficient to stop at the first local minimum of $intra(\mathcal{C})$.

4.5 Algorithm Procedure and Time Complexity

The overall procedure of AGC is shown in Algorithm 1. Denote by n the number of nodes, d the number of attributes, m the number of clusters, and N the number of nonzero entries of the adjacency matrix $\boldsymbol{A}$. Note that for a sparse $\boldsymbol{A}$, $N << n^2$. The time complexity of computing $\boldsymbol{L}_s$ in the initialization is $\mathcal{O}(N)$. In each iteration, for a sparse $\boldsymbol{L}_s$, the fastest way of computing k-order graph convolution in Eq. (8) is to left multiply $\boldsymbol{X}$ by $\boldsymbol{I}-\frac{1}{2}\boldsymbol{L}_s$ repeatedly for k times, which has the time complexity $\mathcal{O}(Ndk)$. The time complexity of performing spectral clustering on $\bar{\boldsymbol{X}}$ in each iteration is $\mathcal{O}(n^2d+n^2m)$. The time complexity of computing $intra(\mathcal{C})$ in each iteration is $\mathcal{O}(\frac{1}{m}n^2d)$. Since m is usually much smaller than n and d, the time complexity of each iteration is approximately $\mathcal{O}(n^2d+Ndk)$. If Algorithm 1 iterates t times, the overall time complexity of AGC is $\mathcal{O}(n^2dt+Ndt^2)$. Unlike existing GCN based clustering methods, AGC does not need to train the neural network parameters, which makes it time efficient.

Algorithm 1 AGC

Input: Node set $\mathcal{V}$, adjacency matrix $\boldsymbol{A}$, feature matrix $\boldsymbol{X}$, and maximum iteration number max_iter.
Output: Cluster partition $\mathcal{C}$.

```
Initialize t = 0 and intra(C^(0)) = +∞. Compute the symmetrically normalized graph Laplacian L_s = I − D^{−1/2} A D^{−1/2}, where D is the degree matrix of A.
repeat
    Set t = t + 1 and k = t.
    Perform k-order graph convolution by Eq. (8) and get X̄^(k).
    Apply the linear kernel K = X̄^(k) (X̄^(k))^⊤, and calculate the similarity matrix W = 1/2 (|K| + |K^⊤|).
    Obtain the cluster partition C^(t) by performing spectral clustering on W.
    Compute intra(C^(t)) by Eq. (17).
until d_intra(t − 1) > 0 or t > max_iter
Set k = t − 1 and C = C^(t−1).
```

5 Improved Adaptive Graph Convolution Method

We first point out the deficiency of only observing the variation of intra-cluster distance in AGC, then propose the improved adaptive graph convolution method (IAGC) with an improved selection strategy of order k, which not only observes the variation of intra-cluster distance, but also considers the inconsistencies of filtered features with graph structure and raw features.

5.1 Improved Selection Strategy of Order k

The strategy of selecting order k in AGC is to find the first local minimum of $intra(\mathcal{C})$. However, in empirical study, we find that this strategy is more sensitive to the rapid decline of clustering performance. When the clustering performance gradually declines, this strategy usually selects a much larger k or does not find the first local minimum of $intra(\mathcal{C})$. Therefore, we propose an extra selection criterion as supplementary.

Intuitively, node features and graph structure are two kinds of complementary information in attributed graphs, so there are usually some differences between the feature-aware node similarity matrix and the graph structure. As the order k increases, the filtered node features incorporate more and more graph structure information, which will make the filtered node features incline more to graph structure than feature-aware node similarity matrix. This phenomenon has been proved by Theorem 1, i.e., filtered node features become increasingly smooth on the graph structure as k increases. Ideally, the filtered node features should not incline too much to one kind of information. In other words, as k increases, the inconsistency Q_g between the filtered node features and graph structure should not be too small, meanwhile the inconsistency Q_f between the filtered node features and feature-aware node similarity matrix should not be too large. This means

Algorithm 2 IAGC

Input: Node set $\mathcal{V}$, adjacency matrix $\boldsymbol{A}$, feature matrix $\boldsymbol{X}$, the number of components to keep in PCA n_p, and maximum iteration number max_iter.
Output: Cluster partition $\mathcal{C}$.
Perform PCA on node features $\boldsymbol{X}$ and obtain the processed data $\boldsymbol{X}$.
Initialize $t = 0$ and $intra(\mathcal{C}^{(0)}) = +\infty$. Compute the symmetrically normalized graph Laplacian $\boldsymbol{L}_s = \boldsymbol{I} - \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{-\frac{1}{2}}$, where $\boldsymbol{D}$ is the degree matrix of $\boldsymbol{A}$.
Compute the average degree d_{avg} of the nodes according to the adjacency matrix $\boldsymbol{A}$, then calculate the node similarity matrix $\boldsymbol{M}$ on the processed data $\boldsymbol{X}$.
repeat
  Set $t = t + 1$ and $k = t$.
  Perform k-order graph convolution by Eq. (8) and get $\bar{\boldsymbol{X}}^{(k)}$.
  Apply the linear kernel $\boldsymbol{K} = \bar{\boldsymbol{X}}^{(k)}(\bar{\boldsymbol{X}}^{(k)})^\top$, and calculate the similarity matrix $\boldsymbol{W} = \frac{1}{2}(|\boldsymbol{K}| + |\boldsymbol{K}^\top|)$.
  Obtain the cluster partition $\mathcal{C}^{(t)}$ by performing spectral clustering on $\boldsymbol{W}$.
  Compute $intra(\mathcal{C}^{(t)})$ by Eq. (17).
  Compute $Q_g^{(t)}$ and $Q_f^{(t)}$ by Eq. (19) and Eq. (20).
until $d\_intra(t-1) > 0$ or $t > max\_iter$
Learn the elbow point e from the curve of $\frac{Q_g}{Q_f}$.
Set the chosen k as $\lceil \frac{e+t-1}{2} \rceil$ and $\mathcal{C} = \mathcal{C}^{(k)}$.

that the ratio of Q_g to Q_f cannot be too small. Therefore, we propose to observe the curve of $\frac{Q_g}{Q_f}$ w.r.t. the order k.

The smoothness of signal (normalized node features) in Eq. (12) can measure whether the nearby nodes on the graph have similar features. The higher the smoothness is, the larger the inconsistency between features and graph is. Therefore, we use the smoothness to compute Q_g and Q_f.

Given the adjacency matrix $\boldsymbol{A}$ based on graph structure, Q_g can be computed by the smoothness of normalized filtered node features w.r.t. $\boldsymbol{A}$.

$$Q_g^{(k)} = \sum_{i=1}^{d} \Omega\left(\frac{\bar{\boldsymbol{X}}_i^{(k)}}{\|\bar{\boldsymbol{X}}_i^{(k)}\|_2}\right). \tag{18}$$

where $\bar{\boldsymbol{X}}_i^{(k)} \in \mathbb{R}^d$ is the filtered node features in the i-th dimension via k-order graph convolution. Eq. (18) indicates the smoothness of the filtered node features in all dimensions.

Eq. (18) can be further simplified by

$$\begin{aligned} Q_g^{(k)} &= \sum_{i=1}^{d} \frac{(\bar{\boldsymbol{X}}_i^{(k)})^\top \boldsymbol{L}_A \bar{\boldsymbol{X}}_i^{(k)}}{\|\bar{\boldsymbol{X}}_i^{(k)}\|_2^2} \\ &= tr\left(\text{norm}(\bar{\boldsymbol{X}}^{(k)})^\top \boldsymbol{L}_A \text{norm}(\bar{\boldsymbol{X}}^{(k)})\right), \end{aligned} \tag{19}$$

where $\boldsymbol{L}_A$ is symmetrically normalized graph Laplacian of $\boldsymbol{A}$, and $\text{norm}(\bar{\boldsymbol{X}}^{(k)})$ is to normalize $\bar{\boldsymbol{X}}^{(k)}$ in each dimension.

Similarly, given the feature-aware node similarity matrix $\boldsymbol{M}$, Q_f can be formulated as

$$\begin{aligned} Q_f^{(k)} &= \sum_{i=1}^{d} \frac{(\bar{\boldsymbol{X}}_i^{(k)})^\top \boldsymbol{L}_M \bar{\boldsymbol{X}}_i^{(k)}}{\|\bar{\boldsymbol{X}}_i^{(k)}\|_2^2} \\ &= tr\left(\text{norm}(\bar{\boldsymbol{X}}^{(k)})^\top \boldsymbol{L}_M \text{norm}(\bar{\boldsymbol{X}}^{(k)})\right), \end{aligned} \tag{20}$$

where $\boldsymbol{L}_M$ is symmetrically normalized graph Laplacian of $\boldsymbol{M}$.

We adopt the inversion of Euclidean distance to compute the feature-aware node similarity matrix $\boldsymbol{M}$. The raw node features usually have a high dimensionality, which will suffer from the "curse of dimensionality" problem. Therefore, it is necessary to reduce the dimension of the raw node features $\boldsymbol{X}$ at first. In this paper, we propose to perform PCA on $\boldsymbol{X}$, and replace the raw node features with the processed node features of PCA. Then the similarity between the i-th node and the j-th node is computed by

$$m_{ij} = \frac{1}{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2^2 + 1}. \tag{21}$$

We construct a r-nearest neighbour graph to ensure the sparsity of $\boldsymbol{M}$. According to the adjacency matrix $\boldsymbol{A}$, we could obtain the average degree d_{avg} of the nodes in the graph structure, and set $r = d_{avg}$ to obtain the sparse node similarity matrix $\boldsymbol{M}$. Finally, we calculate $\boldsymbol{M} = \frac{1}{2}(|\boldsymbol{M}| + |\boldsymbol{M}^\top|)$ to make sure that the node similarity matrix is symmetric.

Synthesizing the AGC method, our improved strategy is to first perform k-order graph convolution iteratively and record the ratio $\frac{Q_g}{Q_f}$ w.r.t. k, until k leads to the first local minimum of $intra(\mathcal{C})$ or k reaches the maximum iteration number. Then a curve of $\frac{Q_g}{Q_f}$ can be obtained when the iteration stops. As elbow point of a curve has the maximum curvature, which means that the curve starts to decline slightly after the elbow point, we consider there is a point between the elbow point e and k, where $\frac{Q_g}{Q_f}$ is small enough. Hence, we select the mean value between the elbow point e and k as the appropriate order k.

In particular, denote $d\_intra(t-1) = intra(\mathcal{C}^{(t)}) - intra(\mathcal{C}^{(t-1)})$, we start from $k = 1$ and increment it by 1 iteratively. In each iteration t, we obtain the cluster partition $\mathcal{C}^{(t)}$ by performing k-order ($k = t$) graph convolution and spectral clustering, and compute the ratio $\frac{Q_g^{(t)}}{Q_f^{(t)}}$. Once $d\_intra(t-1) > 0$ or $t > max\_iter$, we stop the iteration. Finally, we use the kneed toolkit [52] to learn the elbow point e from the curve of $\frac{Q_g}{Q_f}$, and set the chosen k as $\lceil \frac{e+t-1}{2} \rceil$, where $\lceil x \rceil$ computes the smallest integer that is greater than or equal to x.

As the same as AGC, we perform the classical spectral clustering method [50] on the filtered feature matrix $\bar{\boldsymbol{X}}^{(k)}$ to partition the nodes of $\mathcal{V}$ into m clusters.

5.2 Algorithm Procedure of IAGC and Time Complexity

The overall procedure of IAGC is shown in Algorithm 2. Denote by n the number of nodes, d the number of attributes, m the number of clusters, n_p the number of components to keep in PCA, and N the number of nonzero entries of the adjacency matrix $\boldsymbol{A}$. Note that for a sparse $\boldsymbol{A}$, $N << n^2$. In the initialization, the time complexity of performing PCA is $\mathcal{O}(d^2n + d^2n_p)$, the time complexity of computing $\boldsymbol{L}_s$ is $\mathcal{O}(N)$, and the time complexity of computing feature-aware node similarity matrix is $\mathcal{O}(n^2n_p)$. In each iteration, for a sparse $\boldsymbol{L}_s$, the time complexity of computing k-order graph convolution is $\mathcal{O}(Nn_pk)$, the time complexity of performing spectral clustering on filtered data is $\mathcal{O}(n^2n_p + n^2m)$, the time complexity of computing $intra(\mathcal{C})$ in each iteration is $\mathcal{O}(\frac{1}{m}n^2n_p)$, the time complexity of computing $\frac{Q_g}{Q_f}$ is $\mathcal{O}(nn_p^2)$. Hence, the time complexity of each iteration of IAGC is $\mathcal{O}(Nn_pk + n^2n_p + nn_p^2)$. According to [52], if a curve is composed of t points, the time complexity of computing elbow point is $\mathcal{O}(t^2)$. Therefore, if Algorithm 2 iterates t times, the overall time complexity of IAGC is $\mathcal{O}(d^2n + d^2n_p + Nn_pt^2 + n^2n_pt + nn_p^2t)$. Since n_p is usually much smaller than n, the overall time complexity of IAGC is simplified as $\mathcal{O}(d^2n + Nn_pt^2 + n^2n_pt)$. The time complexity of IAGC is a little higher than AGC, but it is still time efficient since it does not need to train the neural network parameters,

6 Experiments

In this section, we conduct extensive experiments on real attributed networks[1] to validate the effectiveness of our proposed methods

1. The source code and data are publicly available at https://github.com/Karenxt/AGCandIAGC-code.

TABLE 1:
Clustering Performance on Cora.

Methods	Input	Acc	NMI	F1	ARI
k-means	Feature	0.3465 ± 0.0519	0.1673 ± 0.0271	0.2542 ± 0.0263	0.0628 ± 0.0573
Spectral-f	Feature	0.3626 ± 0.0129	0.1509 ± 0.0023	0.2564 ± 0.0141	0.0732 ± 0.0452
Spectral-g	Graph	0.3419 ± 0.0628	0.1949 ± 0.0751	0.3017 ± 0.0619	0.1455 ± 0.0069
DeepWalk (*k*-means)	Graph	0.4674 ± 0.0446	0.3175 ± 0.0368	0.3806 ± 0.0475	0.2393 ± 0.0475
DeepWalk (spectral)	Graph	0.4940 ± 0.0006	0.3315 ± 0.0007	0.4014 ± 0.0005	0.2678 ± 0.0009
DNGR (*k*-means)	Graph	0.4924 ± 0.0283	0.3729 ± 0.0204	0.3729 ± 0.0135	0.2101 ± 0.0135
DNGR (spectral)	Graph	0.5134 ± 0.0041	0.3892 ± 0.0028	0.3921 ± 0.0067	0.2271 ± 0.0038
GraRep (*k*-means)	Graph	0.3532 ± 0.0105	0.2246 ± 0.0080	0.3221 ± 0.0077	0.1352 ± 0.0095
GraRep (spectral)	Graph	0.3686 ± 0.0036	0.2656 ± 0.0009	0.3298 ± 0.0002	0.1381 ± 0.0083
NEU (*k*-means)	Graph	0.4924 ± 0.0283	0.3729 ± 0.0204	0.3729 ± 0.0135	0.2101 ± 0.0135
NEU (spectral)	Graph	0.5601 ± 0.0127	0.3911 ± 0.0079	0.4682 ± 0.0086	0.2857 ± 0.0202
GAE (*k*-means)	Both	0.5325 ± 0.0206	0.4069 ± 0.0278	0.4197 ± 0.0265	0.3040 ± 0.0265
GAE (spectral)	Both	0.5731 ± 0.0003	0.4347 ± 0.0008	0.5839 ± 0.0004	0.3391 ± 0.0007
VGAE (*k*-means)	Both	0.5595 ± 0.0237	0.3845 ± 0.0298	0.4150 ± 0.0350	0.2985 ± 0.0350
VGAE (spectral)	Both	0.6132 ± 0.0001	0.4225 ± 0.0002	0.4865 ± 0.0001	0.3426 ± 0.0002
MGAE (*k*-means)	Both	0.6039 ± 0.0409	0.4181 ± 0.0129	0.3536 ± 0.0534	0.3603 ± 0.0308
MGAE (spectral)	Both	0.6343 ± 0.0171	0.4557 ± 0.0106	0.3801 ± 0.0133	0.3801 ± 0.0133
ARGE (*k*-means)	Both	0.6400 ± 0.0111	0.4490 ± 0.0090	0.6190 ± 0.0106	0.3758 ± 0.0106
ARGE (spectral)	Both	0.6664 ± 0.0002	0.4567 ± 0.0006	0.6394 ± 0.0003	0.4064 ± 0.0001
ARVGE (*k*-means)	Both	0.6380 ± 0.0257	0.4500 ± 0.0028	0.6270 ± 0.0186	0.3740 ± 0.0186
ARVGE (spectral)	Both	0.6230 ± 0.0039	0.4429 ± 0.0031	0.5895 ± 0.0024	0.3604 ± 0.0047
GMI (*k*-means)	Both	0.6337 ± 0.0015	0.4231 ± 0.0003	0.4506 ± 0.0010	0.3167 ± 0.0011
GMI (spectral)	Both	0.6667 ± 0.0325	0.4698 ± 0.0206	0.5995 ± 0.1134	0.3989 ± 0.0435
DAEGC	Both	0.6768 ± 0.0016	0.5194 ± 0.0001	0.6231 ± 0.0018	0.4053 ± 0.0087
SDCN	Both	0.6546 ± 0.0105	0.4710 ± 0.0082	0.5732 ± 0.0094	0.3918 ± 0.0024
SENet	Both	**0.7119 ± 0.0363**	**0.5408 ± 0.0097**	**0.6835 ± 0.0245**	**0.4834 ± 0.0158**
GCC	Both	0.6270 ± 0.0009	0.5006 ± 0.0008	0.5388 ± 0.0005	0.3743 ± 0.0026
MinCutPool	Both	0.6492 ± 0.0102	0.4522 ± 0.0141	0.4300 ± 0.0106	0.3528 ± 0.0127
SGCMC	Both	0.6650 ± 0.0105	0.5503 ± 0.0082	0.6236 ± 0.0094	0.4436 ± 0.0024
AGC (*k*-means)	Both	0.6643 ± 0.0197	0.5279 ± 0.0086	0.6541 ± 0.0214	0.4364 ± 0.0130
AGC (spectral)	Both	**0.6892 ± 0.0017**	**0.5368 ± 0.0042**	**0.6561 ± 0.0001**	**0.4478 ± 0.0002**
IAGC (*k*-means)	Both	**0.7093 ± 0.0160**	**0.5628 ± 0.0068**	**0.6889 ± 0.0180**	**0.4838 ± 0.0131**
IAGC (spectral)	Both	**0.7242 ± 0.0000**	**0.5566 ± 0.0000**	**0.6847 ± 0.0000**	**0.4927 ± 0.0000**

AGC and IAGC for attributed graph clustering.

6.1 Datasets

Cora: A citation network [14] where the nodes correspond to the publications and are connected by an undirected edge if one cites the other. There are 2708 nodes, 5429 edges in Cora, and each node is associated with a binary vector of 1433 dimensions. The nodes are classified into 7 categories.

Citeseer: A citation network [14] with 3327 nodes and 4732 edges. The nodes correspond to the publications with 3703 dimension feature vectors, and are classified into 6 categories.

Pubmed: A citation network [14] with 19717 nodes and 44338 edges. The nodes correspond to the publications with 500 dimension feature vectors, and are divided into 3 classes.

Wiki: A webpage network [11] where the nodes are webpages and are connected if one links the other. There are 2405 nodes and 17981 edges in Wiki. The nodes are represented by 4973 dimension feature vectors, and are divided into 17 classes.

Amazon Computers (Computer): A co-purchase graphs [53] where nodes represent goods and are connected if two goods are frequently bought together, node features are bag-of-words encoded product reviews, and class labels are given by the product category. There are 13752 nodes and 133289 edges in Computer. The nodes are represented by 767 dimension feature vectors, and are divided into 10 classes.

6.2 Baselines and Evaluation Metrics

We compare AGC with three kinds of methods. 1) Clustering methods that only use node features: *k*-means and spectral clustering (Spectral-f) that constructs a similarity matrix with the node features by linear kernel. 2) Graph embedding or clustering methods that only use graph structure: spectral clustering (Spectral-g) that takes the node adjacency matrix as the similarity matrix, DeepWalk [7], deep neural networks for graph representations (DNGR) [29], GraRep [26], and NEU [27]. 3) Graph embedding or clustering methods that utilize both node features and graph structure: GAE and VGAE [14], MGAE [15], ARGE and ARVGE [16], GMI [33], DAEGC [34], SDCN [18], SENet [38], GCC [41], MinCutPool [44], and SGCMC [43] with a single view (i.e., raw features). To evaluate the clustering performance, we adopt four widely used performance measures [51]: clustering accuracy (Acc), normalized mutual information (NMI), macro F1-score (F1), and adjusted rand index (ARI). The larger of them indicates better clustering performance.

6.3 Parameter Settings

For AGC, we set $max\_iter = 60$. For IAGC, we set $max\_iter = 60$ and $n_p = 100$. For other baselines, we follow the parameter settings in the original papers and carefully set the hyperparameters. In particular, for DeepWalk, the number of random walks is 10, the number of latent dimensions for each node is 128, and the path length of each random walk is 80. For DNGR, the autoencoder is of three layers with 512 neurons and 256 neurons in the hidden layers respectively. For GraRep, the dimension of node representation in each step is set as 16, and the maximum matrix transition step is set as 5. For NEU, the parameters λ_1 and λ_2 are set as 0.5 and 0.25 respectively, the network embedding is learnt by GraRep, and the maximum iteration number is set as 5. For GAE and VGAE, we construct encoders with a 32-neuron hidden layer and a 16-neuron embedding layer, and train the encoders for 200 iterations using the Adam optimizer

TABLE 2:
Clustering Performance on Citeseer.

Methods	Input	Acc	NMI	F1	ARI
k-means	Feature	0.3849 ± 0.0678	0.1702 ± 0.0621	0.3047 ± 0.0585	0.1184 ± 0.0509
Spectral-f	Feature	0.4623 ± 0.0112	0.2119 ± 0.0093	0.3370 ± 0.0104	0.2157 ± 0.0205
Spectral-g	Graph	0.2591 ± 0.0368	0.1184 ± 0.0211	0.2948 ± 0.0132	0.0135 ± 0.0109
DeepWalk (*k*-means)	Graph	0.3615 ± 0.0466	0.0966 ± 0.0275	0.2670 ± 0.0331	0.1011 ± 0.0331
DeepWalk (spectral)	Graph	0.3745 ± 0.0012	0.1110 ± 0.0006	0.2845 ± 0.0010	0.1097 ± 0.0007
DNGR (*k*-means)	Graph	0.3167 ± 0.0253	0.1791 ± 0.0177	0.4379 ± 0.0248	0.0415 ± 0.0314
DNGR (spectral)	Graph	0.3259 ± 0.0195	0.1802 ± 0.0203	0.4419 ± 0.0211	0.0526 ± 0.0211
GraRep (*k*-means)	Graph	0.4081 ± 0.0019	0.2381 ± 0.0007	0.3568 ± 0.0033	0.0932 ± 0.0006
GraRep (spectral)	Graph	0.4325 ± 0.0001	0.2591 ± 0.0001	0.3726 ± 0.0002	0.1097 ± 0.0001
NEU (*k*-means)	Graph	0.4346 ± 0.0103	0.2565 ± 0.0011	0.3897 ± 0.0102	0.1252 ± 0.0037
NEU (spectral)	Graph	0.4459 ± 0.0001	0.2641 ± 0.0001	0.4021 ± 0.0001	0.1296 ± 0.0001
GAE (*k*-means)	Both	0.4126 ± 0.0297	0.1834 ± 0.0270	0.2913 ± 0.0231	0.1293 ± 0.0231
GAE (spectral)	Both	0.4389 ± 0.0022	0.2136 ± 0.0033	0.3354 ± 0.0008	0.1437 ± 0.0024
VGAE (*k*-means)	Both	0.4438 ± 0.0357	0.2271 ± 0.0230	0.3188 ± 0.0147	0.1569 ± 0.0147
VGAE (spectral)	Both	0.4480 ± 0.0001	0.2513 ± 0.0001	0.3316 ± 0.0001	0.1304 ± 0.0001
MGAE (*k*-means)	Both	0.5426 ± 0.0002	0.2998 ± 0.0001	0.4217 ± 0.0001	0.2021 ± 0.0002
MGAE (spectral)	Both	0.6356 ± 0.0353	0.3975 ± 0.0001	0.5081 ± 0.0241	0.3949 ± 0.0225
ARGE (*k*-means)	Both	0.5730 ± 0.0041	0.3500 ± 0.0055	0.5460 ± 0.0083	0.3428 ± 0.0083
ARGE (spectral)	Both	0.6051 ± 0.0001	0.4058 ± 0.0001	0.5617 ± 0.0001	0.3897 ± 0.0002
ARVGE (*k*-means)	Both	0.5440 ± 0.0112	0.2610 ± 0.0151	0.5290 ± 0.0169	0.2561 ± 0.0169
ARVGE (spectral)	Both	0.6088 ± 0.0007	0.3082 ± 0.0001	0.5409 ± 0.0006	0.2994 ± 0.0003
GMI (*k*-means)	Both	0.5256 ± 0.0002	0.2865 ± 0.0001	0.4733 ± 0.0006	0.2098 ± 0.0002
GMI (spectral)	Both	0.6227 ± 0.0163	0.3703 ± 0.0192	0.6055 ± 0.0098	0.3568 ± 0.0199
DAEGC	Both	0.6676 ± 0.0123	0.3890 ± 0.0156	0.6210 ± 0.0023	0.3974 ± 0.0182
SDCN	Both	0.6574 ± 0.0040	0.3851 ± 0.0054	0.6207 ± 0.0024	0.4003 ± 0.0031
SENet	Both	**0.6727 ± 0.0059**	**0.4160 ± 0.0070**	**0.6350 ± 0.0044**	**0.4208 ± 0.0081**
GCC	Both	**0.6924 ± 0.0034**	**0.4352 ± 0.0026**	**0.6450 ± 0.0033**	**0.4518 ± 0.0031**
MinCutPool	Both	0.5612 ± 0.0074	0.3293 ± 0.0041	0.5409 ± 0.0079	0.3030 ± 0.0068
SGCMC	Both	0.6467 ± 0.0087	0.3852 ± 0.0096	0.4902 ± 0.0026	0.3689 ± 0.0040
AGC (*k*-means)	Both	0.5441 ± 0.0200	0.3222 ± 0.0125	0.5204 ± 0.0301	0.2274 ± 0.0157
AGC (spectral)	Both	**0.6700 ± 0.0024**	**0.4113 ± 0.0036**	**0.6248 ± 0.0019**	**0.4158 ± 0.0002**
IAGC (*k*-means)	Both	0.6281 ± 0.0146	0.4056 ± 0.0043	0.6172 ± 0.0057	0.3724 ± 0.0132
IAGC (spectral)	Both	**0.6907 ± 0.0001**	**0.4308 ± 0.0002**	**0.6471 ± 0.0002**	**0.4432 ± 0.0001**

with learning rate 0.01. For MGAE, the corruption level p is 0.4, the number of layers is 3, and the parameter λ is 10^{-5}. For ARGE and ARVGE, we construct encoders with a 32-neuron hidden layer and a 16-neuron embedding layer. The discriminators are built by two hidden layers with 16 neurons and 64 neurons respectively. On *Pubmed*, we train the autoencoder-related models of ARGE and ARVGE for 2000 iterations with the encoder learning rate 0.001 and the discriminator learning rate 0.008; On the other datasets, we train the models for 200 iterations with the Adam optimizer, with the encoder learning rate and the discriminator learning rate both as 0.001. For GMI, the encoder is with a graph convolutional layer for Citeseer and Pubmed, and is with two graph convolutional layers for the other datasets, and the dimension of each layer is 512. The model is trained for 500 epochs with the Adam learning rate 0.001. For DAEGC, the clustering coefficient γ is set as 10, the encoder is constructed with a 256-neuron hidden layer and a 16-neuron embedding layer, and is trained for 200 epochs with Adam learning rate 0.001. For SDCN, α and β are set as 0.1 and 0.01 respectively, the autoencoder is pretrained by using all the data points with 30 epochs, and the learning rate is 0.001, the dimension of the autoencoder and GCN module is set to d-500-500-2000-10, where d is the dimension of the input data. The whole SDCN model is trained for 200 epochs with Adam learning rate 0.0001. For SENet, the embedding network is with two 16-neuron hidden layers and is trained for 50 epochs with the Adam learning rate 0.03, the trade-off hyperparameter λ is set as 1. For GCC, the propagation order for Cora, Citeseer, Pubmed, Wiki and Computer are 33, 17, 150, 13, 12, respectively, and the tolerance as well as maximum number of iterations are set to 10^{-7} and 150. For MinCutPool, the network is with six 64-neuron hidden layers and trained for 10000 iterations with the Adam learning rate 5e-3. Residual connections connect the second and fourth hidden layers with the first hidden layer. For SGCMC, the graph attention auto-encoder and the self-supervised learning module are with three and two 512-neuron hidden layers, respectively, the learning rate is set to 3×10^{-5}

We set the number of clusters equal to the true number of classes for all the methods. For our proposed methods and the graph embedding baselines such as DeepWalk, DNGR, GraRep, NEU, GAE, VGAE, MGAE, ARGE, ARVGE and GMI, we perform both *k*-means and spectral clustering on the learned node features or embeddings to make a fair comparison.

6.4 Result Analysis

We run each method 10 times for each dataset and report the average clustering results and standard deviations in Tables 1-5, where the top 4 results are highlighted in bold. Note that to make a fair comparison, we compare the graph embedding baselines with our proposed methods which adopt the same clustering method. The observations are as follows.

1) AGC and IAGC consistently outperform the embedding and clustering methods that only exploit either node features or graph structure by a very large margin, due to the clear reason that AGC and IAGC make a better use of available data by integrating both kinds of information, which can complement each other and greatly improve clustering performance.

2) AGC and IAGC generally outperform existing attributed graph embedding and clustering methods that use both node features and graph structure. This is because that AGC and IAGC can better utilize graph information than these methods. In particular, the baselines only take into account neighbours within a fixed hop to aggregate information. In contrast, AGC and IAGC use k-order

TABLE 3:
Clustering Performance on Pubmed.

Methods	Input	Acc	NMI	F1	ARI
k-means	Feature	0.5732 ± 0.0381	0.2912 ± 0.0352	0.5735 ± 0.0346	0.2505 ± 0.0346
Spectral-f	Feature	0.5991 ± 0.0229	0.3255 ± 0.0075	0.5861 ± 0.0001	0.2177 ± 0.0260
Spectral-g	Graph	0.4322 ± 0.0241	0.0786 ± 0.0093	0.5045 ± 0.0038	0.0385 ± 0.0017
DeepWalk (*k*-means)	Graph	0.6186 ± 0.0206	0.1671 ± 0.0237	0.4706 ± 0.0254	0.1809 ± 0.0254
DeepWalk (spectral)	Graph	0.6478 ± 0.0001	0.1833 ± 0.0001	0.5027 ± 0.0001	0.2017 ± 0.0002
DNGR (*k*-means)	Graph	0.4535 ± 0.0169	0.1538 ± 0.0182	0.1790 ± 0.0203	0.0006 ± 0.0203
DNGR (spectral)	Graph	0.4864 ± 0.0058	0.1890 ± 0.0091	0.2036 ± 0.0133	0.0010 ± 0.0327
GraRep (*k*-means)	Graph	0.5419 ± 0.0002	0.1532 ± 0.0001	0.5423 ± 0.0001	0.1161 ± 0.0002
GraRep (spectral)	Graph	0.5963 ± 0.0003	0.1760 ± 0.0001	0.5762 ± 0.0001	0.1937 ± 0.0004
NEU (*k*-means)	Graph	0.5833 ± 0.0001	0.1905 ± 0.0001	0.5892 ± 0.0001	0.1511 ± 0.0001
NEU (spectral)	Graph	0.6190 ± 0.0001	0.2141 ± 0.0001	0.6085 ± 0.0001	0.2157 ± 0.0001
GAE (*k*-means)	Both	0.6408 ± 0.0166	0.2297 ± 0.0197	0.4926 ± 0.0149	0.2225 ± 0.0149
GAE (spectral)	Both	0.6590 ± 0.0001	0.2479 ± 0.0001	0.5294 ± 0.0002	0.2409 ± 0.0001
VGAE (*k*-means)	Both	0.6548 ± 0.0173	0.2509 ± 0.0207	0.5095 ± 0.0186	0.2456 ± 0.0186
VGAE (spectral)	Both	0.6678 ± 0.0001	0.2678 ± 0.0002	0.5238 ± 0.0002	0.2575 ± 0.0001
MGAE (*k*-means)	Both	0.4013 ± 0.0002	0.0678 ± 0.0003	0.3839 ± 0.0001	0.0139 ± 0.0002
MGAE (spectral)	Both	0.4388 ± 0.0183	0.0816 ± 0.0259	0.4198 ± 0.0236	0.0398 ± 0.0236
ARGE (*k*-means)	Both	0.5912 ± 0.0363	0.2317 ± 0.0285	0.5841 ± 0.0350	0.2032 ± 0.0350
ARGE (spectral)	Both	0.6134 ± 0.0004	0.2466 ± 0.0003	0.6009 ± 0.0004	0.2171 ± 0.0004
ARVGE (*k*-means)	Both	0.5822 ± 0.0100	0.2062 ± 0.0057	0.5311 ± 0.0082	0.2045 ± 0.0065
ARVGE (spectral)	Both	0.5633 ± 0.0001	0.1786 ± 0.0002	0.4905 ± 0.0002	0.1651 ± 0.0001
GMI (*k*-means)	Both	0.6160 ± 0.0001	0.2554 ± 0.0001	0.5239 ± 0.0001	0.2470 ± 0.0002
GMI (spectral)	Both	0.6481 ± 0.0042	0.2875 ± 0.0247	0.5410 ± 0.0102	0.2604 ± 0.0180
DAEGC	Both	0.6706 ± 0.0104	0.2663 ± 0.0137	0.6585 ± 0.0109	0.2768 ± 0.0120
SDCN	Both	0.6482 ± 0.0018	0.2862 ± 0.0045	0.6378 ± 0.0061	0.2573 ± 0.0020
SENet	Both	0.6673 ± 0.0139	0.3003 ± 0.0067	0.6617 ± 0.0158	0.2886 ± 0.0097
GCC	Both	**0.7020 ± 0.0000**	**0.3104 ± 0.0000**	**0.6902 ± 0.0000**	**0.3225 ± 0.0000**
MinCutPool	Both	0.6210 ± 0.0016	0.2606 ± 0.0001	0.5346 ± 0.0018	0.2556 ± 0.0087
SGCMC	Both	0.6850 ± 0.0026	0.2986 ± 0.0065	0.5849 ± 0.0046	0.2861 ± 0.0054
AGC (*k*-means)	Both	0.6871 ± 0.0064	0.3012 ± 0.0052	0.6806 ± 0.0076	0.3010 ± 0.0061
AGC (spectral)	Both	**0.6978 ± 0.0000**	**0.3159 ± 0.0000**	**0.6872 ± 0.0000**	**0.3098 ± 0.0000**
IAGC (*k*-means)	Both	**0.6919 ± 0.0001**	**0.3036 ± 0.0002**	**0.6842 ± 0.0001**	**0.3100 ± 0.0003**
IAGC (spectral)	Both	**0.7046 ± 0.0001**	**0.3160 ± 0.0001**	**0.6979 ± 0.0001**	**0.3296 ± 0.0000**

graph convolution with an automatically selected k to aggregate information within k-hop neighbourhood to produce better feature representations for clustering.

3) AGC and IAGC outperform the strong baselines MGAE (spectral) and GCC by a considerable margin on *Cora*, *Citeseer*, *Pubmed* and *Computer*, but perform worse than them on *Wiki*. This is probably because *Wiki* is more densely connected than other datasets and aggregating information within 3-hop neighbourhood may be enough for feature smoothing. But it is not good enough for sparser networks such as *Cora* and *Computer*, on which the performance gaps between our proposed methods and the strong baselines are wider. It shows that AGC and IAGC can better deal with the diversity of networks via adaptively selecting a good k for different networks.

4) AGC outperforms the strong baseline SENet by a considerable margin on *Pubmed*, *Wiki* and *Computer*, but performs worse than SENet on *Cora* and *Citeseer*, because SENet adopts spectral clustering loss to constrain the node embeddings in each layer to capture the global cluster structure, and concatenates the embeddings learned from different layers, making both global cluster structure information and different neighbourhood information encoded into node embeddings, which is helpful for improving the clustering performance.

5) AGC (*k*-means) and IAGC (*k*-means) are comparable to AGC (spectral) and IAGC (spectral) respectively on *Cora*, *Pubmed*, *Wiki* and *Computer*, which verifies the effectiveness of the low-pass graph filters of AGC and IAGC. One exception is that AGC (*k*-means) and IAGC (*k*-means) perform worse than AGC (spectral) and IAGC (spectral) respectively on *Citeseer*, possible reason is that *k*-means is not suitable to cluster the filtered features of *Citeseer*, since *k*-means assumes the resulting clusters form convex sets, whereas spectral clustering does not make strong assumptions on the form of the clusters [50]. The reason also explains why AGC (*k*-means) and IAGC (*k*-means) perform worse than the attributed graph clustering methods such as DAEGC, SDCN, SENet and GCC on *Citeseer*.

6) IAGC performs better than AGC, because IAGC selects the order k from two aspects. One aspect is to observe the variation of intra-cluster distance, which can avoid over-smoothing that causes rapid performance decline. The other aspect is to observe the inconsistencies of filtered features with graph structure and raw features respectively, which can learn an elbow point before the performance gradually declines. By considering both aspects, IAGC can choose a smaller and more precise order k than AGC.

6.5 Efficiency Comparison

The running time of the baselines and our proposed methods (in Python, with a NVIDIA GeForce RTX 3090 GPU) on *Cora*, *Citeseer*, *Pubmed*, *Wiki* and *Computer* is shown in Fig. 4 and Fig. 5. Note that for our proposed methods and the graph embedding baselines such as DeepWalk, DNGR, GraRep, NEU, GAE, VGAE, MGAE, ARGE, ARVGE and GMI, we perform spectral clustering on the learned node features or embeddings.

The following observations can be made. 1) As shown in Fig. 4, AGC and IAGC are comparable with the classical methods such as DeepWalk, DNGR, GraRep and NEU that only use node features or graph structure, since they do not need to train the neural network parameters. 2) As shown in Fig. 5, AGC and MGAE are two most fast methods for attributed graph clustering, followed by IAGC. AGC and IAGC do not need back propagation to train the neural network parameters, and hence are more time efficient even with a relatively large k.

TABLE 4:
Clustering Performance on Wiki.

Methods	Input	Acc	NMI	F1	ARI
k-means	Feature	0.3337 ± 0.0216	0.3020 ± 0.0180	0.2451 ± 0.0225	0.0347 ± 0.0127
Spectral-f	Feature	0.4128 ± 0.0171	0.4399 ± 0.0128	0.2520 ± 0.0001	0.0521 ± 0.0244
Spectral-g	Graph	0.2358 ± 0.0360	0.1928 ± 0.0311	0.1721 ± 0.0298	0.0354 ± 0.0103
DeepWalk (*k*-means)	Graph	0.3846 ± 0.0162	0.3238 ± 0.0137	0.2574 ± 0.0119	0.0571 ± 0.0124
DeepWalk (spectral)	Graph	0.3945 ± 0.0098	0.3403 ± 0.0057	0.2735 ± 0.0055	0.0936 ± 0.0062
DNGR (*k*-means)	Graph	0.3758 ± 0.0122	0.3585 ± 0.0091	0.2538 ± 0.0103	0.0516 ± 0.0310
DNGR (spectral)	Graph	0.3988 ± 0.0265	0.3657 ± 0.0143	0.2864 ± 0.0241	0.0810 ± 0.0293
GraRep (*k*-means)	Graph	0.1928 ± 0.0038	0.1328 ± 0.0036	0.1743 ± 0.0052	0.0465 ± 0.0020
GraRep (spectral)	Graph	0.2440 ± 0.0034	0.1750 ± 0.0030	0.2206 ± 0.0040	0.0729 ± 0.0023
NEU (*k*-means)	Graph	0.3386 ± 0.0065	0.2963 ± 0.0026	0.2865 ± 0.0078	0.0927 ± 0.0045
NEU (spectral)	Graph	0.3648 ± 0.0136	0.3136 ± 0.0057	0.3005 ± 0.0159	0.1193 ± 0.0069
GAE (*k*-means)	Both	0.1733 ± 0.0167	0.1193 ± 0.0132	0.1535 ± 0.0145	0.0121 ± 0.0143
GAE (spectral)	Both	0.1799 ± 0.0027	0.0888 ± 0.0026	0.0776 ± 0.0017	0.0316 ± 0.0021
VGAE (*k*-means)	Both	0.2867 ± 0.0231	0.3028 ± 0.0219	0.2049 ± 0.0198	0.0343 ± 0.0224
VGAE (spectral)	Both	0.4039 ± 0.0040	0.3463 ± 0.0020	0.3022 ± 0.0063	0.0910 ± 0.0034
MGAE (*k*-means)	Both	0.3981 ± 0.0177	0.4034 ± 0.0150	0.3355 ± 0.0229	0.1435 ± 0.0104
MGAE (spectral)	Both	**0.5014 ± 0.0190**	**0.4797 ± 0.0179**	0.3920 ± 0.0169	**0.2205 ± 0.0147**
ARGE (*k*-means)	Both	0.4140 ± 0.0032	0.3950 ± 0.0073	0.3827 ± 0.0084	0.1037 ± 0.0064
ARGE (spectral)	Both	0.4206 ± 0.0090	0.4183 ± 0.0041	0.3903 ± 0.0070	0.1096 ± 0.0129
ARVGE (*k*-means)	Both	0.4155 ± 0.0180	0.4001 ± 0.0193	0.3780 ± 0.0138	0.1051 ± 0.0104
ARVGE (spectral)	Both	0.4101 ± 0.0010	0.4060 ± 0.0006	0.3875 ± 0.0005	0.1032 ± 0.0010
GMI (*k*-means)	Both	0.4193 ± 0.0042	0.3656 ± 0.0073	0.3430 ± 0.0057	0.1054 ± 0.0031
GMI (spectral)	Both	0.4491 ± 0.0052	0.3894 ± 0.0075	0.3677 ± 0.0084	0.1237 ± 0.0055
DAEGC	Both	0.4312 ± 0.0080	0.3827 ± 0.0035	0.3626 ± 0.0066	0.1153 ± 0.0042
SDCN	Both	0.4147 ± 0.0063	0.3792 ± 0.0083	0.3514 ± 0.0069	0.1085 ± 0.0063
SENet	Both	0.4023 ± 0.0107	0.3852 ± 0.0061	0.3543 ± 0.0087	0.1179 ± 0.0025
GCC	Both	**0.5460 ± 0.0196**	**0.5103 ± 0.0062**	**0.4453 ± 0.0214**	**0.3026 ± 0.0142**
MinCutPool	Both	0.4207 ± 0.0016	0.3661 ± 0.0001	0.3368 ± 0.0018	0.1235 ± 0.0087
SGCMC	Both	0.4128 ± 0.0176	0.4083 ± 0.0131	0.3887 ± 0.0103	0.2106 ± 0.0127
AGC (*k*-means)	Both	0.4784 ± 0.0133	0.4366 ± 0.0067	0.3986 ± 0.0104	0.2217 ± 0.0129
AGC (spectral)	Both	0.4765 ± 0.0079	**0.4528 ± 0.0017**	**0.4036 ± 0.0020**	0.1452 ± 0.0031
IAGC (*k*-means)	Both	**0.4814 ± 0.0154**	0.4467 ± 0.0039	**0.4088 ± 0.0107**	**0.2872 ± 0.0106**
IAGC (spectral)	Both	**0.4984 ± 0.0085**	**0.4622 ± 0.0015**	**0.4283 ± 0.0038**	**0.2441 ± 0.0025**

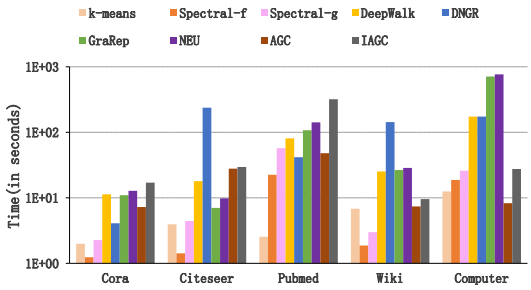


Fig. 4: Running time compared with classical embedding or clustering methods.

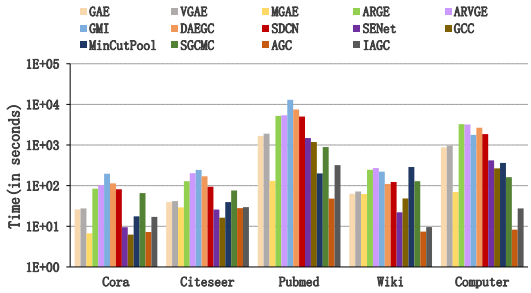


Fig. 5: Running time compared with attributed graph embedding or clustering methods.

6.6 Validity of Selection Strategies

To demonstrate the validity of the proposed selection criterion $d\_intra(k) > 0$ and the improved selection criterion which also considers the elbow point e from the curve of $\frac{Q_g}{Q_f}$, we plot $d\_intra(k)$ and $\frac{Q_g}{Q_f}$ as well as the clustering performance of IAGC (spectral) w.r.t. k in Fig. 6.

From Fig. 6, it can be seen that when $d\_intra(k) > 0$, the clustering performance has begun to decline rapidly, which indicates that a better performance exists before $d\_intra(k) > 0$. Although the selection criterion $d\_intra(k) > 0$ is not sensitive to slight downward trend, e.g., it does not find the first local minimum of $intra(\mathcal{C})$ in Citeseer and Pubmed, it can reliably prevent over-smoothing and find a good cluster partition. By observing the curve of $\frac{Q_g}{Q_f}$, it can be seen that the performance is increasing at the elbow point and is decreasing when the curve flattens, which means that the best performance is after the elbow point. According to the experimental results in Fig. 6, the best order k^* is indeed between the elbow point e and k that makes $d\_intra(k) > 0$ on *Cora*, *Citeseer*, *Pubmed* and *Wiki*, which are 12, 35, 46 and 7, respectively. One exception is on *Computer*, the performance increases again after $d\_intra(k) > 0$, causing the best order $k^* = 39$ which is larger than $k = 24$.

In the improved selection criterion, the selected order k' is $\lceil \frac{e+t-1}{2} \rceil$, where $d\_intra(t-1) > 0$, i.e., $k' = \lceil \frac{e+k}{2} \rceil (k = t-1)$. In particular, k' on these datasets are 21, 34, 35, 11, and 16, which are close to the corresponding k^* on *Cora*, *Citeseer*, *Pubmed* and *Wiki*, demonstrating the effectiveness of the improved criterion.

6.7 Parameter Investigation

Since the best order k^* can be very different for diverse attributed graphs (the best k^* on *Cora*, *Citeseer*, *Pubmed*, *Wiki*, and *Computer* are 12, 35, 46, 7, and 39 respectively), we simply set $max\_iter = 60$ to allow for a large enough search space for AGC and IAGC. For the parameter n_p in IAGC, we adopt grid searching to investigate the impact of n_p on the clustering performance. In particular, we set $n_p = \{50, 100, 150, 200\}$, then run IAGC

TABLE 5:
Clustering Performance on Computer.

Methods	Input	Acc	NMI	F1	ARI
k-means	Feature	0.2619 ± 0.0188	0.1242 ± 0.0172	0.1826 ± 0.0133	0.0658 ± 0.0095
Spectral-f	Feature	0.3048 ± 0.0212	0.1499 ± 0.0128	0.1820 ± 0.0186	0.0705 ± 0.0145
Spectral-g	Graph	0.2628 ± 0.0386	0.1267 ± 0.0181	0.1531 ± 0.0128	0.0684 ± 0.0035
DeepWalk (k-means)	Graph	0.3151 ± 0.0853	0.1549 ± 0.0954	0.2317 ± 0.0365	0.0872 ± 0.0094
DeepWalk (spectral)	Graph	0.3209 ± 0.0237	0.1646 ± 0.0641	0.2311 ± 0.0286	0.1051 ± 0.0235
DNGR (k-means)	Graph	0.3179 ± 0.0503	0.1602 ± 0.0213	0.2539 ± 0.0362	0.0957 ± 0.0061
DNGR (spectral)	Graph	0.3360 ± 0.0417	0.2039 ± 0.0570	0.2788 ± 0.0431	0.1203 ± 0.0382
GraRep (k-means)	Graph	0.3321 ± 0.0010	0.2351 ± 0.0001	0.2797 ± 0.0003	0.1576 ± 0.0002
GraRep (spectral)	Graph	0.3300 ± 0.0078	0.2437 ± 0.0083	0.2740 ± 0.0043	0.1613 ± 0.0089
NEU (k-means)	Graph	0.4211 ± 0.0006	0.2802 ± 0.0003	0.3459 ± 0.0014	0.1513 ± 0.0011
NEU (spectral)	Graph	0.4497 ± 0.0151	0.2843 ± 0.0051	0.3477 ± 0.0031	0.1572 ± 0.0260
GAE (k-means)	Both	0.3549 ± 0.0179	0.2216 ± 0.0225	0.2604 ± 0.0397	0.1118 ± 0.0397
GAE (spectral)	Both	0.4011 ± 0.0015	0.2664 ± 0.0004	0.2005 ± 0.0001	0.1908 ± 0.0023
VGAE (k-means)	Both	0.3561 ± 0.0083	0.2202 ± 0.0027	0.2597 ± 0.0280	0.1037 ± 0.0280
VGAE (spectral)	Both	0.4205 ± 0.0022	0.2504 ± 0.0013	0.2260 ± 0.0032	0.1514 ± 0.0045
MGAE (k-means)	Both	0.3123 ± 0.0078	0.1639 ± 0.0024	0.2405 ± 0.0006	0.1254 ± 0.0028
MGAE (spectral)	Both	0.4633 ± 0.0184	0.4239 ± 0.0168	0.3775 ± 0.0238	0.1794 ± 0.0121
ARGE (k-means)	Both	0.3776 ± 0.0140	0.2516 ± 0.0046	0.2950 ± 0.0052	0.1381 ± 0.0052
ARGE (spectral)	Both	0.4214 ± 0.0020	0.2666 ± 0.0006	0.2834 ± 0.0059	0.1760 ± 0.0024
ARVGE (k-means)	Both	0.3618 ± 0.0247	0.2436 ± 0.0141	0.2807 ± 0.0061	0.1304 ± 0.0061
ARVGE (spectral)	Both	0.4078 ± 0.0021	0.2732 ± 0.0012	0.3103 ± 0.0012	0.1519 ± 0.0028
GMI (k-means)	Both	0.3938 ± 0.0033	0.2636 ± 0.0003	0.3157 ± 0.0007	0.1659 ± 0.0004
GMI (spectral)	Both	0.4536 ± 0.0002	0.4017 ± 0.0003	0.3508 ± 0.0002	0.1788 ± 0.0002
DAEGC	Both	0.4713 ± 0.0006	0.4115 ± 0.0004	0.3826 ± 0.0002	0.1953 ± 0.0034
SDCN	Both	0.4523 ± 0.0017	0.3982 ± 0.0009	0.3712 ± 0.0006	0.1663 ± 0.0008
SENet	Both	0.3767 ± 0.0998	0.2393 ± 0.0775	0.2568 ± 0.0586	0.1213 ± 0.0942
GCC	Both	0.4494 ± 0.0054	0.3708 ± 0.0064	0.3975 ± 0.0041	0.1725 ± 0.0079
MinCutPool	Both	0.3568 ± 0.0036	0.2649 ± 0.0041	0.2271 ± 0.0028	0.1053 ± 0.0037
SGCMC	Both	0.4169 ± 0.0049	0.2970 ± 0.0051	0.3425 ± 0.0086	0.1537 ± 0.0071
AGC (k-means)	Both	**0.4978 ± 0.0297**	**0.4373 ± 0.0171**	**0.4073 ± 0.0387**	**0.2800 ± 0.0252**
AGC (spectral)	Both	**0.4851 ± 0.0020**	**0.4304 ± 0.0006**	**0.4581 ± 0.0014**	**0.2030 ± 0.0012**
IAGC (k-means)	Both	**0.5091 ± 0.0038**	**0.4496 ± 0.0069**	**0.4203 ± 0.0104**	**0.3076 ± 0.0118**
IAGC (spectral)	Both	**0.5824 ± 0.0001**	**0.4485 ± 0.0000**	**0.4664 ± 0.0000**	**0.3447 ± 0.0001**

TABLE 6: Acc of IAGC w.r.t. n_p.

	Cora	Citeseer	Pubmed	Wiki	Computer
n_p=50	0.7246	0.6908	0.7044	0.4968	0.5863
n_p=100	0.7242	0.6907	0.7046	0.4984	0.5824
n_p=150	0.7234	0.6917	0.7047	0.5019	0.5838
n_p=200	0.7256	0.6911	0.7047	0.5015	0.5840

(spectral) 10 times and show the average Acc in Table 6. It can be seen that the performance of IAGC remains relatively stable when $n_p = \{50, 100, 150, 200\}$, so we simply set $n_p = 100$ for IAGC.

6.8 Ablation Study

IAGC performs PCA on raw node features for dimension reduction, in order to avoid the "curse of dimensionality" problem and compute a more accurate feature-aware node similarity matrix.

To verify the effectiveness of PCA, we additionally test IAGC (spectral) without PCA (denoted as IAGC w/o PCA) and AGC (spectral) with PCA (denoted as AGC+PCA). We run both methods for 10 times and show the average Acc in Table 7. It can be seen that 1) IAGC performs better than IAGC w/o PCA, which verifies the necessity of performing PCA. 2) AGC+PCA improves upon AGC, but performs worse than IAGC, showing that PCA can produce better node features, and the improved selection strategy can further boost the clustering performance.

TABLE 7: Effectiveness of PCA.

	Cora	Citeseer	Pubmed	Wiki	Computer
IAGC w/o PCA	0.6816	0.6800	0.6921	0.4800	0.4818
AGC+PCA	0.7141	0.6835	0.7011	0.4820	0.5447
AGC	0.6892	0.6700	0.6978	0.4765	0.4851
IAGC	0.7242	0.6907	0.7046	0.4984	0.5824

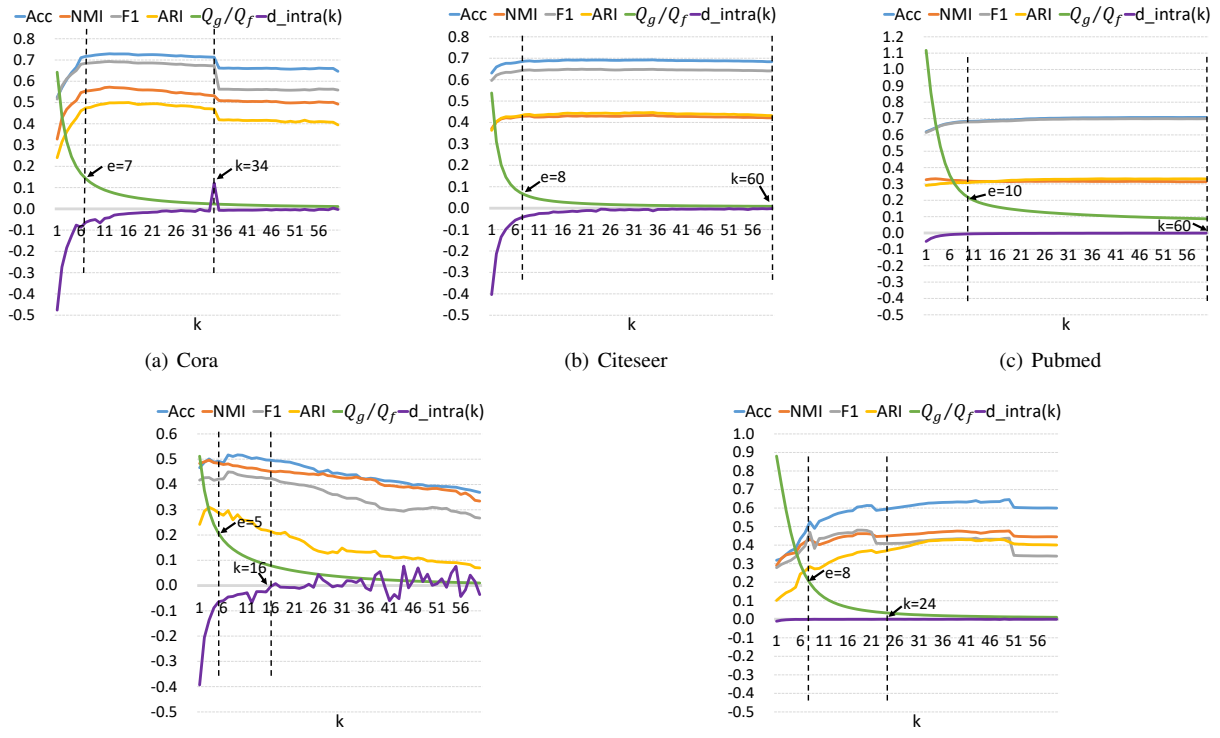
6.9 Results on OGB

To evaluate AGC and IAGC on large-scale datasets, we perform AGC and IAGC on ogbn-arxiv and ogbn-products [54] with a NVIDIA GeForce RTX 3090 GPU. IAGC requires to compute feature-aware node similarity matrix which takes a lot of memory due to dense matrix computations, so it is difficult to perform IAGC on large-scale graphs. We also try to compare our methods with the GNN based methods that specifically designed for node clustering (i.e., DAEGC, SDCN, SENet, GCC, MinCutPool, SGCMC). Note that DAEGC, SENet and SGCMC run out of memory on these two datasets, since they need to compute attention coefficient matrix or similarity matrix for all the nodes with dense matrix computations. Finally, we only show the clustering performance of SDCN, GCC, MinCutPool and AGC in Table 8. As spectral clustering requires the computation of similarity matrix and eigen-decomposition, which results in out of memory, we use k-means instead of spectral clustering in AGC.

Due to space limitation, we only report the NMI value of these methods. The result of GCC on ogbn-products is not given, since it uses the whole feature matrix to update parameters and runs out of memory. It can be seen that AGC performs better than SDCN, GCC and MinCutPool, showing that AGC can effectively select appropriate order k for different graphs and properly apply k-order graph convolution to improve clustering performance.

TABLE 8: Performance on ogbn-arxiv and ogbn-products.

	SDCN	GCC	MinCutPool	AGC
ogbn-arxiv	0.2614	0.4118	0.3584	0.4596
ogbn-products	0.2853	-	0.3630	0.4550



(a) Cora (b) Citeseer (c) Pubmed

(d) Wiki (e) Computer

Fig. 6: Values of $d\_intra(k)$ and $\frac{Q_g}{Q_f}$ and the clustering performance of IAGC w.r.t. k on five attributed graph datasets.

7 Conclusion

We have proposed two simple and effective adaptive graph convolution methods AGC and IAGC for attributed graph clustering. To fully exploit attributed graph data and capture global cluster structures, we design a new k-order graph convolution to aggregate long-range neighborhood relations. Theoretical analysis shows the effectiveness of k-order graph convolution in producing smoother node representations for the downstream clustering task. To optimize clustering performance on different graphs, we design two strategies for adaptively selecting an appropriate order k. AGC observes the variation of intra-cluster distance to choose k, which can find a reliable k to avoid over-smoothing and yield a good cluster partition. IAGC observes both the variation of intra-cluster distance and the inconsistencies of filtered features with graph structure and raw features respectively, which can find the order k closer to the best one. Experimental results demonstrate that our methods achieve competitive performance compared to classical and state-of-the-art methods. A limitation of our selection strategies is that the search of order k needs to start from 0, which increases the time complexity of the proposed methods. In future work, we plan to explore a more efficient way to search for k.

Acknowledgments

This work is supported by National Natural Science Foundation of China (No. 62206038, No. 62106035, No. 61876028), the Fundamental Research Funds for the Central Universities (No. DUT20RC(3)066, No. DUT20RC(3)040), and the General Research Fund No.15222220 funded by the UGC of Hong Kong.

References

[1] H. Cai, V. W. Zheng, and K. C. Chang, "A comprehensive survey of graph embedding: Problems, techniques, and applications," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 9, pp. 1616–1637, 2018.

[2] M. Kim and J. Leskovec, "Modeling social networks with node attributes using the multiplicative attribute graph model," in *UAI*, 2011, pp. 400–409.

[3] Y. Li, C. Sha, X. Huang, and Y. Zhang, "Community detection in attributed graphs: an embedding approach," in *AAAI*, 2018, pp. 338–345.

[4] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *NeurIPS*, 2017, pp. 1024–1034.

[5] S. E. Schaeffer, "Graph clustering," *Computer Science Review*, vol. 1, no. 1, pp. 27–64, 2007.

[6] M. E. Newman, "Finding community structure in networks using the eigenvectors of matrices," *Physical Review E*, vol. 74, no. 3, p. 036104, 2006.

[7] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *KDD*, 2014, pp. 701–710.

[8] D. Wang, P. Cui, and W. Zhu, "Structural deep network embedding," in *KDD*, 2016, pp. 1225–1234.

[9] J. D. Chang and D. M. Blei, "Relational topic models for document networks," in *AISTATS*, ser. JMLR Proceedings, vol. 5, pp. 81–88.

[10] R. Xia, Y. Pan, L. Du, and J. Yin, "Robust multi-view spectral clustering via low-rank and sparse decomposition," in *AAAI*, 2014, pp. 2149–2155.

[11] C. Yang, Z. Liu, D. Zhao, M. Sun, and E. Y. Chang, "Network representation learning with rich text information," in *IJCAI*, 2015, pp. 2111–2117.

[12] X. Wang, D. Jin, X. Cao, L. Yang, and W. Zhang, "Semantic community identification in large attribute networks," in *AAAI*, 2016, pp. 265–271.

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

[14] ——, "Variational graph auto-encoders," in *NeurIPS Workshop on Bayesian Deep Learning*, 2016.

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

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

[17] P. Velickovic, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm, "Deep graph infomax," in *ICLR*, 2019.

[18] D. Bo, X. Wang, C. Shi, M. Zhu, E. Lu, and P. Cui, "Structural deep clustering network," in *WWW*, 2020, pp. 1400–1410.

[19] D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Schölkopf, "Learning with local and global consistency," in *NeurIPS*, 2003, pp. 321–328.

[20] F. Wang, C. Zhang, and T. Li, "Clustering with local and global regularization," *IEEE Trans. Knowl. Data Eng.*, vol. 21, no. 12, pp. 1665–1678, 2009.

[21] J. Han, H. Liu, and F. Nie, "A local and global discriminative framework and optimization for balanced clustering," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 10, pp. 3059–3071, 2019.
[22] X. Zhang, H. Liu, Q. Li, and X. Wu, "Attributed graph clustering via adaptive graph convolution," in *IJCAI*, 2019, pp. 4327–4333.
[23] H. Li, Z. Bu, Z. Wang, and J. Cao, "Dynamical clustering in electronic commerce systems via optimization and leadership expansion," *IEEE Trans. Ind. Informatics*, vol. 16, no. 8, pp. 5327–5334, 2020.
[24] H.-J. Li, W. Xu, S. Song, W.-X. Wang, and M. Perc, "The dynamics of epidemic spreading on signed networks," *Chaos, Solitons & Fractals*, vol. 151, p. 111294, 2021.
[25] D. Berberidis and G. B. Giannakis, "Node embedding with adaptive similarities for scalable learning over graphs," *IEEE Trans. Knowl. Data Eng.*, vol. 33, no. 2, pp. 637–650, 2021.
[26] S. Cao, W. Lu, and Q. Xu, "Grarep: Learning graph representations with global structural information," in *CIKM*, 2015, pp. 891–900.
[27] C. Yang, M. Sun, Z. Liu, and C. Tu, "Fast network embedding enhancement via high order proximity approximation," in *IJCAI*, 2017, pp. 3894–3900.
[28] F. Nie, W. Zhu, and X. Li, "Unsupervised large graph embedding based on balanced and hierarchical k-means," *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 4, pp. 2008–2019, 2022.
[29] S. Cao, W. Lu, and Q. Xu, "Deep neural networks for learning graph representations." in *AAAI*, 2016, pp. 1145–1152.
[30] Q. Dai, Q. Li, J. Tang, and D. Wang, "Adversarial network embedding," in *AAAI*, 2018, pp. 2167–2174.
[31] A. Bojchevski and S. Günnemann, "Bayesian robust attributed graph clustering: Joint learning of partial anomalies and group structure," in *AAAI*, 2018, pp. 2738–2745.
[32] D. He, L. Zhai, Z. Li, D. Jin, L. Yang, Y. Huang, and P. S. Yu, "Adversarial mutual information learning for network embedding," in *IJCAI*, 2020, pp. 3321–3327.
[33] Z. Peng, W. Huang, M. Luo, Q. Zheng, Y. Rong, T. Xu, and J. Huang, "Graph representation learning via graphical mutual information maximization," in *WWW*, 2020, pp. 259–270.
[34] C. Wang, S. Pan, R. Hu, G. Long, J. Jiang, and C. Zhang, "Attributed graph clustering: A deep attentional embedding approach," in *IJCAI*, 2019, pp. 3670–3676.
[35] Z. Lin and Z. Kang, "Graph filter-based multi-view attributed graph clustering," in *IJCAI*, 2021, pp. 2723–2729.
[36] H. Li, Z. Wang, J. Cao, J. Pei, and Y. Shi, "Optimal estimation of low-rank factors via feature level data fusion of multiplex signal systems," *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 6, pp. 2860–2871, 2022.
[37] J. Ji, Y. Liang, and M. Lei, "Deep attributed graph clustering with self-separation regularization and parameter-free cluster estimation," *Neural Networks*, vol. 142, pp. 522–533, 2021.
[38] X. Zhang, H. Liu, X. Wu, X. Zhang, and X. Liu, "Spectral embedding network for attributed graph clustering," *Neural Networks*, vol. 142, pp. 388–396, 2021.
[39] H. Li, L. Wang, Z. Bu, J. Cao, and Y. Shi, "Measuring the network vulnerability based on markov criticality," *ACM Trans. Knowl. Discov. Data*, vol. 16, no. 2, pp. 28:1–28:24, 2022.
[40] N. Park, R. Rossi, E. Koh, I. A. Burhanuddin, S. Kim, F. Du, N. Ahmed, and C. Faloutsos, "Cgc: Contrastive graph clustering for community detection and tracking," in *WWW*, 2022, pp. 1115–1126.
[41] C. Fettal, L. Labiod, and M. Nadif, "Efficient graph convolution for joint node representation learning and clustering," in *WSDM*, K. S. Candan, H. Liu, L. Akoglu, X. L. Dong, and J. Tang, Eds., pp. 289–297.
[42] J. Cheng, Q. Wang, Z. Tao, D. Xie, and Q. Gao, "Multi-view attribute graph convolution networks for clustering," in *IJCAI*, 2020, pp. 2973–2979.
[43] W. Xia, Q. Wang, Q. Gao, X. Zhang, and X. Gao, "Self-supervised graph convolutional network for multi-view clustering," *IEEE Trans. Multim.*, vol. 24, pp. 3182–3192, 2022.
[44] F. M. Bianchi, D. Grattarola, and C. Alippi, "Spectral clustering with graph neural networks for graph pooling," in *ICML*, vol. 119, 2020, pp. 874–883.
[45] F. Wu, A. H. S. Jr., T. Zhang, C. Fifty, T. Yu, and K. Q. Weinberger, "Simplifying graph convolutional networks," in *ICML*, vol. 97, 2019, pp. 6861–6871.
[46] Z. Liu, C. Chen, L. Li, J. Zhou, X. Li, L. Song, and Y. Qi, "Geniepath: Graph neural networks with adaptive receptive paths," in *AAAI*, 2019, pp. 4424–4431.
[47] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains," *IEEE Signal Processing Magazine*, vol. 30, no. 3, pp. 83–98, 2013.
[48] F. R. Chung and F. C. Graham, *Spectral graph theory*. American Mathematical Society, 1997, no. 92.
[49] Q. Li, X. Wu, H. Liu, X. Zhang, and Z. Guan, "Label efficient semi-supervised learning via graph filtering," in *CVPR*, 2019, pp. 9582–9591.
[50] U. Von Luxburg, "A tutorial on spectral clustering," *Statistics and Computing*, vol. 17, no. 4, pp. 395–416, 2007.
[51] C. C. Aggarwal and C. K. Reddy, *Data Clustering: Algorithms and Applications*. Boca Raton: The CRC Press, 2014.
[52] V. Satopaa, J. R. Albrecht, D. E. Irwin, and B. Raghavan, "Finding a "kneedle" in a haystack: Detecting knee points in system behavior," in *ICDCS Workshops*, 2011, pp. 166–171.
[53] J. J. McAuley, C. Targett, Q. Shi, and A. van den Hengel, "Image-based recommendations on styles and substitutes," in *SIGIR*, 2015, pp. 43–52.
[54] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec, "Open graph benchmark: Datasets for machine learning on graphs," *CoRR*, vol. abs/2005.00687, 2020.

Xiaotong Zhang is currently an associate professor in the School of Software, Dalian University of Technology. She received the B.Eng. degree in software engineering and the Ph.D. degree in computer application technology from Dalian University of Technology in 2012 and 2018 respectively. Prior to that, she worked as a postdoctoral fellow in The Hong Kong Polytechnic University. Her research interests include artificial intelligence and data mining.

Han Liu is currently an associate professor in the School of Software, Dalian University of Technology. He received the B.Eng. degree in software engineering and the Ph.D. degree in computer application technology from Dalian University of Technology in 2012 and 2018 respectively. Prior to that, he worked as a postdoctoral fellow in The Hong Kong Polytechnic University. His research interests include artificial intelligence and data mining.

Qimai Li is currently a Ph.D. student at the Department of Computing, the Hong Kong Polytechnic University, supervised by Dr. Wu Xiao-Ming. He obtained B.Eng. degree from the College of Computer Science and Technology, Zhejiang University. His main area of interest includes machine learning on graph, semi-supervised Learning and graph signal processing.

Xiao-Ming Wu is currently an associate professor at the Department of Computing, The Hong Kong Polytechnic University. She obtained her PhD degree from the Department of Electrical Engineering, Columbia University in 2016. Prior to that, she obtained her MPhil degree from the Chinese University of Hong Kong and bachcelor and master degrees from Peking University. Her research interests include machine learning and applications of artificial intelligence in computer vision, natural language processing, and search and recommendation.

Xianchao Zhang is currently a professor in the School of Software, Dalian University of Technology. He received the scholar and master's degrees in mathematics from the National University of Defense Technology, in 1994 and 1998, respectively, and the PhD degree in computer science from the University of Science and Technology of China, in 2000. He is a full professor with the Dalian University of Technology. His research interests include design and analysis of algorithms and machine learning.