Scalable and Adaptive Spectral Embedding for Attributed Graph Clustering

Yunhui Liu
Tieke He*
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing, China
hetieke@gmail.com

Qing Wu*
School of Business
Nanjing University
Nanjing, China
wuqing@nju.edu.cn

Tao Zheng
Jianhua Zhao
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing, China

ABSTRACT

Attributed graph clustering, which aims to group the nodes of an attributed graph into disjoint clusters, has made promising advancements in recent years. However, most existing methods face challenges when applied to large graphs due to the expensive computational cost and high memory usage. In this paper, we introduce Scalable and Adaptive Spectral Embedding (SASE), a simple attributed graph clustering method devoid of parameter learning. SASE comprises three main components: node features smoothing via k-order simple graph convolution, scalable spectral clustering using random Fourier features, and adaptive order selection. With these designs, SASE not only effectively captures global cluster structures but also exhibits linear time and space complexity relative to the graph size. Empirical results demonstrate the superiority of SASE. For example, on the ArXiv dataset with 169K nodes and 1.17M edges, SASE achieves a 6.9% improvement in ACC and a 5.87× speedup compared to the runner-up, S3GC.

CCS CONCEPTS

• **Mathematics of computing** → **Graph algorithms**; • **Information systems** → **Clustering**.

KEYWORDS

Attributed Graph Clustering; Spectral Embedding; Graph Neural Networks

ACM Reference Format:
Yunhui Liu, Tieke He, Qing Wu, Tao Zheng, and Jianhua Zhao. 2024. Scalable and Adaptive Spectral Embedding for Attributed Graph Clustering. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management (CIKM '24), October 21–25, 2024, Boise, ID, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3627673.3679992

*Tieke He and Qing Wu are the corresponding authors.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
CIKM '24, October 21–25, 2024, Boise, ID, USA
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0436-9/24/10
https://doi.org/10.1145/3627673.3679992

1 INTRODUCTION

Attributed graphs, comprising nodes associated with feature attributes and edges characterizing pairwise relationships, are natural and efficient representations of non-Euclidean data. Attributed graph clustering, which aims to partition nodes in such graphs into disjoint groups, has shown significant advancements in recent years [8].

State-of-the-art attributed graph clustering methods focus on simultaneously modeling graph structure and node attributes to identify clusters within the data. In general, these methods can be classified into two categories: neural network (NN)-based and graph filtering (GF)-based. NN-based methods, such as graph auto-encoders [6, 9, 17] and contrastive graph clustering models [1, 20, 24], first embed nodes into a latent space using graph neural networks [3, 7, 15] and then perform clustering algorithms on learned embeddings. Although proven effective, they rely on a large number of trainable parameters, suffering from expensive computation overhead and high memory usage.

Instead, GF-based methods like AGC [21], FGC [5], and IAGC [22], discard parameter learning. They first use a k-order graph convolution, which serves as a low-pass graph filter, to aggregate low-frequency signals from neighbors, thus obtaining smoothed node features. Subsequently, spectral clustering (SC) [11] is applied to the similarity graph constructed by the inner product of these smoothed features. However, these methods are constrained by two significant limitations: 1) Smoothed node features may neglect distinguishable information in the original node features, leading to suboptimal performance on graphs with more informative original features. 2) They exhibit poor scalability when applied to large-scale graphs, due to the SC's quadratic space and time complexity in constructing similarity graphs and computing subsequent eigendecomposition.

To tackle these issues, we propose Scalable and Adaptive Spectral Embedding (SASE) for attributed graph clustering. Specifically, we first conduct k-order simple graph convolution [19] to smooth node features and fuse the original and smoothed node features linearly. Subsequently, we employ random Fourier features [13] to explicitly project fused node features into the kernel space, where SC can be implicitly and efficiently performed without explicitly constructing similarity graphs and computing eigendecomposition. Furthermore, we take the mean ratio of the intra-cluster distance with respect to the nearest-cluster distance for each node as the selection criterion to adaptively determine the order k of graph convolution. With the above designs, SASE not only effectively captures global cluster

structures but also exhibits linear time and space complexity relative to the graph size. Empirical results demonstrate the superiority of SASE. For example, SASE achieves a 6.9% ACC improvement and a 5.87% speedup over the runner-up S3GC [1] on the ArXiv dataset with 169K nodes and 1.17M edges.

2 PRELIMINARIES

2.1 Problem Definition

Consider an undirected attributed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \boldsymbol{X})$, where $\mathcal{V} = \{v_i\}_{i=1}^n$ is a set of n nodes, $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$ is a set of edges, and $\boldsymbol{X} \in \mathbb{R}^{n \times f}$ is the node attribute matrix. The adjacency matrix of the graph is denoted as $\boldsymbol{A} \in \{0,1\}^{n \times n}$, where $\boldsymbol{A}_{ij} = 1$ if an edge exists between node v_i and v_j, and $\boldsymbol{A}_{ij} = 0$ otherwise. We take $\boldsymbol{I}$ to be an identity matrix, and $\mathbf{1}$ to be a column vector of ones. Our goal is to partition the node set $\mathcal{V}$ into m disjoint groups $C = \{C_i\}_{i=1}^m = \{c_i\}_{i=1}^n$.

2.2 Spectral Clustering

Spectral clustering (SC) [11] is one of the most effective clustering approaches that capture hidden cluster structures in the data. It comprises two key stages: spectral embedding and subsequent k-means clustering, while spectral embedding involves the construction of a similarity graph and the eigendecomposition of the resulting graph Laplacian matrix. Given a dataset of n data points $\boldsymbol{X} \in \mathbb{R}^{n \times f}$, SC constructs a fully connected graph $\boldsymbol{W} \in \mathbb{R}^{n \times n}$ using a similarity (kernel) function, e.g., the Gaussian kernel:

$$\boldsymbol{W}_{ij} = \boldsymbol{\kappa}(\boldsymbol{x}_i, \boldsymbol{x}_j; \sigma) = \exp\left(-\frac{||\boldsymbol{x}_i - \boldsymbol{x}_j||^2}{2\sigma^2}\right), \tag{1}$$

where $\boldsymbol{\kappa}$ is the kernel function and σ is the bandwidth parameter. Let $\boldsymbol{D} = \mathrm{diag}(\boldsymbol{W}\mathbf{1})$ be the diagonal degree matrix, and $\boldsymbol{L} = \boldsymbol{I} - \boldsymbol{D}^{-1/2}\boldsymbol{W}\boldsymbol{D}^{-1/2}$ be the normalized graph Laplacian matrix. Let $\boldsymbol{U} \in \mathbb{R}^{n \times m}$ denote the bottom m eigenvectors of $\boldsymbol{L}$, or equivalently, the top m eigenvectors of the normalized adjacency matrix $\boldsymbol{D}^{-1/2}\boldsymbol{W}\boldsymbol{D}^{-1/2}$. SC groups the data points by performing k-means clustering on the spectral embedding matrix $\hat{\boldsymbol{U}}$, obtained from $\boldsymbol{U}$ through ℓ_2 normalization. However, the high computational costs of the similarity matrix $O(n^2 f)$ and the eigendecomposition $O(n^2 m)$ make SC hardly scalable to large-scale datasets.

2.3 Attributed Graph Clustering

Neural network-based methods first train graph neural networks [3, 7, 15] using a label-agnostic loss to obtain low-dimensional node embeddings, followed by traditional or neural clustering on the learned embeddings. For instance, graph auto-encoders [6, 9, 17] learn node embeddings by reconstructing the graph adjacency matrix. However, the quadratic space and time complexity of the adjacency reconstruction loss limits their scalability. Contrastive graph clustering [24] learns node embeddings by aligning augmented node pairs (positive pairs) from the same node while separating any two distinct nodes (negative pairs) from the graph. CCGC [20] leverages intrinsic supervision information from high-confidence clustering results to enhance the quality of positive and negative pairs. To improve scalability, BGRL [14] eliminates the need for negative pairs by minimizing an invariance-based loss for augmented graphs within a batch, while S3GC [1] carefully designs a negative sampler and loss function for contrastive graph clustering. Despite their proven effectiveness, these NN-based methods rely on a large number of trainable parameters, leading to long training time and high memory usage.

Graph filtering-based methods first use a k-order low-pass graph filter to smooth node features and then apply traditional clustering algorithms such as k-means and SC [11] on the smoothed node features. For example, SGC [19] utilizes the normalized adjacency matrix to aggregate long-range neighborhood information, while AGC [21] introduces an adaptive low-pass graph filter. FGC [21] further exploits high-order graph adjacency to learn a fine-grained self-expressive matrix. IAGC [22] boosts AGC by addressing the inconsistencies of filtered features with graph structure and raw features. However, these methods suffer from overlooking critical information in the original features and exhibit poor scalability on large graphs when employing SC. To address these issues, we propose a more effective and scalable GF-based method.

3 METHOD

3.1 Node Features Smoothing

Classical graph neural networks [3, 7, 15] assume that connected nodes tend to have similar features and belong to the same cluster, thus node features are supposed to be smooth on the graph manifold. To this end, we can obtain clustering-friendly node features by smoothing original node features. For simplicity, we adopt a k-order simple graph convolution (SGC) [19] as the low-pass graph filter to smooth node features:

$$\boldsymbol{X}^{(k)} = \left(\hat{\boldsymbol{D}}^{-1/2}\hat{\boldsymbol{A}}\hat{\boldsymbol{D}}^{-1/2}\right)^k \boldsymbol{X}, \tag{2}$$

where $\hat{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}$ is the adjacency matrix with inserted self-loops, $\hat{\boldsymbol{D}} = \mathrm{diag}(\hat{\boldsymbol{A}}\mathbf{1})$ is the diagonal degree matrix, and $\boldsymbol{X}^{(k)} \in \mathbb{R}^{n \times f}$ denotes the smoothed feature matrix. SGC just discards trainable parameters in each layer of GCN [7], so it is faster and more scalable. Additionally, SGC aggregates long-range neighborhood information by applying the k-th power of the normalized adjacency matrix, which is proven to capture global cluster structures better [21].

However, we argue that smoothed node features may neglect key information in the original node features as the order k increases, thus leading to suboptimal performance on graphs with more informative original features. To address this problem, we propose to combine the original and smoothed node features linearly:

$$\widetilde{\boldsymbol{X}} = \alpha \boldsymbol{X} + (1 - \alpha)\boldsymbol{X}^{(k)}, \tag{3}$$

where $\alpha \in [0, 1]$ is the linear coefficient. Note that the resultant feature matrix $\widetilde{\boldsymbol{X}} \in \mathbb{R}^{n \times f}$ usually has a high dimensionality f, which will lead to the "curse of dimensionality" problem and additional computational cost in the following spectral clustering step. Therefore, we generate the final node features $\boldsymbol{Z} \in \mathbb{R}^{n \times d}$ by performing randomized truncated singular value decomposition (T-SVD) [2] on $\widetilde{\boldsymbol{X}}$, where d is the reduced dimensionality.

3.2 Scalable Spectral Clustering

GF-based methods AGC [21], FGC [5], and IAGC [22] employ traditional SC directly on smoothed node features. However, SC encounters two significant computational bottlenecks: similarity graph construction and eigendecomposition, as discussed in Section 2.2. Here, we aim to make SC scalable without explicitly constructing similarity graphs and computing eigendecomposition.

Specifically, we first seek for a projection function $\boldsymbol{\phi}(\boldsymbol{z}) : \mathbb{R}^d \to \mathbb{R}^{2D}$ such that the non-negative Gaussian kernel $\boldsymbol{\kappa}(\boldsymbol{z}_i, \boldsymbol{z}_j; \sigma) \approx \boldsymbol{\phi}(\boldsymbol{z}_i)^\top \boldsymbol{\phi}(\boldsymbol{z}_j)$. In other words, we can approximate the similarity graph adjacency $\boldsymbol{W}$ with $\widetilde{\boldsymbol{Z}}\widetilde{\boldsymbol{Z}}^\top$, where $\widetilde{\boldsymbol{Z}} = \{\boldsymbol{\phi}(\boldsymbol{z}_i)\}_{i=1}^n$. Then we can efficiently perform SC using $\widetilde{\boldsymbol{Z}}$ by employing the following steps: 1) Compute the diagonal degree matrix: $\widetilde{\boldsymbol{D}} = \text{diag}\left(\widetilde{\boldsymbol{Z}}\widetilde{\boldsymbol{Z}}^\top \mathbf{1}\right) = \text{diag}\left(\widetilde{\boldsymbol{Z}}\left(\widetilde{\boldsymbol{Z}}^\top \mathbf{1}\right)\right)$; 2) Normalize $\widetilde{\boldsymbol{Z}}$ using the degree matrix $\widetilde{\boldsymbol{D}}$: $\hat{\boldsymbol{Z}} = \widetilde{\boldsymbol{D}}^{-1/2}\widetilde{\boldsymbol{Z}}$; 3) Compute the top d left singular vectors $\boldsymbol{U}$ of $\hat{\boldsymbol{Z}}$, which are equivalent to the bottom d eigenvectors of the normalized graph Laplacian matrix $\boldsymbol{I} - \widetilde{\boldsymbol{D}}^{-1/2}\widetilde{\boldsymbol{Z}}\widetilde{\boldsymbol{Z}}^\top\widetilde{\boldsymbol{D}}^{-1/2}$; 4) Perform k-means on the ℓ_2 normalized rows $\hat{\boldsymbol{U}}$ of $\boldsymbol{U}$. With these steps, SC can be performed with linear complexity w.r.t. the number of nodes, as detailed in Section 3.4.

Then we discuss how to approximate the projection function $\boldsymbol{\phi}(\boldsymbol{z}) : \mathbb{R}^d \to \mathbb{R}^{2D}$ through Random Fourier Features (RFF) [13]. For the non-negative Gaussian kernel $\boldsymbol{\kappa}(\boldsymbol{z}_i, \boldsymbol{z}_j; \sigma)$, we first randomly sample $\{\boldsymbol{\omega}_1, \boldsymbol{\omega}_2, \ldots, \boldsymbol{\omega}_D\}$ from a Gaussian distribution $\mathcal{N}\left(\mathbf{0}, \frac{1}{\sigma}\boldsymbol{I}\right)$. Subsequently, the projection function is defined as:

$$\boldsymbol{\phi}(\boldsymbol{z}) = \frac{\left[\cos(\boldsymbol{\omega}_1^\top \boldsymbol{z}), \ldots, \cos(\boldsymbol{\omega}_D^\top \boldsymbol{z}), \sin(\boldsymbol{\omega}_1^\top \boldsymbol{z}), \ldots, \sin(\boldsymbol{\omega}_D^\top \boldsymbol{z})\right]^\top}{\sqrt{D}}. \tag{4}$$

This approach explicitly projects node features into a kernel space, where SC can be efficiently performed using the resulting $\{\boldsymbol{\phi}(\boldsymbol{z}_i)\}_{i=1}^n$. Additionally, alternative non-negative kernels like the quadratic and Laplacian kernels can be employed, or other kernel approximation techniques such as Nystroem Approximation [18] and Tensor Sketch [12] can be considered.

3.3 Adaptive Order Selection

The selection of the convolution order k plays a pivotal role in our model. A small order may result in insufficient propagation of neighborhood information, while a large order can lead to oversmoothing, where the features of nodes from different clusters blend together and become indistinguishable [21]. To address this concern, AGC proposes using the intra-cluster distance as the selection criterion to adaptively determine the order. However, this criterion overlooks inter-cluster structures and involves quadratic time complexity in computation. Therefore, we introduce a more comprehensive and scalable criterion. Given spectral embeddings $\{\hat{\boldsymbol{u}}_i\}_{i=1}^n$, k-means clustering assignments $C = \{c_i\}_{i=1}^n$, and cluster centroids $\{\boldsymbol{p}_i\}_{i=1}^m$, our criterion score is defined as:

$$s = \frac{1}{n}\sum_{i=1}^{n} \frac{a(i)}{b(i)}, \tag{5}$$

where $a(i) = ||\hat{\boldsymbol{u}}_i - \boldsymbol{p}_{c_i}||$ is the distance of node v_i to its cluster centroid $\boldsymbol{p}_{c_i}$, and $b(i) = \min_{c_i \neq j} ||\hat{\boldsymbol{u}}_i - \boldsymbol{p}_j||$ is the distance of node v_i to the centroid of its nearest cluster. $s \in [0, 1]$ effectively captures global cluster structures since lower intra-cluster distance and higher inter-cluster distance will yield a better (smaller) score. Additionally, computing s entails a time complexity of $O(nmd)$, which scales linearly with the number of nodes.

Following AGC, we adaptively select the order k as follows. Starting from $k = 1$, we increment it by 1 iteratively. At each iteration t, we first obtain the clustering assignments $C^{(t)}$, as outlined in Sections 3.1 and 3.2. Subsequently, we compute the criterion score $s^{(t)}$. Once $\Delta = s^{(t-1)} - s^{(t)} < 0$, we stop the iteration and set the chosen $k = t - 1$. Consequently, the final clustering result is $C^{(t-1)}$.

3.4 Complexity Analysis

Space Complexity. The space complexity for SASE is $O(|\mathcal{E}| + n(f + d + D))$, where $|\mathcal{E}|$ is the total edge count, nf accounts for storing fused node features $\widetilde{\boldsymbol{X}}$, nd for node embeddings $\boldsymbol{Z}$ and spectral embeddings $\hat{\boldsymbol{U}}$, and nD for projected node embeddings $\widetilde{\boldsymbol{Z}}$.

Time Complexity. The time complexity is $O(k|\mathcal{E}|f + nkf + n(f + D)\log(d) + n(m + D)d)$. Specifically, computing mixed node features $\widetilde{\boldsymbol{X}} \in \mathbb{R}^{n\times f}$ takes $O(k|\mathcal{E}|f + knf)$ time; performing randomized T-SVD [2] on $\widetilde{\boldsymbol{X}}$ to obtain $\boldsymbol{Z} \in \mathbb{R}^{n\times d}$ takes $O(nf\log(d))$ time; projecting $\boldsymbol{Z}$ to $\widetilde{\boldsymbol{Z}} \in \mathbb{R}^{n\times 2D}$ using RFF takes $O(ndD)$ time; computing the diagonal degree matrix $\widetilde{\boldsymbol{D}}$ and $\hat{\boldsymbol{Z}}$ takes $O(nD)$ time; randomized T-SVD on $\hat{\boldsymbol{Z}}$ for $\boldsymbol{U} \in \mathbb{R}^{n\times d}$ takes $O(nD\log(d))$ time; applying k-means clustering to $\hat{\boldsymbol{U}}$ roughly takes $O(nmd)$ time; computing the selection criterion s takes $O(nmd)$ time.

In summary, SASE can be executed with linear space and time complexity w.r.t. the graph size. Additionally, SASE does not have any trainable parameters. Therefore, SASE enjoys very high efficiency and scalability.

4 EXPERIMENTS

4.1 Experimental Setup

4.1.1 Datasets. We conduct experiments on 3 small datasets: Cora, CiteSeer, PubMed [7], and 1 large dataset: ArXiv [4]. They are citation networks with nodes as papers and edges as citations. All nodes are labeled based on the respective subjects.

4.1.2 Baselines. We compare SASE with two traditional clustering methods k-means and SC [11]; ten NN-based methods DMoN [10], SENet [23], GAE [6], DAEGC [17], RARGA [9], GRACE [24], CCGC [20], BGRL [14], DGI [16], and S3GC [1]; four GF-based methods SGC [19], AGC [21], FGC [5], and IAGC [22]. Their results are either taken from the published paper or obtained using official codes when there are no corresponding results.

4.1.3 Implementation Details. We implement SASE using PyTorch on a 32GB NVIDIA Tesla V100 GPU, aligning cluster numbers with ground-truth classes and evaluating using Accuracy, NMI, and ARI. Hyper-parameters include setting $2D$ projection dimensionality to 100 and a maximum iteration number of 50, with others tuned via grid search.

4.2 Experimental Results

4.2.1 Performance. As shown in Table 1, SASE surpasses all baselines on three datasets, except Cora. We attribute this exception to

Table 1: Overall performance on node clustering (in percentage). "-" denotes that the method ran Out of Memory.

Dataset	Cora			CiteSeer			PubMed			ArXiv		
Metric	ACC	NMI	ARI	ACC	NMI	ARI	ACC	NMI	ARI	ACC	NMI	ARI
k-means	34.1±3.2	15.4±3.8	9.5±2.0	43.8±3.6	20.7±2.8	16.8±3.0	60.1±0.0	31.3±0.2	28.1±0.0	17.6±0.4	21.6±0.4	7.4±0.5
SC	59.1±1.2	46.1±1.0	34.4±1.0	46.0±0.7	23.9±0.5	18.5±0.4	66.6±0.0	28.8±0.0	30.3±0.0	-	-	-
DMoN	51.7±2.9	47.3±1.0	30.1±1.2	38.5±1.7	30.3±2.7	20.0±2.8	35.1±1.8	25.7±1.9	10.8±1.2	25.0±0.7	35.6±0.8	12.7±0.3
SENet	71.9±0.7	55.1±0.7	49.0±1.1	67.5±0.8	41.7±0.8	42.4±1.0	67.6±0.6	30.6±1.5	29.7±1.1	-	-	-
GAE	64.5±1.9	52.4±1.3	43.9±2.3	59.9±0.7	34.5±0.9	32.7±0.9	66.1±0.7	27.2±1.5	26.3±1.2	-	-	-
DAEGC	70.4±0.4	52.9±0.7	45.6±0.4	67.5±1.4	39.4±0.9	40.8±1.2	67.1±1.0	26.6±1.4	27.7±1.2	-	-	-
RARGA	71.2±0.7	50.7±0.8	47.1±2.3	48.6±0.7	28.5±0.3	18.9±1.3	69.2±0.9	30.0±1.2	30.9±1.4	-	-	-
GRACE	68.3±0.3	52.0±0.2	46.2±0.5	65.8±0.1	39.1±0.1	40.4±0.1	63.7±0.0	30.8±0.0	27.6±0.0	-	-	-
CCGC	73.9±1.2	56.5±1.0	52.5±1.9	69.8±0.9	44.3±0.8	45.7±1.8	68.8±0.6	32.1±0.2	30.8±0.4	-	-	-
BGRL	68.8±1.8	53.2±1.5	46.2±2.9	65.6±1.2	41.4±1.3	38.7±1.9	65.8±0.0	30.1±0.0	27.3±0.0	22.7±1.0	32.1±0.5	13.0±1.2
DGI	70.9±1.0	55.8±0.6	48.9±1.4	68.6±1.0	43.5±0.9	44.5±1.6	65.7±1.0	32.2±0.8	29.2±2.0	31.4±1.9	41.2±0.8	22.3±2.7
S3GC	**74.2±0.8**	**58.8±0.9**	**54.4±1.2**	68.8±1.0	44.1±1.1	44.8±0.9	**71.3±0.9**	33.3±1.0	34.5±0.9	35.0±1.2	46.3±0.3	27.0±1.6
SGC	64.9±2.0	51.8±1.3	43.7±2.0	68.3±0.2	43.1±0.2	43.7±0.2	68.5±0.0	31.9±0.0	31.0±0.0	38.2±0.5	45.2±0.2	33.3±0.5
AGC	68.9±0.2	53.7±0.4	44.8±0.0	67.0±0.2	41.1±0.4	41.6±0.0	69.8±0.0	31.6±0.0	31.0±0.0	-	-	-
FGC	69.3±0.0	54.2±0.1	47.0±0.1	68.2±0.0	43.2±0.1	43.9±0.1	69.2±0.0	32.1±0.0	31.5±0.0	-	-	-
IAGC	72.4±0.0	55.7±0.0	49.3±0.0	69.1±0.0	43.1±0.0	44.3±0.0	70.5±0.0	31.6±0.0	33.0±0.0	-	-	-
SASE	71.4±1.9	55.9±1.2	48.7±1.7	**70.2±0.1**	**44.9±0.1**	**46.8±0.1**	**71.3±0.0**	**37.0±0.0**	**35.3±0.0**	**41.9±0.6**	**46.8±0.1**	**37.6±0.4**

Table 2: Dataset statistics and hyper-parameters.

Dataset	#Nodes	#Edges	#Feat.	#Clus.	k	α	d
Cora	2,708	5,278	1,433	7	12	0.20	32
CiteSeer	3,327	4,614	3,703	6	11	0.30	12
PubMed	19,717	44,325	500	3	43	0.30	3
ArXiv	169,343	1,166,243	128	40	11	0.05	64

its requirement for powerful fitting abilities in non-linear neural networks. Compared to the runner-up S3GC, SASE improves NMI by 3.7% on PubMed and ACC by 6.9% on ArXiv, since SASE can aggregate long-range neighborhood information via k-order graph convolution while S3GC utilizes only a 1-layer GCN. Additionally, in contrast to GF-based methods, SASE achieves better performance on CiteSeer and PubMed. This superiority arises from SASE's capacity to retain crucial information in the original node features, as verified in Section 4.2.3.

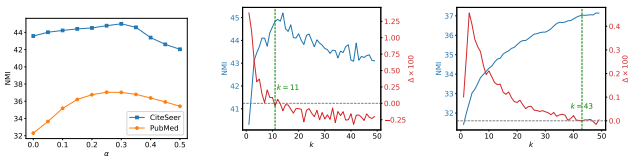
4.2.2 Scalability. In Table 3, we report the time costs of various methods on PubMed and ArXiv. It is observed that: 1) SASE greatly improves the scalability of GF-based methods AGC, FGC, and IAGC, which take quadratic space complexity during SC, e.g., storing the dense similarity graph for SC on ArXiv requires about 107 GiB of memory. 2) SASE demonstrates significant speed advantages over NN-based methods. For example, SASE achieves a 15.61× speedup over DGI on PubMed and a 5.87× speedup over S3GC on ArXiv.

Table 3: Execution time of various methods in seconds.

Dataset	DMoN	BGRL	DGI	S3GC	SGC	AGC	FGC	IAGC	SASE
PubMed	33.60	376.85	6.40	28.84	1.92	26.72	90.82	22.99	**0.41**
ArXiv	1850.49	-	196.19	193.05	40.46	-	-	-	**32.91**

4.2.3 Effectiveness of Fusing Original Features. Figure 1(a) shows the variation in NMI w.r.t. different weights α assigned to original node features. Initially, performance improves with increasing α but then declines. This observation highlights the efficacy of combining original and smoothed node features, particularly on datasets with highly informative node features. For example, SASE with $\alpha > 0$ consistently outperforms SASE with $\alpha = 0$ on PubMed.

4.2.4 Effectiveness of Adaptive Order Selection. Figures 1(b) and (c) depict the variations in NMI and Δ w.r.t. different orders k. We can see that when $\Delta < 0$, the corresponding NMI values closely approach the best performance. The selected k for Citeseer and PubMed, 11 and 43 respectively, approximate the optimal values of 13 and 48 on these datasets. These affirm the reliability of our proposed selection criterion in finding a favorable cluster partition.



(a) NMI w.r.t. α on CiteSeer and PubMed (b) NMI and Δ w.r.t. k on CiteSeer (c) NMI and Δ w.r.t. k on PubMed

Figure 1: Impact of α and k on CiteSeer and PubMed.

5 CONCLUSION

This paper presents SASE, which comprises node features smoothing via k-order simple graph convolution, scalable spectral clustering using random Fourier features, and adaptive order selection. SASE not only effectively captures global cluster structures but also exhibits linear time and space complexity relative to the graph size. Empirical results demonstrate SASE's effectiveness and efficiency.

ACKNOWLEDGMENTS

This work is partially supported by the National Key Research and Development Program of China (2021YFB1715600), and the National Natural Science Foundation of China (62306137).

REFERENCES

[1] Fnu Devvrit, Aditya Sinha, Inderjit Dhillon, and Prateek Jain. 2022. S3GC: scalable self-supervised graph clustering. *Advances in Neural Information Processing Systems* 35 (2022), 3248–3261.

[2] Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. 2011. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review* 53, 2 (2011), 217–288.

[3] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *Advances in neural information processing systems* 30 (2017).

[4] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems* 33 (2020), 22118–22133.

[5] Zhao Kang, Zhanyu Liu, Shirui Pan, and Ling Tian. 2022. Fine-grained attributed graph clustering. In *Proceedings of the 2022 SIAM International Conference on Data Mining (SDM)*. SIAM, 370–378.

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

[7] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations*.

[8] Yue Liu, Jun Xia, Sihang Zhou, Xihong Yang, Ke Liang, Chenchen Fan, Yan Zhuang, Stan Z Li, Xinwang Liu, and Kunlun He. 2022. A Survey of Deep Graph Clustering: Taxonomy, Challenge, Application, and Open Resource. *arXiv preprint arXiv:2211.12875* (2022).

[9] Nairouz Mrabah, Mohamed Bouguessa, Mohamed Fawzi Touati, and Riadh Ksantini. 2022. Rethinking graph auto-encoder models for attributed graph clustering. *IEEE Transactions on Knowledge and Data Engineering* (2022).

[10] Emmanuel Müller. 2023. Graph clustering with graph neural networks. *Journal of Machine Learning Research* 24 (2023), 1–21.

[11] Andrew Ng, Michael Jordan, and Yair Weiss. 2001. On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems* 14 (2001).

[12] Ninh Pham and Rasmus Pagh. 2013. Fast and scalable polynomial kernels via explicit feature maps. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. 239–247.

[13] Ali Rahimi and Benjamin Recht. 2007. Random features for large-scale kernel machines. *Advances in neural information processing systems* 20 (2007).

[14] Shantanu Thakoor, Corentin Tallec, Mohammad Gheshlaghi Azar, Mehdi Azabou, Eva L Dyer, Remi Munos, Petar Veličković, and Michal Valko. 2022. Large-Scale Representation Learning on Graphs via Bootstrapping. In *International Conference on Learning Representations*.

[15] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *International Conference on Learning Representations*.

[16] Petar Veličković, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. 2019. Deep Graph Infomax. In *International Conference on Learning Representations*. https://openreview.net/forum?id=rklz9iAcKQ

[17] Chun Wang, Shirui Pan, Ruiqi Hu, Guodong Long, Jing Jiang, and Chengqi Zhang. 2019. Attributed Graph Clustering: A Deep Attentional Embedding Approach. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI'19)*. 3670–3676.

[18] Christopher Williams and Matthias Seeger. 2000. Using the Nyström method to speed up kernel machines. *Advances in neural information processing systems* 13 (2000).

[19] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. 2019. Simplifying graph convolutional networks. In *International conference on machine learning*. PMLR, 6861–6871.

[20] Xihong Yang, Yue Liu, Sihang Zhou, Siwei Wang, Wenxuan Tu, Qun Zheng, Xinwang Liu, Liming Fang, and En Zhu. 2023. Cluster-guided Contrastive Graph Clustering Network. In *AAAI Conference on Artificial Intelligence*.

[21] Xiaotong Zhang, Han Liu, Qimai Li, and Xiao-Ming Wu. 2019. Attributed graph clustering via adaptive graph convolution. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI'19)*. 4327–4333.

[22] Xiaotong Zhang, Han Liu, Qimai Li, Xiao-Ming Wu, and Xianchao Zhang. 2023. Adaptive Graph Convolution Methods for Attributed Graph Clustering. *IEEE Transactions on Knowledge and Data Engineering* (2023).

[23] Xiaotong Zhang, Han Liu, Xiao-Ming Wu, Xianchao Zhang, and Xinyue Liu. 2021. Spectral embedding network for attributed graph clustering. *Neural Networks* 142 (2021), 388–396.

[24] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. 2020. Deep graph contrastive representation learning. *arXiv preprint arXiv:2006.04131* (2020).