Efficient Graph Convolution for Joint Node Representation Learning and Clustering

Chakib Fettal
Université de Paris
Centre Borelli UMR9010
and Informatique CDC
chakib.fettal@etu.u-paris.fr

Lazhar Labiod
Université de Paris
Centre Borelli UMR9010
lazhar.labiod@u-paris.fr

Mohamed Nadif
Université de Paris
Centre Borelli UMR9010
mohamed.nadif@u-paris.fr

ABSTRACT

Attributed graphs are used to model a wide variety of real-world networks. Recent graph convolutional network-based representation learning methods have set state-of-the-art results on the clustering of attributed graphs. However, these approaches deal with clustering as a downstream task while better performances can be attained by incorporating the clustering objective into the representation learning process. In this paper, we propose, in a unified framework, an objective function taking into account both tasks simultaneously. Based on a variant of the simple graph convolutional network, our model does clustering by minimizing the difference between the convolved node representations and their reconstructed cluster representatives. We showcase the efficiency of the derived algorithm against state-of-the-art methods both in terms of clustering performance and computational cost on the *de facto* benchmark graph clustering datasets. We further demonstrate the usefulness of the proposed approach for graph visualization through generating embeddings that exhibit a clustering structure.

CCS CONCEPTS

• **Computing methodologies** → **Cluster analysis**; **Dimensionality reduction and manifold learning**.

KEYWORDS

Graph Convolutional Networks; Node Embedding; Node Clustering

ACM Reference Format:
Chakib Fettal, Lazhar Labiod, and Mohamed Nadif. 2022. Efficient Graph Convolution for Joint Node Representation Learning and Clustering. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining (WSDM '22), February 21–25, 2022, Tempe, AZ, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3488560.3498533

1 INTRODUCTION

In data science, low-dimensional representation learning is commonly used for visualization purposes, but it can also play a significant role in the clustering task, where the aim is to divide a dataset into homogeneous clusters. Indeed, working with a low-dimensional space can be useful when partitioning data, and a number of approaches are reported in the literature. Recently, the authors in [20] have performed experiments on the sequential combination of deep representation learning techniques such as the autoencoder (AE), variational AE [4] and convolutional AE [14, 21], and some popular clustering methods such as k-means; interactive Python Notebooks and further technical details can be found at [1]. They observed that this improves clustering results but that there is no 'one size fits all'. Thus, low-dimensional representation learning followed by cluster analysis can be helpful in data science. The `k-means` applied on data embeddings, derived from classical embedding methods such as the AE for instance, is a popular approach. This procedure is carried out sequentially and is referred to as *the tandem approach* [43]. However, AE may sometimes be unsuitable for reducing dimension before clustering; it can fail to retain information which could be valuable for the clustering task. Hence, jointly optimizing for both tasks –representation learning and clustering– is a good alternative [2, 3, 13, 24]. Learning representations that are both faithful to the data while simultaneously adjusting them to a have a clustering-friendly structure can lead to a better clustering performance [6, 9, 13].

Clustering in the context of attributed graphs, which are graphs whose nodes and/or edges have attributes or features, despite being an important unsupervised task, has proved more impervious to such advances. Furthermore, some of these attributed graph clustering methods suffer from high spatial and/or computational complexity. Unlike most existing approaches, this paper aims to overcome this weakness by considering a joint graph embedding and clustering, which alternates iteratively between both tasks, that is to say between embedding and clustering. Attributed graphs are used to model a wide variety of real-world networks such as recommender systems [12, 32, 47], computer vision [30, 35, 46], Natural language processing [26, 37] and physical systems [19, 34]. Due to the irregular high-dimensional non-euclidean structure of graphs as well as the various node-level features it may contain, looking for suitable euclidean-representations that incorporate the structural and features' information of these graphs is an interesting challenge in machine learning [17]. Recent literature proposes to learn these representations automatically. Loosely speaking, these representation learning methods aim at embedding the nodes into a low-dimensional space where in the embedded nodes' proximity should be similar enough to that of those in the original graph representation. These methods can be based on approaches such as

WSDM '22, February 21–25, 2022, Tempe, AZ, USA

ACM ISBN 978-1-4503-9132-0/22/02…$15.00
https://doi.org/10.1145/3488560.3498533

[1] https://github.com/rezacsedu/Deep-learning-for-clustering-in-bioinformatics

factorization [1, 7], random walks [15, 29], or neighborhood autoencoders [41, 50]. Recently, the *Graph Convolutional Network* (GCN) [10, 23] has garnered a lot of attention due to its ability to learn high-quality graph representations, and by extension, its effectiveness for different graph-related tasks such as node classification, link prediction, and node clustering which is the task our paper focuses on. This node clustering, however, is generally performed as a downstream task but some efficient GCN-based approaches for the simultaneous embedding and clustering have recently emerged [5, 48]. In this paper, we propose to rely on the GCN to develop a novel *Graph Convolutional Clustering* model referred to as GCC that is capable of taking into account both tasks simultaneously. Our contributions in this paper can be summarized as follows:

- We provide a variant of the GCN propagation matrix and demonstrate how it makes the GCN truly act as a low-pass filter.
- We propose a new formulation combining the graph convolutional representation learning and the clustering processes and show how our proposed GCC approach is related to some other methods.
- We derive an efficient algorithm referred to as GCC[2] and study its computational complexity in detail. We release the code[3] for easy reproducibility.
- We perform experimentations to showcase the worth of our proposal both in terms of clustering and quality of embedding.

This paper is organized as follows. Section 2 presents related works. Section 3 presents the proposed approach and its derivation. Section 4 is devoted to the proposed algorithm and its computational complexity study. In section 5, we compare GCC with the state-of-the-art in terms of clustering and evaluate its performance in terms of embedding. Finally, Section 6 presents our conclusions.

2 RELATED WORK

Our contributions lie in the intersection of several research topics, graph representation learning, graph clustering and graph convolutional neural networks.

Unsupervised Graph Representation learning. Unsupervised Graph Representation learning is generally done either through contrastive learning or via autoencoders.

The contrastive methods learn representations in a self-supervised way. They commonly rely on maximizing mutual information. DGI [40], for example, maximizes mutual information between node and graph representations. InfoGraph [38] expands the previous concept to graph substructures of different scales rather than just the node-level e.g. edges, triangles.

Autoencoder-based models learn embeddings by trying to reconstruct some property of the graph, generally the adjacency matrix. Variational Graph Autoencoders (VGAE) [22] extend the concept of variational autoencoders to the graph context, it uses a GCN based encoder and a dot product decoder. Linear variational Graph autoencoders [33] simplify VGAE by defining the encoder to be a linear transformation with a one-hop propagation matrix.

Graph Clustering. Graph clustering is the process of grouping nodes into clusters depending on the structure of the graph and/or node-level features. By only considering node attributes, classical clustering algorithms can be used to cluster the graph. Algorithms that rely on graph structure exclusively include the spectral clustering algorithm [28] that optimizes the ratio and normalized-cut criteria. Graclus [11] is mathematically equivalent to the spectral clustering algorithm but is faster due to not having to compute the eigenvalues of the graph Laplacian.

Finally, approaches that leverage both graph structure and node attributes commonly learn representations before applying classical clustering algorithms on them [42, 49]. However, some recent works explored integrating the clustering loss directly into the objective.

Clustering-friendly Graph Representation Learning. Literature on joint representation learning and clustering claim that doing the two tasks simultaneously can improve clustering quality. DCN [44] proposed to include the k-means clustering loss to the autoencoder loss as a regularization. Deep k-means [13] followed on this concept by proposing a fully differentiable formulation of this problem. In the context of attributed graph clustering, joint clustering and embedding is starting to receive some attention. GEMSEC [31] maximizes the information between labels and visual input data indices in order to self-label the data. AGC [48] proposes to exploit high-order neighborhoods in the clustering process through an adaptive rule for neighborhood order selection. Deep Modularity Network [5] clusters the graph by maximizing spectral modularity. Graph InfoClust (GIC) [27] computes clusters by maximizing the mutual information between nodes contained in the same cluster.

3 PROPOSED METHOD

In this section we describe how we formulate the simultaneous node embedding and clustering problem. Then we propose a new model for solving it (as depicted in figure 1).

3.1 Preliminaries and Notations

Let $\mathcal{G} = (V, \mathbf{A}, \mathbf{X})$ be an attributed undirected graph where V represents the vertex set consisting of nodes $\{v_1, ..., v_n\}$, $\mathbf{A} \in \mathbb{R}^{n \times n}$ is a symmetric adjacency matrix where a_{ij} denotes the edge weight between nodes v_i and v_j, and $\mathbf{X} \in \mathbb{R}^{n \times d}$ is a node-level feature matrix. Tr denotes the trace of a matrix. In what follows k represents the number of clusters. f is the embedding dimension. $\mathbf{1}_m$ represents a column vector of m ones. $\mathbf{I}_m$ represents an identity matrix of dimension m. If $\mathbf{G}$ is a matrix then $\mathbf{m}_i$ is its i-th row vector, $\mathbf{m}'_j$ is its j-th column vector and m_{ij} is the j-th element of the i-th row.

3.2 Joint Graph Representation Learning and Clustering

We formulate the simultaneous node embedding and clustering problem as follows

$$\min_{\theta_1,\theta_2,\mathbf{G},\mathbf{F}} \underbrace{\left\| \mathsf{dec}_{\theta_2}\Big(\mathsf{enc}_{\theta_1}\big(\mathsf{agg}(\mathbf{A},\mathbf{X})\big)\Big) - \mathsf{agg}(\mathbf{A},\mathbf{X}) \right\|^2}_{\text{reconstruction term}} + \alpha \underbrace{\left\| \mathsf{enc}_{\theta_1}\big(\mathsf{agg}(\mathbf{A},\mathbf{X})\big) - \mathbf{GF} \right\|^2}_{\text{clustering regularization term}} \quad (1)$$

$$\text{s.t.} \quad \mathbf{G} \in \{0,1\}^{n \times k},\ \mathbf{G}\mathbf{1}_k = \mathbf{1}_n$$

[2] From now on, in order to distinguish between a model and its derived algorithm, we will use *typewriter font* for an algorithm. Consequently, GCC is the model and GCC its derived algorithm.

[3] https://github.com/chakib401/graph_convolutional_clustering

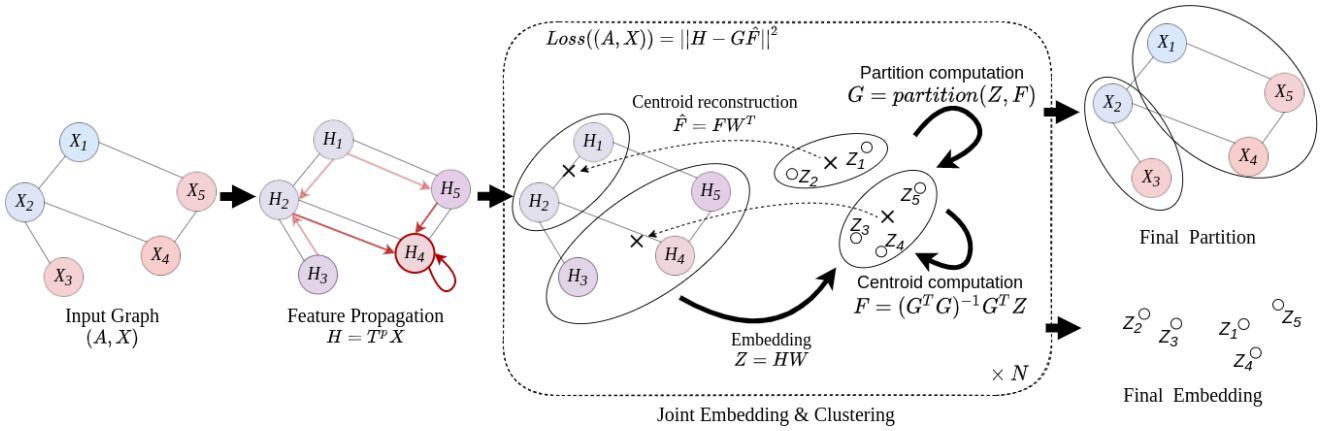


Figure 1: Schema of the GCC model: GCC creates an initial representation of the graph before iteratively learning to embed and cluster the data. The graph signal is represented by the colors of the node. Feature propagation results in a smoother signal.

where enc_{θ_1} is the encoding function, dec_{θ_2} is the decoding function, $\mathsf{agg}(\mathbf{A},\mathbf{X})$ is some aggregate of $\mathbf{A}$ and $\mathbf{X}$ which represents the information contained in the graph (structure and node-features), $\mathbf{G} \in \{0,1\}^{n\times k}$ the binary classification matrix, $\mathbf{F} \in \mathbb{R}^{k\times d}$ play the role of centroids in the embedding space and α is a coefficient that regulates the trade-off between seeking reconstruction and clustering.

The clustering regularizer is the k-means clustering loss [25] on the encoded observations. It penalizes transformations that do not result in a clustering-friendly representations.

3.3 Linear Graph Embedding

Linear graph autoencoders (LGAE) [33] have shown that a linear encoder with an inner product decoder can be powerful enough to reach competitive results w.r.t more complex GCN-based models on the link prediction and node clustering tasks. Consequently, we also define our encoder to be a simple linear transformations i.e.

$$\mathsf{enc}(\mathsf{agg}(\mathbf{A},\mathbf{X});\mathbf{W}_1) = \mathsf{agg}(\mathbf{A},\mathbf{X})\mathbf{W}_1$$

In LGAE, the decoder attempts to reconstruct the adjacency matrix $\mathbf{A}$ rather than an aggregation of $\mathbf{A}$ and $\mathbf{X}$. This means that this type of decoder is not suitable for our problem. Therefore, we also define the decoder as a simple linear transformation

$$\mathsf{dec}(\mathbf{Z};\mathbf{W}_2) = \mathbf{Z}\mathbf{W}_2$$

where $Z = \mathsf{agg}(\mathbf{A},\mathbf{X})\mathbf{W}_1$.

3.4 Normalized Simple Graph Convolution

Our choice for the aggregate function is inspired by the simple graph convolution proposed in SGC [42]. We set

$$\mathsf{agg}(\mathbf{A},\mathbf{X}) = \mathbf{T}^p\mathbf{X} \tag{2}$$

but rather than have $\mathbf{T}$ be the symmetric normalized adjacency matrix with added self-loops, we define it to be

$$\mathbf{T} = {\mathbf{D}_\mathbf{T}}^{-1}(\mathbf{I} + \tilde{\mathbf{S}}) \tag{3}$$

where $\tilde{\mathbf{S}} = \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}$ with $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ and $\tilde{\mathbf{D}}$ (resp. $\mathbf{D}_\mathbf{T}$) being the diagonal matrix of degrees of $\tilde{\mathbf{A}}$ (resp. $\mathbf{I} + \tilde{\mathbf{S}}$).

The GCN, and by extension the SGC, do graph signal filtering with matrix $\mathbf{I} - \tilde{\mathbf{S}} = \mathbf{I} - \tilde{\mathbf{D}}^{-1/2}(\mathbf{I} - \tilde{\mathbf{L}})\tilde{\mathbf{D}}^{-1/2}$ where $\tilde{\mathbf{L}}$ is the Laplacian of $\tilde{\mathbf{A}}$. The frequency response function of this filter is $h(\tilde{\lambda_l}) = 1 - \tilde{\lambda_l}$ where λ_l is a frequency of the graph. In the GCN stacking K-layers, or equivalently raising $\tilde{\mathbf{S}}$ to power K in SGC, implies doing the filtering with frequency response function $h_K(\tilde{\lambda_l}) = (1 - \tilde{\lambda_l})^K$. This filter is low-pass on $[0,1]$ but not $[0,1.5]$. We then propose to further add self-loops and row normalize matrix $\tilde{\mathbf{S}}$. This has the following effects

- From the spectral perspective: The proposed normalization further shrinks the spectrum of the matrix to lie in $[0,1]$, as can be seen in figure 2, which makes the filter truly low-pass.
- From the spatial perspective: Each transformed vertex becomes a weighted-average of the neighbors which is more intuitive but it also takes into account column degree information unlike direct random walk adjacency normalization.

We further motivate this choice in the experiments section. Thereby, with this aggregation function, our problem turns into

$$\begin{aligned} \min_{\mathbf{G},\mathbf{F},\mathbf{W}_1,\mathbf{W}_2} \quad & \left\| \mathbf{T}^p\mathbf{X} - \mathbf{T}^p\mathbf{X}\mathbf{W}_1\mathbf{W}_2 \right\|^2 + \alpha \left\| \mathbf{T}^p\mathbf{X}\mathbf{W}_1 - \mathbf{G}\mathbf{F} \right\|^2 \\ \text{s.t.} \quad & \mathbf{G} \in \{0,1\}^{n\times k},\ \mathbf{G}\mathbf{1}_k = \mathbf{1}_n \end{aligned} \tag{4}$$

Both terms of (4) make it possible to express a connection between the two tasks, the first term plays the role of linear autoencoder and the second the role of clustering in the embedding space. We decide in the following to give the same weight for the two terms ($\alpha = 1$).

3.5 Graph Convolutional Clustering

To obtain a mutual supplementation between embedding and clustering, we assume $\mathbf{W} = \mathbf{W}_1 = {\mathbf{W}_2}^\top$ and add an orthogonality

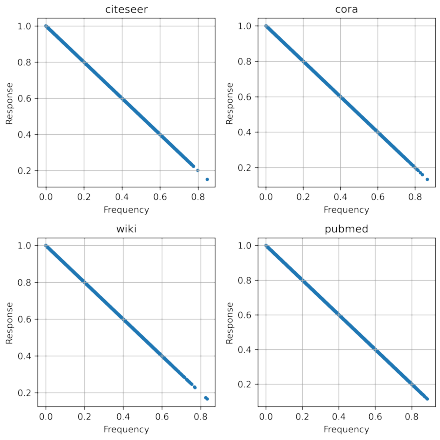


Figure 2: frequency response of the proposed GCN filter plotted against the frequency on four real-world datasets

constraint on $\mathbf{W}$ in (4). It gives rise to the following problem

$$\begin{aligned} \min_{\mathbf{G},\mathbf{F},\mathbf{W}} \quad & \left\| \mathbf{T}^p\mathbf{X} - \mathbf{T}^p\mathbf{X}\mathbf{W}\mathbf{W}^\top \right\|^2 + \left\| \mathbf{T}^p\mathbf{X}\mathbf{W} - \mathbf{G}\mathbf{F} \right\|^2 \\ \text{s.t.} \quad & \mathbf{G} \in \{0,1\}^{n\times k},\ \mathbf{G}\mathbf{1}_k = \mathbf{1}_n,\ \mathbf{W}^\top\mathbf{W} = \mathbf{I}_k \end{aligned} \tag{5}$$

Similar to [43], solving this problem can be proven to be equivalent to

$$\begin{aligned} \min_{\mathbf{G},\mathbf{F},\mathbf{W}} \quad & \left\| \mathbf{T}^p\mathbf{X} - \mathbf{G}\mathbf{F}\mathbf{W}^\top \right\|^2 \\ \text{s.t.} \quad & \mathbf{G} \in \{0,1\}^{n\times k},\ \mathbf{G}\mathbf{1}_k = \mathbf{1}_n,\ \mathbf{W}^\top\mathbf{W} = \mathbf{I}_k \end{aligned} \tag{6}$$

To prove this, we first decompose the reconstruction term

$$\begin{aligned} ||\mathbf{T}^p\mathbf{X} - \mathbf{T}^p\mathbf{X}\mathbf{W}\mathbf{W}^\top||^2 &= ||\mathbf{T}^p\mathbf{X}||^2 + ||\mathbf{T}^p\mathbf{X}\mathbf{W}\mathbf{W}^\top||^2 - 2||\mathbf{T}^p\mathbf{X}\mathbf{W}||^2 \\ &= ||\mathbf{T}^p\mathbf{X}||^2 - ||\mathbf{T}^p\mathbf{X}\mathbf{W}||^2 \quad \text{due to } \mathbf{W}^\top\mathbf{W} = \mathbf{I}_k. \end{aligned}$$

Similarly, the clustering regularization term can be decomposed as follows

$$||\mathbf{T}^p\mathbf{X}\mathbf{W} - \mathbf{G}\mathbf{F}||^2 = ||\mathbf{T}^p\mathbf{X}\mathbf{W}||^2 + ||\mathbf{G}\mathbf{F}||^2 - 2\mathrm{Tr}((\mathbf{T}^p\mathbf{X}\mathbf{W})^\top\mathbf{G}\mathbf{F})$$

Summing the two resulting expressions we get

$$\begin{aligned} ||\mathbf{T}^p\mathbf{X}||^2 + ||\mathbf{G}\mathbf{F}||^2 - 2\mathrm{Tr}((\mathbf{T}^p\mathbf{X}\mathbf{W})^\top\mathbf{G}\mathbf{F}) &= ||\mathbf{T}^p\mathbf{X} - \mathbf{G}\mathbf{F}\mathbf{W}^\top||^2 \\ &\text{due to } ||\mathbf{G}\mathbf{F}\mathbf{W}^\top|| = ||\mathbf{G}\mathbf{F}|| \end{aligned}$$

Thus, optimizing (5) is equivalent to optimizing (6) □.

Before tackling the resolution of this problem in section 4, we will first look at how our proposed GCC approach is related to some other methods.

3.6 Connections to Existing Work

Simple Graph Convolution Variants. Similarly to SGC, the computation of $\mathbf{T}^p\mathbf{X}$ can be considered to be a pre-processing step with a different propagation matrix $\mathbf{T}$. This representation is then used for a downstream task. In the original paper that task was classification where the representation is fed to a linear regression model corresponding to a fully connected neural network layer with sigmoid activations. Other variants of the simple graph convolution can also be used as the aggregation function e.g. for the simple spectral graph convolution (S²GC) [49] we have $\mathtt{agg}(\mathbf{A},\mathbf{X}) = \frac{1}{p}\sum_{i=1}^{p}\mathbf{T}^i\mathbf{X}$.

Graph Autoencoder and Linear Graph Autoencoder. Our model can be seen as a case of the non-probabilistic variant of the VGAE model adapted to graph clustering. Like VGAE, the encoder we use is a form of GCN but rather than an inner-product decoder, we use a linear decoder. The original graph autoencoder was used for link-prediction, i.e., it tried to reconstruct a completed version of the adjacency matrix $\mathbf{A}$. In our case we reconstruct the convolved matrix $\mathbf{T}^p\mathbf{X}$

Deep Clustering Network. From $\mathbf{X}$, the DCN algorithm [44] also performs unsupervised clustering using a deep autoencoder; it uses an optimization objective that is a weighted combination of a reconstruction error and a clustering error. The DCN cost function is given by

$$\begin{aligned} \min_{\theta_1,\theta_2,\mathbf{G},\{\mathbf{f}_i\}} \ & \ell\big(g_{\theta_2}(f_{\theta_1}(\mathbf{x}_i)),\mathbf{x}_i\big) + \frac{\lambda}{2}\sum_i \left\| f_{\theta_1}(\mathbf{x}_i) - \mathbf{G}\mathbf{f}_i \right\|_2^2 \\ \text{s.t.} \quad & s_{ij} \in \{0,1\},\ \mathbf{1}^\top\mathbf{f}_i = 1 \end{aligned}$$

where f is the encoder and g is the decoder, $\{\mathbf{f}_i\}$ are the centroids, $\mathbf{G}$ is a membership matrix and ℓ is a loss function. If we take $\lambda = 2$, the encoder and decoder to be linear functions $f(\mathbf{x};\mathbf{W}) = \mathbf{x}\mathbf{W}$ and $g(\mathbf{x};\mathbf{W}^\top) = \mathbf{x}\mathbf{W}^\top$ with a semi-orthogonality constraint on $\mathbf{W}$, the loss function to be the mean squared error and by considering the observations to be the rows of $\mathbf{T}^p\mathbf{X}$ we get the problem as formulated in (5). As for the optimization process, the update rule is the same for the cluster assignment while it differs in the centroids, encoder and decoder updates.

4 OPTIMIZATION AND ALGORITHM

Directly optimization problem in (6) is tricky so we use the following alternating iterative approach. The algorithm alternately fixes two of the matrices $\mathbf{F}$, $\mathbf{G}$ and $\mathbf{W}$ and solves for the third one.

4.1 Optimization Procedure

For each matrix, through fixing the two other matrices we obtain a formula which can be solved directly. The solutions to these modified problems are guaranteed to decrease the overall cost function monotonically. The initialization and update rules are described in what follows.

Initialization. We initialize $\mathbf{W}$ with the first f components obtained from applying a randomized *Principal Component Analysis* (PCA) on $\mathbf{T}^p\mathbf{X}$. Matrices $\mathbf{F}$ and $\mathbf{G}$ are then obtained via a k-means on $\mathbf{T}^p\mathbf{X}\mathbf{W}$.

Update Rule for F. By fixing $\mathbf{G}$ and $\mathbf{W}$ and solving for $\mathbf{F}$ we obtain a linear least squares problem. By setting the derivative to zero, we obtain the normal equation which is the optimal solution to the given problem. The update rule is then

$$\mathbf{F} = (\mathbf{G}^\top\mathbf{G})^{-1}\mathbf{G}^\top\mathbf{T}^p\mathbf{X}\mathbf{W}. \tag{7}$$

Intuitively, each row vector $\mathbf{f}_i$ is set to the average of the embeddings $\mathbf{X}\mathbf{W}$ that are assigned to cluster i. In the k-means algorithm

this corresponds to the centroid update step.

Update Rule for W. By fixing **G** and **F** and solving for **W**, the update rule is as follows

$$\mathbf{W} = \mathbf{U}\mathbf{V}^\top \quad \text{s.t.} \quad [\mathbf{U}, \Sigma, \mathbf{V}] \quad = \mathtt{SVD}\big((\mathbf{T}^p\mathbf{X})^\top \mathbf{G}\mathbf{F}\big) \tag{8}$$

where $\Sigma = (\sigma_{ii})$, **U**, and **V** are respectively the singular values, the left and right singular vectors of the matrix $(\mathbf{T}^p\mathbf{X})^\top\mathbf{G}\mathbf{F}$.

To prove this, fixing **F** and **G** in (6) leads to the following generalized Procrustes problem

$$\min_{\mathbf{W}} \quad \left\| \mathbf{T}^p\mathbf{X} - \mathbf{G}\mathbf{F}\mathbf{W}^\top \right\|^2 \quad \text{s.t.} \quad \mathbf{W}^\top\mathbf{W} = \mathbf{I}_k. \tag{9}$$

As $||\mathbf{T}^p\mathbf{X} - \mathbf{G}\mathbf{F}\mathbf{W}^\top||^2 = ||\mathbf{T}^p\mathbf{X}||^2 + ||\mathbf{G}\mathbf{F}\mathbf{W}^\top||^2 - 2\mathrm{Tr}(\mathbf{W}\mathbf{F}^\top\mathbf{G}^\top\mathbf{T}^p\mathbf{X})$. and since $||\mathbf{G}\mathbf{F}\mathbf{W}^\top||^2 = ||\mathbf{G}\mathbf{F}||^2$ (9) is equivalent to

$$\max_{\mathbf{W}} \quad \mathrm{Tr}\big(\mathbf{W}\mathbf{F}^\top\mathbf{G}^\top\mathbf{T}^p\mathbf{X}\big) \quad \text{s.t.} \quad \mathbf{W}^\top\mathbf{W} = \mathbf{I}_k.$$

By taking $[\mathbf{U}, \Sigma, \mathbf{V}] = \mathtt{SVD}\big(\mathbf{F}^\top\mathbf{G}^\top\mathbf{T}^p\mathbf{X}\big)$, we have

$$\begin{aligned}\mathrm{Tr}\big(\mathbf{W}\mathbf{F}^\top\mathbf{G}^\top\mathbf{T}^p\mathbf{X}\big) &= \mathrm{Tr}\big(\mathbf{W}\mathbf{U}\Sigma\mathbf{V}^\top\big) \\ &= \sum_{i=1}^{f} \sigma_{ii} < \mathbf{w}_i'\mathbf{U}, \mathbf{v}_i' > \\ &\leq \sum_{i=1}^{f} \sigma_{ii}||\mathbf{w}_i'\mathbf{U}|| \times ||\mathbf{v}_i'|| = \sum_{i=1}^{f} \sigma_{ii} = \mathrm{Tr}(\Sigma).\end{aligned}$$

This implies that an upper bound for (9) is attained when $\mathrm{Tr}(\mathbf{W}\mathbf{U}\Sigma\mathbf{V}^\top) = \mathrm{Tr}(\Sigma)$ or equivalently when $\mathbf{V}^\top\mathbf{W}\mathbf{U} = \mathbf{I}$ meaning that the maximum is attained at $\mathbf{W} = \mathbf{V}\mathbf{U}^\top$ □.

Update Rule for G. By fixing **F** and **W** and solving for **F**, we get a problem that can be optimized with the assignment step of the k-means algorithm. The update rule is, then, given as

$$g_{ij^*} \leftarrow \begin{cases} 1 & \text{if } j^* = \arg\min_j \left\| (\mathbf{T}^p\mathbf{X}\mathbf{W})_i - \mathbf{f}_j \right\|^2 \\ 0 & \text{otherwise.} \end{cases} \tag{10}$$

4.2 The GCC Algorithm

The steps in the GCC algorithm are outlined in Algorithm 1. The convergence of GCC is guaranteed, but it will only reach a local optimum according to the initial conditions. A possible strategy to overcome this is to run GCC several times and to select the best result relative to the objective function. The selection of the propagation order p is integral to the overall performance of the algorithm. A smaller p can mean insufficient neighborhood information is being propagated while a larger p can cause over-smoothing of the graph signal. Figure 3 shows projections of the Cora dataset using the t-SNE algorithm [39] for different values of p (with a perplexity of 50).

With AGC in [48], the authors proposed to first select an interval of possible values for p and then retain the first p that is a local minimum of an intra-cluster metric. Since our loss function contains information about the clustering performance, it can serve as a metric for the selection of p. Thus, similarly to AGC, we select the p via our loss function as follows: We stop and select $p = p^*$ if the change in square-root of the loss function $||\mathbf{T}^p\mathbf{X} - \mathbf{G}\mathbf{F}\mathbf{W}^\top||$ between

Algorithm 1: GCC

Input : - Adjacency matrix **A**
- Feature matrix **X**
- Propagation order p
- Number of clusters k
- Embedding dimension f
- Tolerance ϵ
- Maximum number of iterations max_iter

Output: - Membership indicator $\mathbf{G} \in \{0,1\}^{n\times k}$
- Embedded centers $\mathbf{F} \in \mathbb{R}^{k\times f}$
- Embedding matrix $\mathbf{W} \in \mathbb{R}^{d\times f}$

Compute **T** from **A**;
Initialize **W** with a randomized PCA on $\mathbf{T}^p\mathbf{X}$;
Initialize **G** with a k-means on $\mathbf{T}^p\mathbf{X}\mathbf{W}$;
while $\left\| \mathbf{T}^p\mathbf{X} - \mathbf{G}\mathbf{F}\mathbf{W}^\top \right\| > \epsilon$ **or** max_iter *not reached* **do**
  Update **F** using formula (7);
  Update **W** using formula (8);
  Update **G** using formula (10);
end

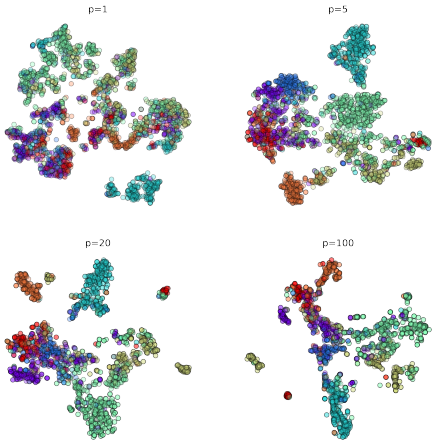


Figure 3: Visualization of the Cora GCC-embeddings using t-SNE for different values of p.

p^* and $p^* - 1$ is less than $\frac{d}{n}$ for $p \in \{0, \ldots, 150\}$. The detailed rule is described in Algorithm 2.

As our loss function w.r.t p is always decreasing for every dataset in the interval we chose. We stop when the change in the loss is lower than a constant that is a function of the input dimensions rather than wait for a local minimum.

4.3 Complexity Analysis

In what follows, we analyze the computational complexity of each operation in the GCC algorithm as well as the overall one.

Computing agg(A, X). The computational complexity of the p-th order simple graph convolution is $O(p|E|d)$ as each multiplication costs $|E|d$ and p such multiplications are needed.

Initializing W and G. Initializing **W** with PCA costs $O(nd\log(k))$ operations as claimed in [16]. For **G**, computing $\mathbf{T}^p\mathbf{X}\mathbf{W}$ takes $O(ndf)$

Algorithm 2: Propagation order selection rule

Input : - Adjacency matrix $\mathbf{A}$
- Feature matrix $\mathbf{X}$
- Number of clusters k
- Embedding dimension f

Output: Propagation order p^*

for $p \in \{2, \ldots, 100\}$ **do**
  $\mathbf{G}, \mathbf{F}, \mathbf{W} \leftarrow \texttt{GCC}(\mathbf{A}, \mathbf{X}, p, k, f)$;
  $\text{loss}_p \leftarrow \left\| \mathbf{T}^p\mathbf{X} - \mathbf{G}\mathbf{F}\mathbf{W}^\top \right\|$;
  if $| loss_p - loss_{p-1} | < \frac{d}{n}$ **then**
    $p* \leftarrow p - 1$
  end
end

while k-means applied on $\mathbf{T}^p\mathbf{X}\mathbf{W}$ is in $O(tnkf)$ where t is the number of iterations of k-means; ergo, the overall complexity of initialization is $O(nd\log(k) + ndf + tnkf)$.

Updating F. In (7), the cost of computing the embeddings matrix $\mathbf{T}^p\mathbf{X}\mathbf{W}$ is $O(ndf)$.

Since $\mathbf{G}$ is an indicator matrix, it can be stored as a vector rather than a matrix and multiplications that include it can be replaced by indexing operations. Thus, computing the transformation $(\mathbf{G}^\top\mathbf{G})^{-1}\mathbf{G}^\top$ takes $O(n+k)$ and applying it on the embeddings $\mathbf{T}^p\mathbf{X}\mathbf{W}$ costs $O(nf)$. Since $n > k$, the total complexity of the update of $\mathbf{F}$ is then $O(ndf)$.

Updating W. In (8), computing $(\mathbf{T}^p\mathbf{X})^\top\mathbf{G}\mathbf{F}$ for the SVD costs $O(ndf)$ because of G being indices. The SVD operation itself costs $O(df^2)$. As for calculating $\mathbf{V}\mathbf{U}^\top$, it is also in $O(df^2)$. This brings us to a total of $O(ndf + df^2)$ operations.

Updating G. $\mathbf{T}^p\mathbf{X}\mathbf{W}$ having already being computed, in (10) the complexity of computing $\mathbf{G}$ comes from searching each embedded vector's closest centroid, this takes $O(nkf)$.

Loss computation. Is in $O(dkf + nd)$ or $O(ndf)$ depending on the order of the multiplication and the indexing operation in the product $\mathbf{G}\mathbf{F}\mathbf{W}^\top$.

Overall complexity. The totality of the previous operations cost $O\big(p|E|d + (t'+t)nkf + t'(ndf + df^2 + dkf) + nd\log(k)\big)$ where t' is the number of iterations of our algorithm (generally converges within 5-15 iteration).

For simplicity's sake we assume $t' = t$, we can also assume that $k, f \leq \min(n, d)$, which is often the case in graph datasets (this condition can always be satisfied by adding duplicate nodes or constant features), this allows us to set $f = k$. Consequently, the total complexity is given as $O(p|E|d + tndk)$.

In comparison, computing $\mathbf{T}^p\mathbf{X}$ and applying a k-means on it takes $O(p|E|d+tndk)$, the same as our method. In practice, however, our algorithm is significantly faster than the k-means algorithm as its most theoretically heavy computations are matrix multiplications which can be efficiently performed on GPUs.

5 EXPERIMENTS

To evaluate our proposed model, we conduct experiments on four datasets and compare it against a number of state-of-the-art approaches for the node clustering task.

5.1 Datasets

We evaluate GCC on four widely-used attributed network datasets (Cora, Citeseer, Pubmed and Wiki). The nodes in Cora and Citeseer are associated with binary word vectors, while the ones in Pubmed and Wiki with tf-idf weighted word vectors. The summary statistics of the datasets are shown in table 1.

Table 1: Dataset statistics.

Dataset	#Nodes	#Edges	#Features	#Classes
CiteSeer [36]	3327	4732	3703	6
Cora [36]	2708	5429	1433	7
PubMed [36]	19717	44338	500	3
Wiki [45]	2405	17981	4973	17

5.2 A Fair Comparison with Baseline Methods

In our work we focus on clustering and related methods. Below we look at how our GCC algorithm performs in comparison with state-of-the art unsupervised methods. Approaches that use information from the actual labels, be it supervised or semi-supervised, are not considered such as [40]. The baseline methods are categorized as follows:

(1) **Methods that use node-level features only.** Spherical k-means [18] is k-means applied on data projected on the unit sphere. It will serve as the node features clustering baseline along with DCN [44].
(2) **Methods that use graph structure only.** Spectral is the spectral clustering algorithm with the normalized Laplacian as the input similarity matrix.
(3) **Methods that use both.** LVAE [33] is the linear graph variational autoencoder and LAE is its non-probabilistic version. GIC [27], AGE [8] proposes a Laplacian smoothing filter that acts as a low-pass filter applied in adaptive learning scheme. S²GC proposes a new method for the aggregation of K-hop neighborhoods that is a trade-off of low- and high-pass filter bands, it then applies spectral clustering on the output of that operation.

In the experiments we use the implementations that are publicly available on Github repositories of the authors.

5.3 Experimental Settings

To evaluate the clustering results, we employ three performance metrics: clustering Accuracy (Acc), Normalized Mutual Information (NMI) and clustering macro F1-score (F1). Larger values imply better performance. We report the mean values of the three metrics for each algorithm over 20 executions except for AGE which we average over three runs because of high execution time.

For our model, we set the embedding dimension $f = k$ for each dataset. The propagation parameter p is selected via the heuristic rule described prior; we obtain $p = 5$ for citeseer, $p = 12$ for Cora, $p = 150$ for Pubmed and $p = 4$ for Wiki. We row normalize the feature vectors and use tf-idf normalization on the binary word vectors of Citeseer and Cora so that all dataset are in tf-idf.

For other methods, we employ the parameters recommended by the authors for every dataset, For S²GC we expand the possible

Table 2: Clustering performance on four datasets averaged over 20 runs. AGE was averaged over 3 runs. AGE, LAE and LVAE failed to scale to Pubmed; OOM denotes out of memory.

Method	Input	Citeseer			Cora			Pubmed			Wiki		
		Acc	F1	NMI	Acc	F1	NMI	Acc	F1	NMI	Acc	F1	NMI
Sph. k-means	**X**	42.64	40.16	19.91	33.97	30.93	15.33	59.51	58.16	31.26	33.65	23.30	29.90
DCN	**X**	19.16	11.44	2.91	20.01	11.81	2.32	15.87	7.06	4.07	44.28	17.14	12.45
Spectral	**A**	21.60	9.46	1.54	30.00	8.78	2.36	58.96	43.53	18.30	23.20	13.74	18.05
LAE (2020)	(**A**, **X**)	43.49	41.33	22.66	65.43	66.21	48.89		OOM		45.26	40.90	45.99
LVAE (2020)	(**A**, **X**)	39.46	38.26	20.53	64.11	65.31	48.47		OOM		47.38	42.92	47.79
AGE (2020)	(**A**, **X**)	57.85	55.01	35.74	69.17	67.30	56.91		OOM		53.79	41.39	52.63
GIC (2021)	(**A**, **X**)	68.78	64.02	43.82	70.45	68.95	52.55	64.30	64.86	26.02	46.46	40.29	48.24
S²GC (2021)	(**A**, **X**)	68.13	63.79	42.26	69.68	66.41	54.83	70.81	**69.96**	**32.32**	52.71	44.40	48.96
GCC (ours)	(**A**, **X**)	**69.45**	**64.54**	**45.13**	**74.29**	**70.35**	**59.17**	**70.82**	69.89	32.30	**54.56**	**46.10**	**54.61**

Table 3: Wall-clock time in seconds for different methods on the four datasets averaged over 20 runs (3 runs for AGE).

Method	CiteSeer	Cora	Pubmed	Wiki
Sph. k-means	18.1	3.2	8.3	20.2
LAE	12.3	8.9	OOM	27.3
LVAE	11.9	6.3	OOM	29.3
AGE	2461	936.3	OOM	3058.7
GIC	8.4	5.7	13.9	8.3
S²GC	7.7	1.0	24.8	6.1
GCC	2.5	1.0	11.8	2.9

propagation order to $\{1, \ldots, 150\}$, the same as GCC for a fair comparison. All models were run on the same machine with a 12GB memory GPU an a RAM of 12GB. Note that we could not run AGE, LAE and LVAE on Pubmed due to out of memory (OOM) issues.

5.4 Clustering Results

Clustering performances of the different methods are reported in table 2. Best performances are shown in bold. It is clear that the methods that use both **A** and **X** perform significantly better than those that use either of them individually. We see how GCC outperforms other attributed graph clustering methods in terms of accuracy, F1 and NMI except for Pubmed where S²GC and GCC are comparable. The algorithm is also stable as its standard deviation on the accuracy is 0.13 on Citeseer, 0.02 on Cora, 0.00 on Pubmed, and 1.58 on Wiki.

We also report the average running time of each algorithm in table 3. Notice how the running times of our algorithm are the lowest on the four datasets when compared to the stare-of-the-art especially when comparing with the AGE algorithm. We mentioned earlier how algorithm has a similar theoretical complexity to that of k-means. We can see here that in practice our algorithm is faster on most datasets due to the fact that it can be efficiently run on a GPU.

5.5 Embedding and Visualization

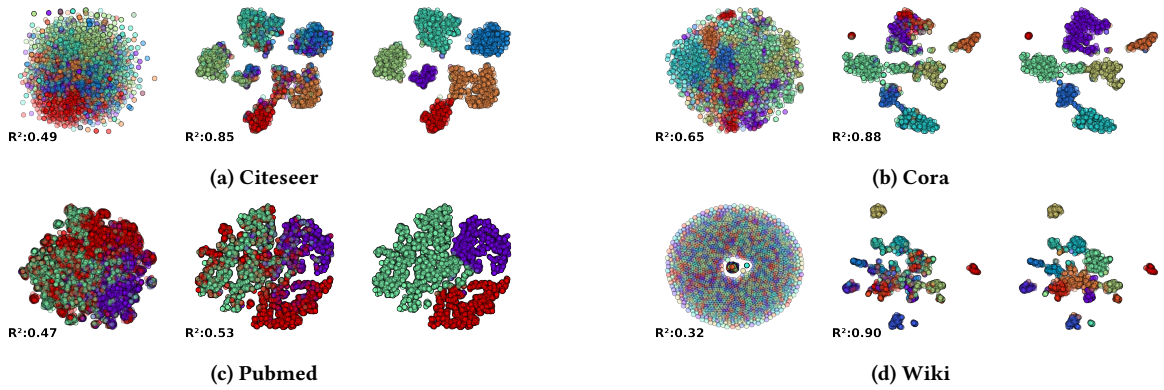
The GCC model offers the ability to display the cluster-based structure inherent to multivariate data. Figure 4 presents the lower-dimensional representations produced by our model projected on a 2-d space by t-SNE (with a perplexity of 50). We can see a clear difference in the structures of the projections of the raw data and those of the generated embeddings. To further judge the quality of the embedding, we use the R-squared measure, i.e., $R^2 = \frac{\mathrm{Tr}(S_b)}{\mathrm{Tr}(S_t)}$, where S_b is the between-class scatter matrix and S_t is the total scatter matrix. We report this measure in Figure 4 on the true labels plots to quantify separability. The R-squared is larger for 2-d projections of the GCC embeddings on all four datasets. On Pubmed, the structure is less pronounced (0.47 vs 0.53) but we can still see the formation of three clusters.

This shows how our model can be efficiently used for data visualization to generate more interpretable embeddings or for a dimensionality reduction step before feeding the output representations to more complex clustering algorithms down the line. Note also that such visualisations can help the user in assessing the number of clusters.

5.6 Choice of Propagation Matrix

We conduct experiments to further motivate our choice of propagation matrix. We compare the following propagation matrices: *i)* Augmented symmetric norm. $\mathbf{A}_{\text{sym}} = \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}$. *ii)* Augmented random walk norm. $\mathbf{A}_{\text{rw}} = \tilde{\mathbf{D}}^{-1}\tilde{\mathbf{A}}$. *iii)* Our norm. $\mathbf{A}_{\text{ours}} = \mathbf{T} = \mathbf{D_T}^{-1}(\mathbf{I} + \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2})$. We do an analysis on the clustering accuracy of our model with these normalizations for propagation orders $p \in \{1, .., 20\}$. These results are averaged over 20 runs and the same parameters are used for all normalizations.

We see in figure 5 how our proposed propagation matrix offers the maximum accuracy on three out of the four datasets ($\mathbf{A}_{\text{rw}}$ slightly outperforms it on Pubmed). We also see that it is more stable and well-behaved compared to the other two on all four datasets. The symmetric normalization especially is prone to large changes even for consecutive propagation orders. These results can be explained by the fact that the GCN when using the symmetric and random walk normalizations is not strictly low-pass. Figure 6 shows the frequency response functions for the GCN with the three propagation matrices for propagation order $p \in \{1, 2, 3\}$. We see how the absolute value of the frequency response function is not always decreasing w.r.t the frequencies for $\mathbf{A}_{\text{rw}}$ and $\mathbf{A}_{\text{sym}}$ as opposed to $\mathbf{A}_{\text{ours}}$.



(a) Citeseer (b) Cora

(c) Pubmed (d) Wiki

Figure 4: *Left column:* t-SNE projection of the original features colored according to the real labels. *Middle column:* t-SNE projection of the GCC embeddings colored according to the real labels. *right column:* t-SNE projection of the GCC embeddings colored according to the predicted labels. R-squared is used to measure of class separability for real classes (left and middle column), e.g., 0.49 vs 0.85 for Citeseer.

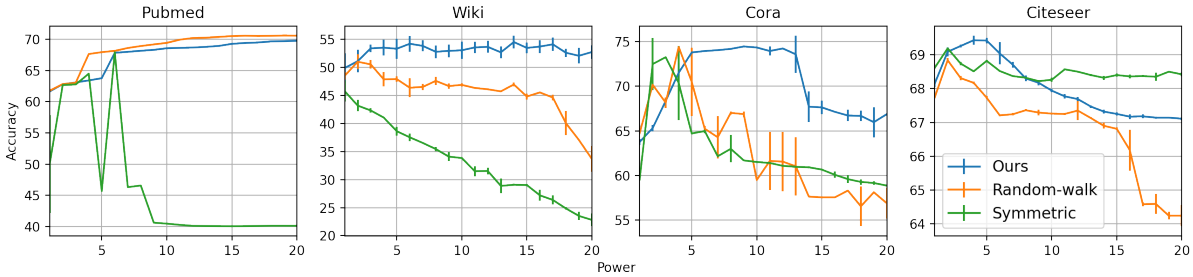


Figure 5: Accuracy with GCC using different propagation matrices averaged over 20 runs

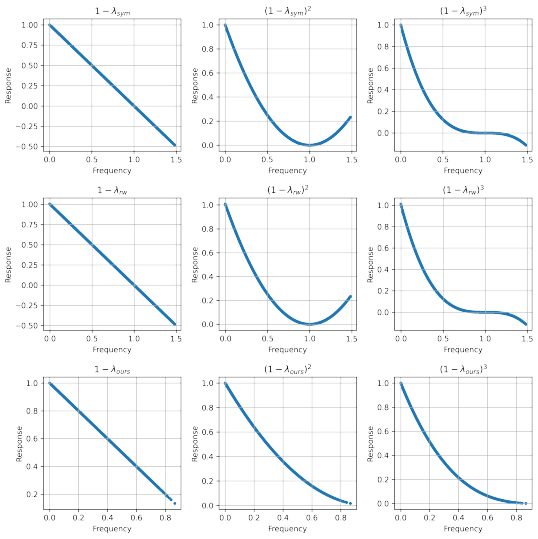


Figure 6: Frequency response plotted against the frequency for different propagation matrices on Cora. *Left column*: frequency response is $1-\lambda$. *Middle column*: frequency response is $(1-\lambda)^2$. *Right column*: frequency response is $(1-\lambda)^3$.

6 CONCLUSION

In this paper, we harnessed the simple formulation of the graph convolutional network to obtain an efficient model that addresses both node embedding and clustering in a unified framework. First, we provided a normalization that makes the GCN encoder act as a low pass filter in the strict sense. Secondly, we proposed a novel approach where the objective function to be optimized leverages information from both the GCN embedding reconstruction loss and the cluster structure of these embeddings. Thirdly, we derived GCC whose complexity has been rigorously studied. In doing so, we showed how GCC achieves better performances compared to other graph clustering algorithms in a more efficient manner. Note that all the compared methods are unsupervised in nature in order to have a fair comparison with our model. Our experiments demonstrated the interest of our approach. We also showed how GCC is related to other methods including some GCN variants.

The proposed model is flexible model and can be extended in several directions, thus opening up opportunities for future research. For instance, in our approach we have assumed that the α coefficient which regulates the trade-off between seeking reconstruction and clustering is equal to one, it would be interesting to investigate the choice of this value. On the other hand, while our focus in this work is clustering, it would be worthwhile to extend the problem, e.g., to co-clustering, which is a useful in a wide range of real-world scenarios like document clustering.

REFERENCES

[1] Amr Ahmed, Nino Shervashidze, Shravan Narayanamurthy, Vanja Josifovski, and Alexander J Smola. 2013. Distributed large-scale natural graph factorization. In *Proceedings of the 22nd international conference on World Wide Web*. 37–48.

[2] Kais Allab, Lazhar Labiod, and Mohamed Nadif. 2016. A semi-NMF-PCA unified framework for data clustering. *IEEE Transactions on Knowledge and Data Engineering* 29, 1 (2016), 2–16.

[3] Kais Allab, Lazhar Labiod, and Mohamed Nadif. 2018. Simultaneous spectral data embedding and clustering. *IEEE transactions on neural networks and learning systems* 29, 12 (2018), 6396–6401.

[4] Jinwon An and Sungzoon Cho. 2015. Variational autoencoder based anomaly detection using reconstruction probability. *Special Lecture on IE* 2, 1 (2015), 1–18.

[5] Bryan Perozzi Anton Tsitsulin, John Palowitch and Emmanuel Müller. 2020. Graph Clustering with Graph Neural Networks. In *Proceedings of the 16th International Workshop on Mining and Learning with Graphs (MLG)*.

[6] Yuki M. Asano, Christian Rupprecht, and Andrea Vedaldi. 2020. Self-labelling via simultaneous clustering and representation learning. In *International Conference on Learning Representations (ICLR)*.

[7] Mikhail Belkin and Partha Niyogi. 2003. Laplacian Eigenmaps for Dimensionality Reduction and Data Representation. *Neural Computation* 15, 6 (2003), 1373–1396.

[8] Ganqu Cui, Jie Zhou, Cheng Yang, and Zhiyuan Liu. 2020. Adaptive graph encoder for attributed graph embedding. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 976–985.

[9] Geert De Soete and J Douglas Carroll. 1994. K-means clustering in a low-dimensional Euclidean space. In *New approaches in classification and data analysis*. Springer, 212–219.

[10] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems* 29 (2016), 3844–3852.

[11] Inderjit S. Dhillon, Yuqiang Guan, and Brian Kulis. 2007. Weighted Graph Cuts without Eigenvectors A Multilevel Approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29, 11 (2007), 1944–1957.

[12] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. 2019. Graph Neural Networks for Social Recommendation. In *The World Wide Web Conference* (San Francisco, CA, USA) *(WWW '19)*. Association for Computing Machinery, New York, NY, USA, 417–426. https://doi.org/10.1145/3308558.3313488

[13] Maziar Moradi Fard, Thibaut Thonet, and Eric Gaussier. 2020. Deep k-means: Jointly clustering with k-means and learning representations. *Pattern Recognition Letters* 138 (2020), 185–192.

[14] Kamran Ghasedi Dizaji, Amirhossein Herandi, Cheng Deng, Weidong Cai, and Heng Huang. 2017. Deep clustering via joint convolutional autoencoder embedding and relative entropy minimization. In *Proceedings of the IEEE international conference on computer vision*. 5736–5745.

[15] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 855–864.

[16] 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.

[17] William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Representation Learning on Graphs: Methods and Applications. *IEEE Data Eng. Bull.* 40, 3 (2017), 52–74.

[18] Kurt Hornik, Ingo Feinerer, Martin Kober, and Christian Buchta. 2012. Spherical k-Means Clustering. *Journal of Statistical Software, Articles* 50, 10 (2012), 1–22.

[19] Yedid Hoshen. 2017. VAIN: Attentional Multi-agent Predictive Modeling. In *Neural Information Processing Systems (NIPS)*. 2701–2711.

[20] Md Rezaul Karim, Oya Beyan, Achille Zappa, Ivan G Costa, Dietrich Rebholz-Schuhmann, Michael Cochez, and Stefan Decker. 2020. Deep learning-based clustering approaches for bioinformatics. *Briefings in Bioinformatics* (2020), 1–23.

[21] Md Rezaul Karim, Michael Cochez, Achille Zappa, Ratnesh Sahay, Dietrich Rebholz-Schuhmann, Oya Beyan, and Stefan Decker. 2020. Convolutional Embedded Networks for Population Scale Clustering and Bio-ancestry Inferencing. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* (2020).

[22] Thomas N Kipf and Max Welling. 2016. Variational Graph Auto-Encoders. *NIPS Workshop on Bayesian Deep Learning* (2016).

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

[24] Lazhar Labiod and Mohamed Nadif. 2021. Efficient regularized spectral data embedding. *Advances in Data Analysis and Classification* 15, 1 (2021), 99–119.

[25] S. P. Lloyd. 1982. Least squares quantization in PCM. *IEEE Trans. Inf. Theory* 28 (1982), 129–136.

[26] Diego Marcheggiani and Ivan Titov. 2017. Encoding Sentences with Graph Convolutional Networks for Semantic Role Labeling. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Copenhagen, Denmark, 1506–1515.

[27] Costas Mavromatis and George Karypis. 2021. Graph InfoClust: Maximizing Coarse-Grain Mutual Information in Graphs. In *PAKDD (1)*. 541–553.

[28] Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. 2001. On Spectral Clustering: Analysis and an Algorithm. In *International Conference on Neural Information Processing Systems: Natural and Synthetic*. 849–856.

[29] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. DeepWalk: online learning of social representations. In *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 701–710.

[30] Xiaojuan Qi, Renjie Liao, Jiaya Jia, Sanja Fidler, and Raquel Urtasun. 2017. 3D Graph Neural Networks for RGBD Semantic Segmentation. In *2017 IEEE International Conference on Computer Vision (ICCV)*. 5209–5218.

[31] Benedek Rozemberczki, Ryan Davies, Rik Sarkar, and Charles Sutton. 2019. Gemsec: Graph embedding with self clustering. In *Proceedings of the 2019 IEEE/ACM international conference on advances in social networks analysis and mining*. 65–72.

[32] Aghiles Salah and Mohamed Nadif. 2017. Social regularized von Mises–Fisher mixture model for item recommendation. *Data Mining and Knowledge Discovery* 31, 5 (2017), 1218–1241.

[33] Guillaume Salha, Romain Hennequin, and Michalis Vazirgiannis. 2020. Simple and Effective Graph Autoencoders with One-Hop Linear Models. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD)*. 319–334.

[34] Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin Riedmiller, Raia Hadsell, and Peter Battaglia. 2018. Graph networks as learnable physics engines for inference and control. In *International Conference on Machine Learning*. 4470–4479.

[35] Victor Garcia Satorras and Joan Bruna Estrach. 2018. Few-Shot Learning with Graph Neural Networks. In *International Conference on Learning Representations*. https://openreview.net/forum?id=BJj6qGbRW

[36] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. 2008. Collective classification in network data. *AI magazine* 29, 3 (2008), 93–93.

[37] Linfeng Song, Yue Zhang, Zhiguo Wang, and Daniel Gildea. 2018. A Graph-to-Sequence Model for AMR-to-Text Generation. In *the Association for Computational Linguistics, ACL 2018*. 1616–1626.

[38] Fan-Yun Sun, Jordan Hoffman, Vikas Verma, and Jian Tang. 2020. InfoGraph: Unsupervised and Semi-supervised Graph-Level Representation Learning via Mutual Information Maximization. In *International Conference on Learning Representations*. https://openreview.net/forum?id=r1lfF2NYvH

[39] Laurens van der Maaten and Geoffrey E. Hinton. 2008. Visualizing High-Dimensional Data Using t-SNE. *Journal of Machine Learning Research* 9 (2008), 2579–2605.

[40] Petar Velickovic, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. 2019. Deep Graph Infomax. *ICLR (Poster)* 2, 3 (2019), 4.

[41] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 1225–1234.

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

[43] Michio Yamamoto and Heungsun Hwang. 2014. A general formulation of cluster analysis with dimension reduction and subspace separation. *Behaviormetrika* 41, 1 (2014), 115–129.

[44] Bo Yang, Xiao Fu, Nicholas D. Sidiropoulos, and Mingyi Hong. 2017. Towards K-means-friendly Spaces: Simultaneous Deep Learning and Clustering. In *Proceedings of the 34th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 70)*, Doina Precup and Yee Whye Teh (Eds.). 3861–3870.

[45] Cheng Yang, Zhiyuan Liu, Deli Zhao, Maosong Sun, and Edward Y. Chang. 2015. Network Representation Learning with Rich Text Information. In *IJCAI*.

[46] Jianwei Yang, Jiasen Lu, Stefan Lee, Dhruv Batra, and Devi Parikh. 2018. Graph R-CNN for Scene Graph Generation. , 690–706 pages.

[47] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 974–983.

[48] Xiaotong Zhang, Han Liu, Qimai Li, and Xiao-Ming Wu. 2019. Attributed Graph Clustering via Adaptive Graph Convolution. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, 4327–4333.

[49] Hao Zhu and Piotr Koniusz. 2021. Simple Spectral Graph Convolution. In *9th International Conference on Learning Representations, ICLR, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

[50] Wenwu Zhu, Xin Wang, and Peng Cui. 2020. Deep learning for learning graph representations. In *Deep Learning: Concepts and Architectures*. Springer, 169–210.