Social Network Analysis and Mining (2025) 15:14
https://doi.org/10.1007/s13278-025-01419-w

ORIGINAL ARTICLE

Refined graph encoder embedding via self-training and latent community recovery

Cencheng Shen[1] · **Jonathan Larson**[2] · **Ha Trinh**[2] · **Carey E. Priebe**[3]

Received: 8 August 2024 / Revised: 28 January 2025 / Accepted: 29 January 2025 / Published online: 14 March 2025
© The Author(s) 2025

Abstract
This paper introduces a refined graph encoder embedding method, enhancing the original graph encoder embedding through linear transformation, self-training, and hidden community recovery within observed communities. We provide the theoretical rationale for the refinement procedure, demonstrating how and why our proposed method can effectively identify useful hidden communities under stochastic block models. Furthermore, we show how the refinement method leads to improved vertex embedding and better decision boundaries for subsequent vertex classification. The efficacy of our approach is validated through numerical experiments, which exhibit clear advantages in identifying meaningful latent communities and improved vertex classification across a collection of simulated and real-world graph data.

Keywords Graph embedding · Discriminant analysis · Hidden community recovery

1 Introduction

Graph data has surged in popularity, serving as an ideal data structure for capturing interactions across diverse domains, including social networks, citation systems, communication networks, and physical and biological systems (Girvan and Newman 2002; Newman 2003; Barabasi AL, Oltvai ZN, 2004; Boccaletti et al. 2006; Varshney et al. 2011; Ugander et al. 2011). This rise is driven by the increasing availability of public graph datasets (Leskovec and Krevl 2014; Rossi and Ahmed 2015; Hu et al. 2020) and a growing interest in graph learning techniques, such as graph convolutional networks (GCNs) (Kipf and Welling 2017; Wu et al. 2019; Xu et al, 2019), node2vec (Grover and Leskovec 2016), and information network embedding (Tang et al. 2015). These methods have enabled numerous applications, such as knowledge graph embeddings (Dettmers et al. 2018), graph recommendation (He et al. 2020), label propagation (Wang and Leskovec 2022), cross-media retrieval (Yao et al. 2024), and contrastive learning for clustering (Zhao et al. 2024), among many others.

Many graph analysis methods can be viewed as graph embedding techniques, which project graph-structured data into low-dimensional spaces while preserving both structural and semantic properties. Among them, spectral embedding holds a foundational role in graph data analysis. It maps graph data into a d-dimensional space using either the graph adjacency matrix or the graph Laplacian (Rohe et al. 2011; Sussman et al. 2012; Priebe et al. 2019). Spectral embedding is particularly notable for its strong theoretical properties, as its vertex embeddings converge to the underlying latent positions under popular random graph models (Sussman et al. 2014; Athreya et al. 2018; Rubin-Delanchy et al. 2022). Consequently, spectral embedding provides a versatile and theoretically sound graph learning technique, with many works spanning vertex classification (Tang et al. 2013; Mehta et al. 2021), community detection (Mu et al. 2022; Gallagher et al. 2023), vertex nomination (Zheng et al. 2022), and the analysis of multiple graphs and time-series (Arroyo et al. 2021; Gallagher et al. 2021).

However, the scalability of spectral embedding has been a major bottleneck due to its use of singular value decomposition (SVD), which can be time-consuming for moderate to large graphs. To address this bottleneck, a recent method called one-hot graph encoder embedding (Shen et al. 2023b) has been proposed, offering a supervised extension

✉ Cencheng Shen
shenc@udel.edu

[1] University of Delaware, Newark, DE 19716, USA

[2] Microsoft Research, Redmond, WA 98052, USA

[3] Johns Hopkins University, Baltimore, MD 21218, USA

of spectral embedding. By incorporating vertex labels when available, this approach significantly improves computational efficiency. Furthermore, the graph encoder embedding often outperforms spectral embedding in finite-sample performance across diverse applications, such as large-scale dynamic graph analysis (Shen et al. 2024b), general graphs including distance and kernel matrices (Shen 2024a, b), and computing graph covariance and correlation (Shen et al. 2024a, 2025b).

In this paper, we propose an enhanced version of the graph encoder embedding that incorporates self-training and latent community recovery. The refined graph encoder embedding can detect hidden sub-communities within the given labels. It has the capability to detect all sub-communities if desired, or to selectively detect latent communities that can benefit subsequent vertex classification and prevent over-refinement. Using stochastic block models with varying parameter choices, we provide illustrative examples to show what the refined method can detect, the meaning of the recovered communities in context, and why detection shall be restricted for the benefit of vertex classification. Theoretical rationale is provided, and the proposed method is validated through simulations and a wide range of real graph data. All experiments are carried out on a local desktop with MATLAB 2024a, Windows 10, an Intel 16-core CPU, and 64GB of memory. The appendix includes algorithm pseudocode, theorem proofs, and additional numerical experiments. The code and data are available on GitHub.[1]

2 Review

In this section, we briefly review the stochastic block model (SBM), a classical random graph model. This is followed by an overview of the adjacency spectral embedding and the original graph encoder embedding.

2.1 Graph adjacency and stochastic block models

A graph consists of a set of vertices $\{v_i, i = 1, \dots, n\}$ and a set of edges $\{e_j, j = 1, \dots, s\}$, which can be succinctly represented by an $n \times n$ adjacency matrix $\mathbf{A}$. In this matrix, $\mathbf{A}(i,j) = 0$ indicates the absence of an edge between vertex i and j, while $\mathbf{A}(i,j) = 1$ indicates the existence of an edge. The adjacency matrix can also be weighted to reflect a weighted graph and, more generally, can represent any similarity or dissimilarity matrix, such as pairwise distance or kernel matrices.

Under SBM, each vertex i is assigned a label $\mathbf{Y}(i) \in \{1, \dots, K\}$. The probability of an edge between a vertex from class k and a vertex from class l is determined by a block probability matrix $\mathbf{B} = [\mathbf{B}(k,l)] \in [0,1]^{K \times K}$. For any $i \neq j$, it holds that

$$\mathbf{A}(i,j) \sim \text{Bernoulli}(\mathbf{B}(\mathbf{Y}(i), \mathbf{Y}(j))).$$

To generate an undirected graph, simply set $\mathbf{A}(j,i) = \mathbf{A}(i,j)$ for all $i < j$. More details on SBM can be found in Holland et al. (1983); Snijders and Nowicki (1997); Airoldi et al. (2008); Karrer and Newman (2011).

Moreover, the degree-corrected stochastic block model is an extension of the SBM that accounts for the sparsity observed in real graphs (Zhao et al. 2012; Yan et al. 2012). It assigns a degree parameter $\theta_i \in [0,1]$ to each vertex i. Given the degrees, each edge from vertex i to another vertex j is independently generated as follows:

$$\mathbf{A}(i,j) \sim \text{Bernoulli}(\theta_i \theta_j \mathbf{B}(\mathbf{Y}(i), \mathbf{Y}(j))).$$

2.2 Spectral embedding and encoder embedding

Spectral embedding is a classical method in machine learning (Ng et al. 2001; Belkin and Niyogi 2003). When applied to a graph adjacency matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$, it utilizes the singular value decomposition (SVD):

$$\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^T.$$

Let $\mathbf{S}_d$ be the first $d \times d$ submatrix of $\mathbf{S}$, and $\mathbf{V}_d$ be the first $n \times d$ submatrix of $\mathbf{V}$. The adjacency spectral embedding (ASE) is computed as:

$$\mathbf{Z}^{ASE} = \mathbf{V}_d \mathbf{S}_d^{0.5} \in \mathbb{R}^{n \times d},$$

where the ith row of $\mathbf{Z}^{ASE}$ represents the vertex embedding of vertex i. The Laplacian spectral embedding (LSE) follows the same formulation, except the adjacency matrix $\mathbf{A}$ is replaced by the corresponding graph Laplacian $\mathbf{L}$. For detailed discussions on graph spectral embedding and its theoretical properties under random graph models, see Rohe et al. (2011); Sussman et al. (2012); Priebe et al. (2019).

The graph encoder embedding (GEE) is a recent method that can be viewed as a supervised extension of spectral embedding, incorporating label information via a label vector $\mathbf{Y} \in [0, 1, \dots, K]^n$. Here, K denotes the total number of classes, and $\mathbf{Y}(i) = 0$ indicates that the label for vertex i is unknown. The method proceeds as follows:

- **Class Observation Count**: Compute the number of known labels for each class:

$$n_k = \sum_{i=1}^{n} 1(\mathbf{Y}(i) = k)$$

for $k = 1, \dots, K$.

[1] https://github.com/cshen6/GraphEmd

- **Normalized One-Hot Encoding Matrix**: Construct a normalized one-hot encoding matrix $\mathbf{W} \in [0,1]^{n\times K}$. For each vertex $i = 1, \ldots, n$, set:

  $$\mathbf{W}(i,k) = 1/n_k$$

  if $\mathbf{Y}(i) = k$, and 0 otherwise.
- **Embedding Computation**: Compute the embedding through a simple matrix multiplication:

  $$\mathbf{Z} = \mathbf{AW} \in [0,1]^{n\times K}.$$

  where $\mathbf{Z}(i,:)$ represents the embedding of vertex i.

Unlike spectral embedding, which derives a projection matrix via singular value decomposition, GEE directly utilizes the label vector to construct the projection matrix $\mathbf{W}$. Moreover, GEE shares similar theoretical properties with spectral embedding, including convergence to the underlying latent positions. For further details on its theoretical properties and iterative techniques for managing unknown labels, refer to (Shen et al. 2023a, b, 2025a).

3 Refined graph encoder embedding

3.1 Linear transformation for self-training

We first outline the steps for applying linear discriminant analysis (LDA) to transform the graph encoder embedding, enabling self-training on the embedding as follows:

- **Input**: The graph adjacency matrix $\mathbf{A} \in \{0,1\}^{n\times n}$ and a label vector $\mathbf{Y} \in \{0,1,\ldots,K\}^n$, where 1 to K represent known labels and 0 is a dummy category for vertices with unknown labels.
- **Graph Encoder Embedding**: Compute the graph encoder embedding $\mathbf{Z}$ as described in Sect. 2.2.
- **Linear Transformation**: Apply LDA to $\mathbf{Z}$. Specifically, let $\mu_k \in \mathbb{R}^K$ denote the class-conditional mean:

  $$\mu_k = E(\mathbf{Z}(i,:)|\mathbf{Y}(i) = k)$$

  for $k = 1, \ldots, K$. Denote $\mu = [\mu_1, \mu_2, \ldots, \mu_K] \in \mathbb{R}^{K\times K}$ as the matrix of concatenated means, and let $\Sigma \in \mathbb{R}^{K\times K}$ be the estimated common covariance matrix of $\mathbf{Z}$. Denote Σ^+ as the pseudo-inverse of Σ, $\vec{n}_K$ as a row vector where the kth entry is n_k, and diag $(\cdot)$ as an operator that extracts the diagonal terms of a matrix as a row vector. The linearly transformed GEE embedding is computed as:

  $$\mathbf{Z}_1 = \mathbf{Z}\Sigma^+\mu - (\text{ diag }(\mu'\Sigma^+\mu) - \text{ log }(\vec{n}_K/n)). \quad (1)$$

  Here, the term after the first minus sign is a row vector, subtracted from each row of $\mathbf{Z}\Sigma^+\mu$. Equation 1 simply represents the linear discriminant analysis expressed as a single matrix equation.
- **Self-Training**: Compute the updated label vector $\mathbf{Y}_1$ and a mismatch indicator vector δ_1 as follows:

  $$\mathbf{Y}_1(i) = \arg\max_{k=1,\ldots,K} \mathbf{Z}_1(i,k)$$

  $$\delta_1(i) = 1\{\mathbf{Y}_1(i) \neq \mathbf{Y}(i) \text{ and } \mathbf{Y}(i) > 0\},$$

  where $\delta_1(i)$ indicates whether the original label $\mathbf{Y}(i)$ and the self-trained label $\mathbf{Y}1(i)$ are mismatched.
- **Output**: The linear transformed encoder embedding $\mathbf{Z}_1 \in \mathbb{R}^{n\times K}$, the updated label vector $\mathbf{Y}_1 \in \{1,\ldots,K\}^n$, and an indicator vector $\delta_1 \in [0,1]^n$.

The purpose of GEE with LDA is to align the original GEE such that the dimension attaining the maximum value determines the class assignment for each vertex. While one could use a neural network with softmax outputs for this purpose, the graph encoder embedding is approximately normally distributed, as stated in Theorem 1. Therefore, LDA is a suitable and more computationally efficient choice for estimating conditional probabilities and aligning the embedding. A pseudocode version of the algorithm is provided in appendix algorithm 1.

3.2 Refined GEE via self-training and latent community recovery

We present the main method, which outputs a refined graph encoder embedding along with potential latent community information, through iterative self-training and latent community discovery:

- **Input**: The graph adjacency matrix $\mathbf{A} \in \{0,1\}^{n\times n}$ and a label vector $\mathbf{Y} \in \{0,1,\ldots,K\}^n$, where 1 to K represent known labels and 0 is a dummy category for vertices with unknown labels. Additional parameters include the maximum number of refinements for self-training (γ_Y) and latent community discovery (γ_K), set to 5 by default; stopping criteria $\epsilon \in [0,1]$ and $\epsilon_n \in \mathbb{N}$, set to 0.3 and 5 by default.
- **Initial Embedding**: Apply LDA to GEE as in Sect. 3.1, yielding the vertex embedding $\mathbf{Z}_1$, the updated label vector $\mathbf{Y}_1$, and an indicator vector δ_1.
- **Refinement via Self-Training**: Update the vertex embedding using $\mathbf{A}$ and the new label vector $\mathbf{Y}_1$. Denote the outputs as $\mathbf{Z}_2$, $\mathbf{Y}_2$, and δ_2.
- **Stopping Criterion**: Check whether δ_1 and δ_2 are sufficiently close, defined by:

  $$\sum_{i=1}^{n} \delta_1(i) - max(\sum_{i=1}^{n} \delta_1(i) * \epsilon, \epsilon_n) < \sum_{i=1}^{n} \delta_1(i) \cdot \delta_2(i). \quad (2)$$

If the condition holds, proceed to the next step. Otherwise, set $\mathbf{Y}_1 = \mathbf{Y}_2$ and repeat the previous step until the iteration limit γ_Y is reached. After each iteration, concatenate embeddings and labels:

$\mathbf{Z}_1 = [\mathbf{Z}_1, \mathbf{Z}_2], \mathbf{Y} = [\mathbf{Y}, \mathbf{Y}_1]$.

- **Refinement via Latent Communities**: Assign mismatched vertices to a new class, update the vertex embedding using $\mathbf{A}$ and $\mathbf{Y}_1 + \delta_1 * K$. Denote the outputs as $\mathbf{Z}_2$, $\mathbf{Y}_2$, and δ_2. Repeat this process until the stopping criterion (Eq. 2) is satisfied or the iteration limit γ_K is reached. Concatenate intermediate embeddings and labels as in the previous step.
- **Output**: The refined graph encoder embedding $\mathbf{Z}$ and the concatenated label matrix $\mathbf{Y}$.

As an example of the refinement step, suppose the initial label vector is $\mathbf{Y}_1 = [1, 1, 1, 2, 2, 2]$, with $K = 2$. If the 1st and 4th vertices are identified as mismatched according to δ_1, their labels are updated by assigning each to a new group. The updated label vector becomes $\mathbf{Y}_2 = [3, 1, 1, 4, 2, 2]$, where the new label for a mismatched vertex is determined by incrementing its original label by K. This refinement process continues iteratively, assigning new labels to mismatched vertices, until the stopping criterion is satisfied.

The iterative self-training phase terminates when the mismatch between the input and self-trained labels becomes insignificant, as determined by the parameters ϵ and ϵ_n. Specifically, Eq. 2 requires that the updated mismatch indicator δ_2 yields at least a 30% reduction (or a minimum of 5 indices) compared to the previous mismatch indicator δ_1. Similarly, during latent community discovery, mismatched vertices are iteratively reassigned to new classes until the mismatch indicator no longer shows a significant reduction. At termination, the method outputs the original encoder embedding along with all refined embeddings from both self-training and latent community assignment, concatenated into a single matrix.

The parameters γ_K and γ_Y control the number of iterations, while ϵ and ϵ_n determine how aggressive the refinement process is by assessing whether the next refinement is sufficiently similar to the previous one. Larger values of ϵ and ϵ_n cause the algorithm to stop more easily, while smaller values make the algorithm harder to stop, leading to more aggressive refinement. For example, if the downstream task involves visualizing all hidden communities or detecting outliers, one can set ϵ and ϵ_n to their smallest possible values (e.g., 0) and γ_K and γ_Y to large values (e.g., 100). This ensures that the refinement process continues as long as mismatched indices keep decreasing. Our default parameter choices are designed to be slightly conservative and have performed well across simulations and real data experiments for vertex classification. A pseudocode version of the main method is provided in appendix algorithm 2, along with additional simulations exploring parameter sensitivity.

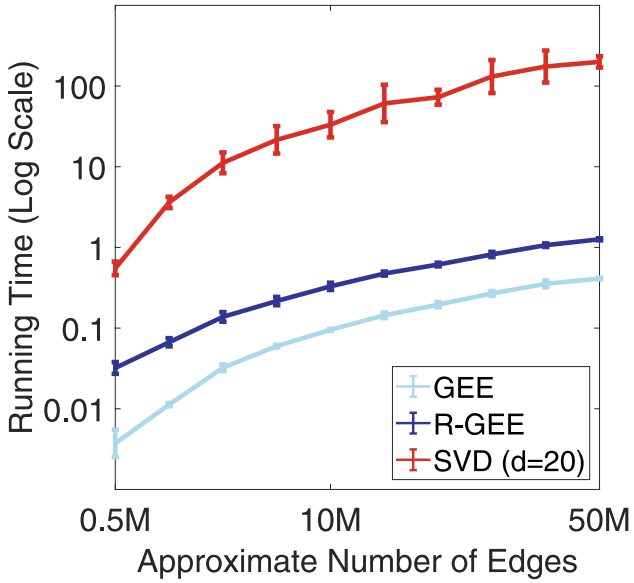


Fig. 1 This figure shows the running time comparison between GEE, Refined GEE, and SVD. The X-axis represents the approximate number of edges, and the Y-axis represents the running time on a log-10 scale

3.3 Running time analysis

The original graph encoder embedding (GEE) has a time complexity of $O(nK + s)$ (Shen et al. 2023b), where s is the number of edges, making it linear with respect to the number of vertices and edges. Let K_M be the largest possible number of refined classes, the refined encoder embedding (R-GEE) in algorithm 2 has a time complexity of $O(nK_M + nK_M^2 + s)$, where the quadratic term K_M^2 comes from using linear discriminant. As $K_M = \gamma_K K$, or at most $5K$ in the default parameter, the method remains linear with respect to the number of vertices and edges.

Figure 1 shows the running time using simulation model 3 with sparse adjacency matrix input (see Sect. 5 for model details), as n increases from 3000 to 30000. The average running time and standard deviation are reported based on 10 Monte Carlo replicates, performed on a local desktop running MATLAB 2024a on Windows 10, equipped with an Intel 16-core CPU and 64GB of memory. It is clear that the R-GEE, although slower than the original GEE, is still vastly faster than singular value decomposition, which is the major computational step of spectral embedding. At $n = 30000$, the number of edges is about 50 million; a single SVD into $d = 20$ requires about 200 seconds, while the graph encoder

embedding takes 0.4 second and the refined method takes 1.2 second.

4 Theoretical rationale

In this section, we provide the theoretical rationale for the self-training by linear transformation and the latent community assignment, explaining why the proposed method works and when community refinement helps improve embedding quality. All proofs are provided in the appendix.

Theorem 1 *The graph encoder embedding is asymptotically normally distributed under SBM. Specifically, as n increases, for a given ith vertex of class y, it holds that*

$$\text{diag}\,(\vec{n}_K)^{0.5} \cdot (\mathbf{Z}(i,:) - \mu_y) \xrightarrow{n} \mathcal{N}(0, \Sigma_y).$$

The expectation and covariance are: $\mu_y = \mathbf{B}(y,:)$ *and* $\Sigma_y(k,k) = \mathbf{B}(y,k)(1-\mathbf{B}(y,k))$. *Assuming* Σ_y *is the same across all* $y \in [1,K]$, *the transformation in Eq. 1 satisfies*

$$\mathbf{Z}_1(i,k) \xrightarrow{n} Prob(Y = k | X = \mathbf{Z}(i,:)).$$

Theorem 1 shows that the original graph encoder embedding is asymptotically normally distributed. As a result, the proposed linear transformation approximates the conditional probability, making it an appropriate choice for subsequent self-training.

Theorem 2 *Suppose the graph is distributed as the stochastic block model with block probability* $\mathbf{B} \in \mathbb{R}^{K \times K}$ *and observed label vector* $\mathbf{Y} \in [1, \ldots, K]$. *Then for any two vertices i, j, the encoder embedding* $\mathbf{Z}$ *using observed labels satisfies*:

$$\|\mathbf{Z}(i,:) - \mathbf{Z}(j,:)\|_2 - \|\mathbf{B}(\mathbf{Y}(i),:) - \mathbf{B}(\mathbf{Y}(j),:)\|_2 \xrightarrow{n} 0$$

Suppose the same graph can be viewed as a realization of a latent stochastic block model with $\mathbf{B}_0 \in \mathbb{R}^{K_0 \times K_0}$ *and a latent label vector* $\mathbf{Y}_0 \in [1, \ldots, K_0]$ *where* $K_0 > K$. *The for the same two vertices i, j, the resulting encoder embedding* $\mathbf{Z}_0$ *using the latent labels satisfies*:

$$\|\mathbf{Z}_0(i,:) - \mathbf{Z}_0(j,:)\|_2 - \|\mathbf{B}_0(\mathbf{Y}_0(i),:) - \mathbf{B}_0(\mathbf{Y}_0(j),:)\|_2 \xrightarrow{n} 0$$

Theorem 2 suggests that when comparing the encoder embedding using observed labels versus the encoder embedding using latent labels, the margin of separation fully depends on the block probability vector between the observed model and the latent model. This means that, from a margin separation perspective, the refined encoder embedding using latent communities could perform better or worse than the original encoder embedding using observed communities. Therefore, for the refined GEE to improve over the original GEE, it needs to properly decide whether to refine the given labels or not. Moreover, it is important to concatenate the embedding in each refinement, because the concatenated embedding retains previous embedding information and is more robust against slight over-refinement.

Under the stochastic block model, this theorem can help verify whether the latent community leads to an improvement or deterioration in the margin of separation over the observed community. Note that the theorem focuses on asymptotic behavior. In finite-sample performance, the embedding variance certainly plays a role in the decision boundary. In this paper, we only considered the mean difference to illustrate the key idea, for simplicity of presentation and to avoid overly complicating mathematical expressions. This is because the variance is generally similar across the groups and bounded above by 0.25 in SBM.

5 Simulations

We start with three stochastic block models, each serving as a representative case, and use Theorem 2 to verify whether the latent community leads to better embedding separation among groups. We then use embedding visualization, vertex classification, and precision/recall metrics to verify the results and assess the effectiveness of the refined algorithm.

5.1 Model parameters

5.1.1 Simulated graph 1

For each vertex, we let the latent communities as $Y_0 \in \{1,2,3,4\}$ with probability 0.25 each, set the latent block probability matrix as

$$\mathbf{B}_0 = \begin{bmatrix} 0.5, 0.2, 0.1, 0.1 \\ 0.2, 0.2, 0.1, 0.1 \\ 0.1, 0.1, 0.2, 0.2 \\ 0.1, 0.1, 0.2, 0.5 \end{bmatrix},$$

and generate the degree parameter by $\theta_i \overset{i.i.d.}{\sim} Uniform(0.1, 1)$. Next, we set the observed communities as $Y = 1$ if $Y_0 \in \{1,2\}$, and $Y = 2$ if $Y_0 \in \{3,4\}$. Namely, the first two latent communities are observed as one group, while the last two latent communities are observed as another group. Therefore, the observed block probability matrix can be computed as

$$\mathbf{B} = \begin{bmatrix} 0.275, 0.1 \\ 0.1, 0.275 \end{bmatrix}.$$

Now we use Theorem 2 to check the margin of separation. When using the latent labels, the margin of separation between classes 2 and 3 equals

$\|(0.2, 0.2, 0.1, 0.1) - (0.1, 0.1, 0.2, 0.2)\| = 0.2$. When using the observed labels, the difference is $\|(0.275, 0.1) - (0.1, 0.275)\| = 0.25$. Therefore, using the observed labels actually provides a larger margin of separation between these vertices. Note that if we consider the separation between classes 1 and 4, then the latent communities are better; however, those two latent groups are less important than the separation between latent classes 2 and 3.

5.1.2 Simulated graph 2

The latent communities and block probability matrix are exactly the same as in simulated graph 1, except the observed communities are set up as follows: $Y = 1$ if $Y_0 \in \{1, 3\}$, and $Y = 2$ if $Y_0 \in \{2, 4\}$. As a result, the observed block probability matrix can be computed as

$$\mathbf{B} = \begin{bmatrix} 0.225, 0.15 \\ 0.15, 0.225 \end{bmatrix}.$$

In this case, the latent communities have a margin of 0.2 between latent class 2 and 3, which becomes 0.11 when using observed communities. Therefore, this simulation provides an example where label refinement is necessary and significantly improves the embedding quality.

5.1.3 Simulated graph 3

In this simulation, we set the latent communities as $Y_0 \in \{1, 2, 3, 4, 5\}$ with probability 0.2 each, and set the latent block probability matrix as

$$\mathbf{B}_0 = \begin{bmatrix} 0.5, 0.2, 0.2, 0.1, 0.1 \\ 0.1, 0.2, 0.1, 0.2, 0.1 \\ 0.1, 0.1, 0.2, 0.1, 0.2 \\ 0.1, 0.2, 0.1, 0.5, 0.1 \\ 0.1, 0.1, 0.2, 0.1, 0.5 \end{bmatrix}.$$

The observed communities are: $Y = 1$ if $Y_0 \in \{1, 2, 3\}$; $Y = 2$ if $Y_0 = 4$; and $Y = 3$ if $Y_0 = 5$. Then the observed block probability matrix can be computed as

$$\mathbf{B} = \begin{bmatrix} 0.178, 0.133, 0.133 \\ 0.133, 0.500, 0.100 \\ 0.133, 0.100, 0.500 \end{bmatrix}$$

This simulation is somewhat similar to simulated graph 1 but presents an interesting mixed situation where some decision boundaries are improved using latent groups, while others are worse. For example, the difference between vertices in latent group 2 and 4 is 0.3 using latent labels, which is enlarged to 0.37 using observed labels. However, class 1 and 4 are separated by 0.5745 using latent labels, which is reduced to 0.37 using observed labels, and similarly for class 1 versus 5, or class 4 versus 5.

5.2 Latent community recovery

Figure 2 shows the graph connectivity for simulated graphs 1 and 2, with vertices colored by latent community (left), observed community (center), and GEE-refined community (right), which uses the label output via one label refinement of R-GEE ($\gamma_K = 1$ and $\gamma_Y = 0$).

Since the latent communities do not improve the embedding separation for simulated graph 1, we expect R-GEE to largely ignore the latent communities. This is indeed the case in the top row of Fig. 2, where the refinement only highlights a few vertices in the middle, and most vertices remain in their observed groups.

The situation is different for simulated graph 2, where the latent communities significantly improve the embedding quality and decision boundary. In this case, R-GEE successfully identifies the latent communities, assigning most vertices in latent communities 2 and 3 to different groups, so the right panel closely matches the left panel in the bottom row of Fig. 2. To maintain a clear and consistent visualization, simulation graph 3 is not shown here, as it merely represents a mixed case between graph 1 and graph 2.

5.3 Vertex classification evaluation

The top row of Fig. 3 reports the 10-fold cross-validation for the simulated graphs using 30 replicates. For each replicate, we generate a simulated graph of increasing vertex size, along with the corresponding latent and observed labels.

GEE0 computes the original GEE using the latent community labels Y_0. GEE computes the original GEE using the observed community labels Y. R-GEE uses the proposed algorithm with default parameters and the observed community labels Y as input. ASE stands for adjacency spectral embedding into $d = 20$. Each method is then evaluated via an LDA classifier for the observed labels Y. Note that the classification task is always for the observed labels Y, and the latent labels Y_0 are only used for embedding. Moreover, for all GEE methods, the labels of testing observations are assigned to 0 prior to the embedding.

For simulated graph 1, GEE0 using the latent labels has the worst classification error, while all other methods perform well and similarly to each other. This result matches the model setting and our previous verification that latent communities yield worse embedding quality. For simulated graph 2, GEE using the observed labels performed the worst, while R-GEE, GEE0, and ASE all performed very well. This is a reversal of simulated graph 1 and also matches the model setting and previous verification that latent communities improve the embedding quality in this case. For

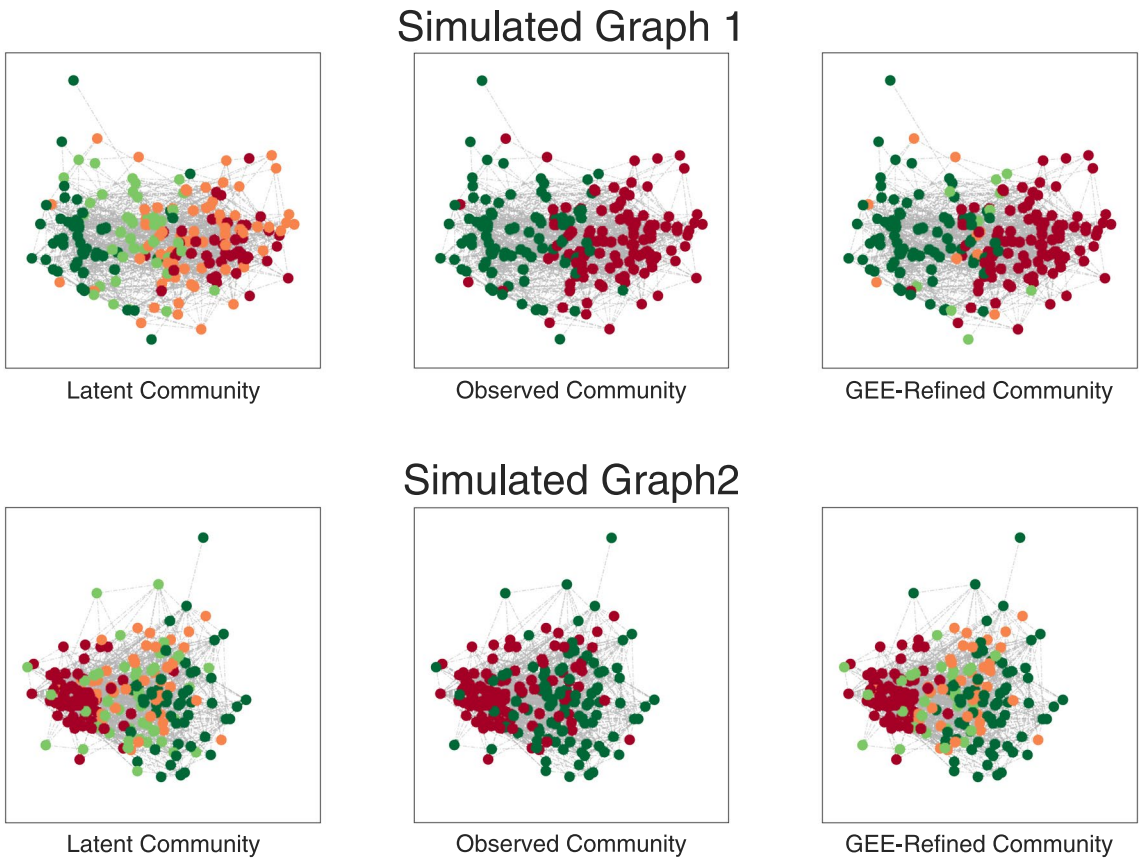


Fig. 2 This figure visualizes the graph using latent labels (left panel), observed labels (center panel), and GEE-refined labels after one refinement iteration (right panel). In the left panel, the colors dark green, light green, red, and orange represent the four ground-truth latent communities. In the center panel, the dark green and light green vertices from the left panel are combined into a single observed community, colored dark green, while the red and orange vertices are similarly merged into a single observed community, colored red. In the right panel, the R-GEE algorithm refines the observed labels from the center panel and partially recovers the ground-truth latent communities, with the refined communities again represented by light green and orange. (Colour figure online)

simulated graph 3, it is a mixed case where some refinement helps marginally, and GEE0, GEE, and R-GEE all performed relatively well with some small differences.

The bottom row of Fig. 3 shows the precision and recall of R-GEE. In simulated graphs 1 and 3, the vertex classification results indicate that latent communities are not important, so while the precision is high (all discovered new communities belong to the true latent communities), the recall is relatively low (many vertices from the latent communities are not discovered). For simulated graph 2, discovering the latent communities is critical, and indeed both precision and recall are very high, showing that R-GEE is performing as intended.

Overall, this figure shows that the proposed algorithm works as designed, recovering latent communities only when they are useful for vertex classification, and retaining excellent embedding quality that is not overly refined, as evidenced by the good classification error that converges to 0 in every case.

6 Real data evaluation

We collected a diverse set of real graphs with associated labels from various sources, including the Network Repository[2] (Rossi and Ahmed 2015), Stanford network data,[3] and other public graph data. Specifically, we experimented on the adjnoun graph (Newman 2003), C. elegans neuron data, which provides two binary graphs (Pavlovic et al. 2014), the EU email network (Yin et al. 2017), the karate club graph (Zachary 1977), the lastFM Asia social network (Rozemberczki and Sarkar 2020), the letter graph, the political blogs graph (Adamic and Glance 2005), a political retweet graph, the Pubmed Citation network, and a Wikipedia article graph (Shen et al. 2014) with four graphs.

[2] http://networkrepository.com/

[3] https://snap.stanford.edu/

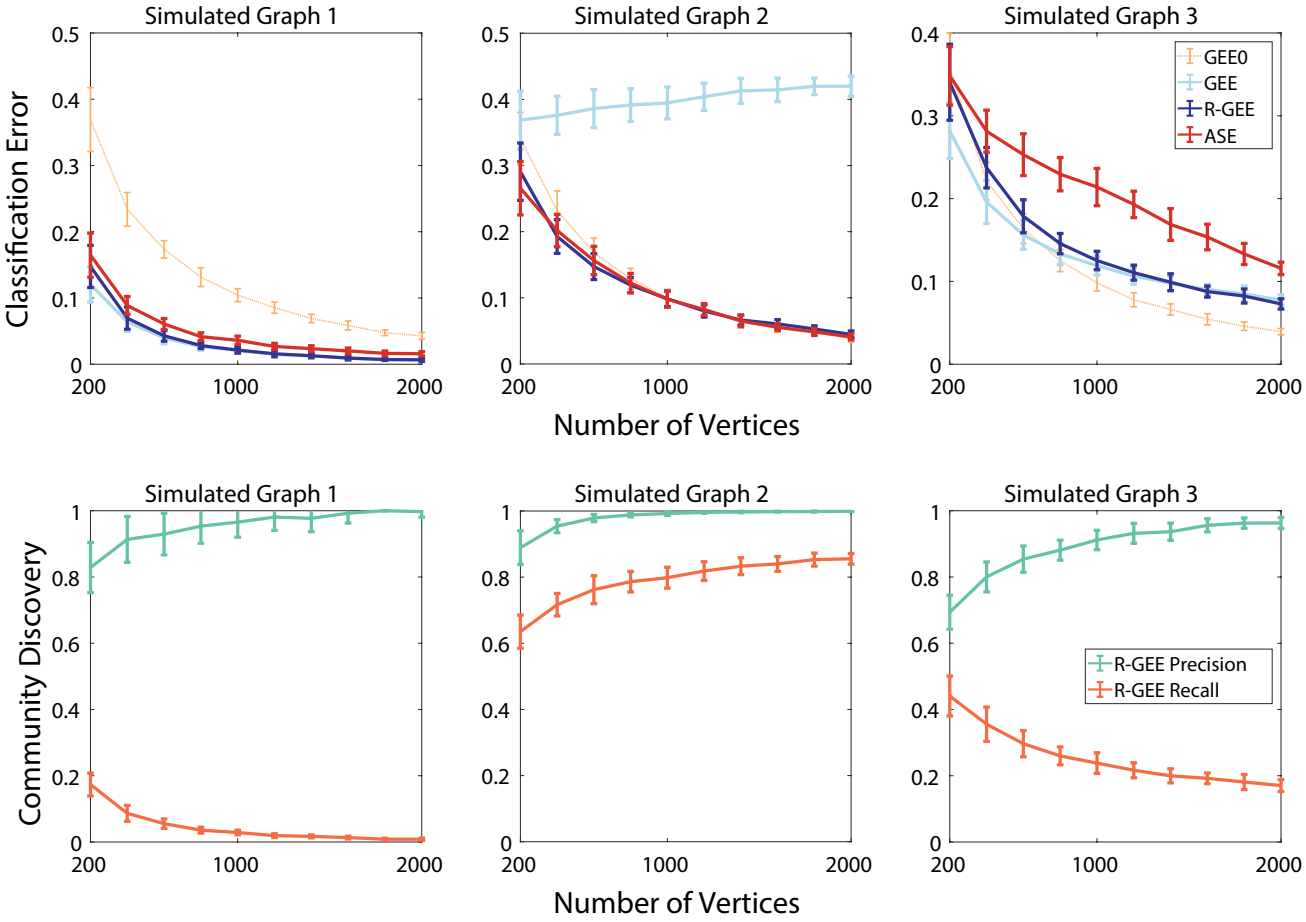


Fig. 3 The first row of the figure reports the 10-fold cross-validation error and standard deviation for the three simulated graphs, using 30 replicates. The bottom row of the figure reports the precision and recall for refined GEE in recovering the latent communities

6.1 Vertex classification

For a more comprehensive evaluation in the real data experiments, we compared GEE, R-GEE, ASE, LSE, and node2vec. R-GEE used the default parameters; ASE and LSE project into $d = 20$ dimensions; node2vec uses the graspy package (Chung and D. PB, Bridgeford EW, 2019) with default parameters and 128 dimensions. For each dataset and each method, we carried out 10-fold validation and reported the average classification error using LDA, along with one standard deviation, in Table 1 with 30 random seeds. Any directed graph was transformed to undirected, and any singleton vertex was removed. Note that unlike the simulated graphs, real graphs do not come with any known latent communities.

Table 1 clearly shows that refined GEE is able to preserve or improve the classification error compared to original GEE. In a few cases where it is worse, the difference is only marginal. Moreover, GEE and R-GEE are either the best or very close to the best in terms of classification error across all real data experiments. It should be noted that all methods with parameters could attain better performance if we tuned the parameters for each real dataset, but we chose to use consistent parameter choices throughout the experiments. Therefore, the results reported here should be viewed as a conservative illustration of the proposed method, not the best possible error.

6.2 Refinement visualization on real data

Figure 4 illustrates the community refinement results for two representative cases: the karate club graph and the political blogs graph. R-GEE successfully detects useful hidden communities with just one refinement: for the karate club graph, the algorithm identifies an anomaly vertex that always connects with the other group, and another vertex located at the intersection between the two classes; for the political blogs, our algorithm identifies blogs that are dominantly connected to the other party. Whether these are "swinger" blogs or "imposter" blogs is an issue of practical importance.

While the refined embedding is undoubtedly useful for uncovering potential hidden communities, it is important to

Table 1 This table reports the 10-fold vertex classification error and standard deviation for real graphs, using 30 random replicates. All numbers are in percentile. Note that there are two text dissimilarity datasets (Wiki TE and Wiki TF, which are cosine dissimilarity of the underlying articles) where node2vec is not applicable

	(n, K)	R-GEE	GEE	ASE	LSE	N2V
AdjNoun	(112, 2)	14.9 ± 2.1	14.6 ± 2.0	18.7 ± 1.3	**14.2** ± 1.4	45.9 ± 3.3
C-Elegans Ac	(253, 3)	37.1 ± 2.7	49.7 ± 1.5	38.4 ± 1.3	**35.3** ± 1.3	45.3 ± 2.0
C-Elegans Ag	(253, 3)	**38.0** ± 2.3	40.8 ± 1.7	42.3 ± 1.1	42.6 ± 1.2	40.2 ± 2.3
Coil-RAG	(11687, 100)	**19.5** ± 1.1	**19.5** ± 1.1	97.3 ± 0.1	95.5 ± 0.1	79.1 ± 0.2
Email	(1005, 42)	29.4 ± 1.1	33.1 ± 0.5	44.4 ± 0.4	88.5 ± 0.4	**28.9** ± 0.4
Karate	(34, 2)	**9.5** ± 2.3	**9.5** ± 2.3	17.4 ± 4.6	16.6 ± 4.2	13.8 ± 2.7
LastFM	(7624, 18)	17.5 ± 0.5	18.1 ± 0.2	48.9 ± 0.1	23.1 ± 0.1	**14.7** ± 0.1
Letter	(10482, 26)	**3.2** ± 0.2	**3.2** ± 0.2	89.9 ± 0.2	88.9 ± 0.2	74.7 ± 0.2
PolBlogs	(1224, 2)	5.3 ± 0.3	5.0 ± 0.3	9.5 ± 0.2	**4.9** ± 0.2	5.1 ± 0.1
PolTweet	(1847, 2)	**2.6** ± 0.1	2.7 ± 0.1	29.8 ± 0.1	4.6 ± 0.1	38.8 ± 0.1
PubMed	(19716, 3)	**20.3** ± 0.1	20.4 ± 0.1	37.4 ± 0.1	34.0 ± 0.1	58.8 ± 0.2
Wiki TE	(1382, 5)	**16.0** ± 0.5	20.4 ± 0.3	26.2 ± 0.2	26.6 ± 0.2	n/a
Wiki TF	(1382, 5)	**15.8** ± 0.5	20.9 ± 0.3	27.7 ± 0.3	27.7 ± 0.3	n/a
Wiki GE	(1382, 5)	**33.5** ± 0.9	41.2 ± 0.5	46.4 ± 0.4	53.8 ± 0.4	40.2 ± 0.5
Wiki GF	(1382, 5)	**42.9** ± 0.8	50.6 ± 0.8	47.1 ± 0.3	56.7 ± 0.5	46.6 ± 0.4

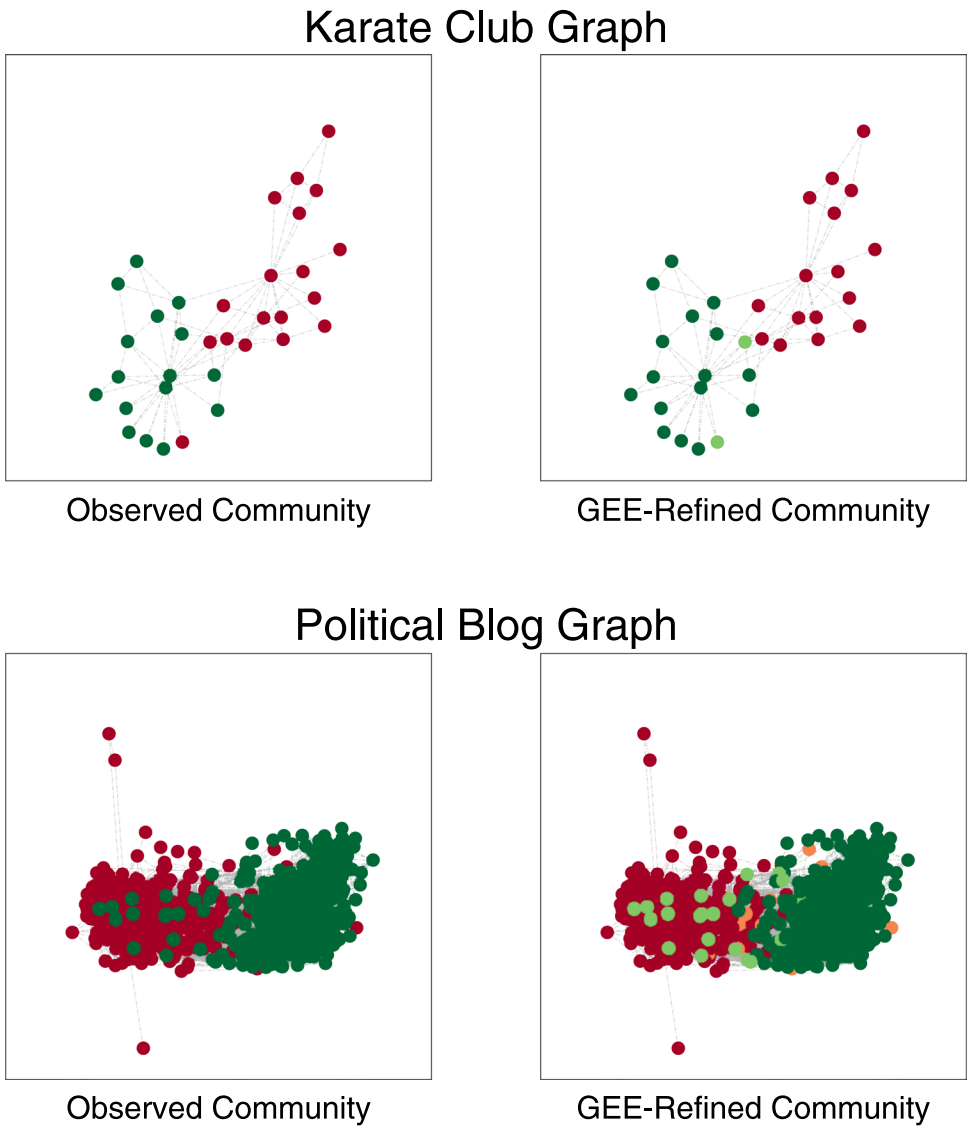


Fig. 4 This figure visualizes two real graphs, the karate club and political blogs, using observed labels (left panel) and GEE-refined labels after one refinement iteration (right panel). In the left panel, the graphs are drawn with vertices colored by observed labels: dark green and red. In the right panel, the same graphs are shown with vertex colors representing refined classes. For the karate club graph, two vertices from the dark green community are refined into a new group, colored light green. For the political blogs graph, some vertices in the dark green group are refined into a new group, also colored light green, while some vertices in the red group are refined into another new group, colored orange. (Colour figure online)

Table 2 This table reports the vertex classification results for multiple-graph data with a common vertex set. All numbers are in percentile

	R-GEE	GEE	U-ASE	U-LSE	N2V
C-Elegans Ac+Ag	**33.7** ± 2.2	40.2 ± 1.7	34.7 ± 1.3	40.2 ± 1.3	51.9 ± 2.8
Wiki TE+TF	**14.6** ± 0.5	18.0 ± 0.3	20.7 ± 0.3	21.1 ± 0.3	n/a
Wiki TE+GE	**14.6** ± 0.6	17.8 ± 0.4	21.2 ± 0.3	30.2 ± 0.3	n/a
Wiki TF+GF	**15.7** ± 0.5	18.7 ± 0.3	21.1 ± 0.3	31.3 ± 0.4	n/a
Wiki GE+GF	**32.2** ± 0.8	39.2 ± 0.8	43.7 ± 0.3	50.8 ± 0.5	39.9 ± 0.7
Wiki TE+TF+GE+GF	**13.3** ± 0.5	16.1 ± 0.3	18.0 ± 0.3	27.9 ± 0.4	n/a

note that these latent communities may not always enhance the performance of downstream tasks. For example, in the karate club dataset, Fig. 4 clearly identifies two vertices that should form their own community. However, as shown in Table 1, this refinement does not improve vertex classification performance when evaluated against the original class assignments. A similar situation occurs with the political blogs dataset. Although it is visually evident that some vertices should be refined into separate groups, the refinement does not lead to better vertex classification performance. Since the classification benchmark used the original observed labels, we believe this is because the refinement does not always contribute to improved separation of those original labels.

6.3 Vertex classification on multiplex graphs

Some of the real data, specifically the C. elegans data and the Wikipedia data, come with multiple graphs of a common vertex set. The graph encoder embedding can be directly used for multiple graph inputs (Shen et al. 2024) by concatenating the embeddings, as can the refined version. LDA classifier can then be applied to measure the quality of the joint embedding via vertex classification.

For the spectral embedding, we use the unfolded spectral embedding (Gallagher et al. 2021): Given M graphs of matched common vertices, the unfolded version concatenates all adjacency matrices by rows into $\mathbf{A} \in \mathbb{R}^{n \times Mn}$, and applies SVD to yield $\mathbf{Z}^{UASE} = \mathbf{V}_d \mathbf{S}_d^{0.5} \in \mathbb{R}^{Mn \times d}$, where each $n \times d$ matrix is the embedding for the corresponding graph. We then reshape and concatenate the embedding into $\mathbb{R}^{n \times Md}$ and carry out the vertex classification using linear discriminant analysis.

For node2vec, we simply apply node2vec to each graph, concatenate their embeddings, and apply LDA. Everything else is exactly the same as in Sect. 6, and Table 2 reports the average vertex classification error and the standard deviation for different combinations of the graph data. The results are consistent with those in Table 1, where refined GEE always improves over the original GEE, and is the best performer throughout all combinations. Additionally, using multiple graphs improves over single graph results.

7 Conclusion

This paper introduces a refined graph encoder embedding method, offering both theoretical justification for its effectiveness and practical insights into the role of latent community structures in improving vertex classification. We demonstrate the advantages of the method through a combination of theoretical results, simulations, and real-world experiments.

In contrast to deep learning-based methods such as GCNs and their many variants, our approach offers unique advantages. Deep learning methods excel at numerical optimization and adapt well to various graph-based tasks by defining appropriate objective functions and utilizing gradient descent for parameter optimization. However, these methods often lack theoretical foundation, and their black-box nature provides limited interpretability regarding the graph's structure. They can also be computationally slow for large graphs. By contrast, the refined graph encoder embedding method presented in this paper not only delivers excellent performance but is also theoretically grounded and, most importantly, provides valuable insights into the latent community structures of the graph — an area where deep learning methods are inherently limited.

Looking ahead, several promising directions for future research emerge. While this study employs linear discriminant analysis for self-training, alternative approaches, including deep learning-based classifiers, could be investigated to further enhance empirical performance on real-world datasets. Additionally, the discovery of underlying latent communities opens avenues for a wide range of downstream analyses, such as improving semi-supervised learning, advancing graph recommendation systems, and addressing other graph-based tasks. Another intriguing possibility is to use the graph encoder embedding as a structured node feature initialization for graph neural networks. Recent work has demonstrated the effectiveness of randomized node feature initialization (Eliasof et al. 2023), suggesting that higher-quality feature representations generated could yield even better results. Lastly, while our study primarily focused on the discovery and utilization of latent communities for vertex classification, their practical value in domain-specific

graph data warrants further exploration. This could provide significant insights and enhance the interpretability of community structures across various real graphs.

Appendix

PseudoCode

Algorithm 1 GEE Self Training via Linear Discriminant Analysis (GEELDA)

Input: The graph adjacency matrix $\mathbf{A} \in \mathbf{R}^{n \times n}$ and a label vector $\mathbf{Y} \in \{0, 1, \ldots, K\}^n$, where 1 to K represent known labels, and 0 is a dummy category for vertices with unknown labels.

Output: A linear transformed encoder embedding $\mathbf{Z}_1 \in \mathbb{R}^{n \times K}$, self-trained new label $\mathbf{Y}_1 \in \{0, 1, \ldots, K\}^n$, and an indicator vector $\delta \in [0, 1]^n$.

function GEELDA($\mathbf{A}, \mathbf{Y}$)

$\mathbf{Z} = \text{GEE}(\mathbf{A}, \mathbf{Y})$; ▷ original one-hot graph encoder embedding

$\mathbf{Z}_1 = \text{LDA}(\mathbf{Z}, \mathbf{Y})$; ▷ transform the encoder embedding by Equation 1

$[, \mathbf{Y}_1] = \text{rowmax}(\mathbf{Z}_1)$; ▷ the maximum dimension per vertex

$ind = \text{find}(\mathbf{Y} == 0)$;

$\mathbf{Y}_1(ind) = 0$; ▷ omit vertices with unknown labels

$\delta_1 = (\mathbf{Y} \neq \mathbf{Y}_1)$;

end function

Algorithm 2 Refined Graph Encoder Embedding (R-GEE)

Input: The graph adjacency matrix $\mathbf{A} \in \mathbf{R}^{n\times n}$ and a label vector $\mathbf{Y} \in \{0, 1, \ldots, K\}^n$; number of refinement γ_K and γ_Y, set to 5 by default; stopping criterion $\epsilon \in [0, 1]$ and $\epsilon_n \in \mathbb{N}$, set to 0.3 and 5 by default.

Output: The refined graph encoder embedding $\mathbf{Z} \in \mathbb{R}^{n\times d}$, a concatenated label matrix $\mathbf{Y}$.

function R-GEE($\mathbf{A}, \mathbf{Y}, \gamma_K, \gamma_Y, \epsilon, \epsilon_n$)
  $[\mathbf{Z}_1, \mathbf{Y}_1, \delta_1] = \text{GEELDA}(\mathbf{A}, \mathbf{Y})$; $\mathbf{Y} = \mathbf{Y}_1$;
  for $k = 1, \ldots, \gamma_Y$ **do**
    $[\mathbf{Z}_2, \mathbf{Y}_2, \delta_2] = \text{GEELDA}(\mathbf{A}, \mathbf{Y}_1)$;
    if $sum(\delta_1) - max(sum(\delta_1) * \epsilon, \epsilon_n) < sum(\delta_1 \cdot \delta_2)$ **then**
      Break;
    else
      $\mathbf{Z}_1 = [\mathbf{Z}_1, \mathbf{Z}_2]$; $\mathbf{Y}_1 = \mathbf{Y}_2$; $\mathbf{Y} = [\mathbf{Y}, \mathbf{Y}_1]$;
      $\delta_1 = \delta_1 \cdot \delta_2$;
    end if
  end for
  for $k = 1, \ldots, \gamma_K$ **do**
    $[\mathbf{Z}_2, \mathbf{Y}_2, \delta_2] = \text{GEELDA}(\mathbf{A}, \mathbf{Y}_1 + \delta_1 * K)$;
    if $sum(\delta_1) - max(sum(\delta_1) * \epsilon, \epsilon_n) < sum(\delta_1 \cdot \delta_2)$ **then**
      Break;
    else
      $\mathbf{Z}_1 = [\mathbf{Z}_1, \mathbf{Z}_2]$; $\mathbf{Y}_1 = \mathbf{Y}_2$; $\mathbf{Y} = [\mathbf{Y}, \mathbf{Y}_1]$;
      $\delta_1 = \delta_1 \cdot \delta_2$;
    end if
  end for
end function

B theorem proofs

Theorem 1 *The graph encoder embedding is asymptotically normally distributed under SBM. Specifically, as n increases, for a given ith vertex of class y, it holds that*

$$\text{diag}\,(\vec{n}_K)^{0.5} \cdot (\mathbf{Z}(i,:) - \mu_y) \xrightarrow{n} \mathcal{N}(0, \Sigma_y).$$

The expectation and covariance are: $\mu_y = \mathbf{B}(y,:)$ *and* $\Sigma_y(k,k) = \mathbf{B}(y,k)(1 - \mathbf{B}(y,k))$. *Assuming* Σ_y *is the same across all* $y \in [1, K]$, *the transformation in Eq.* 1 *satisfies*

$$\mathbf{Z}_1(i,k) \xrightarrow{n} Prob(Y = k | X = \mathbf{Z}(i,:)).$$

Proof First, a necessary assumption is that as n goes to infinity, so does n_k; i.e., as the number of vertices goes to infinity, the number of vertices per class also increases to infinity. This is a standard regularity assumption in pattern recognition because, without it, the class would become trivial as n increases.

Under SBM, each dimension $k = 1, \ldots, K$ of the vertex embedding satisfies

$$\begin{aligned}\mathbf{Z}(i,k) &= \mathbf{A}(i,:)\mathbf{W}(:,k)\\ &= \frac{\sum_{j=1}^{n} I(\mathbf{Y}(j)=k)\mathbf{A}(i,j)}{n_k}\\ &= \frac{\sum_{j=1,j\neq i,\mathbf{Y}(j)=k}^{n} Bern(\mathbf{B}(y,k))}{n_k}.\end{aligned}$$

If $k = y$, the numerator is a summation of $(n_k - 1)$ i.i.d. Bernoulli random variables, since the summation includes a diagonal entry of $\mathbf{A}$, which is always 0. Otherwise, $k \neq y$ and the numerator is a summation of n_k i.i.d. Bernoulli random variables.

Checking the Lyapunov condition and applying the central limit theorem, we have

$$\sqrt{n_k}(\mathbf{Z}(i,k) - \mathbf{B}(y,k)) \xrightarrow{d} \mathcal{N}(0, \mathbf{B}(y,k)(1-\mathbf{B}(y,k))).$$

for each dimension k.

Note that $\mathbf{Z}(i,k)$ and $\mathbf{Z}(i,l)$ are always independent when $k \neq l$. This is because every vertex belongs to a unique class, so the same Bernoulli random variable never appears in another dimension. Concatenating every dimension yields that

$$Diag(\vec{n})^{0.5} \cdot (\mathbf{Z}(i,:) - \mathbf{B}(y,:)) \xrightarrow{d} \mathcal{N}(0, \Sigma_y).$$

Note that more details on the Lyapunov condition, as well as cases for other random graph models, can be found in Theorem 1 in Shen et al. (2023b).

Now, given $\mathbf{Z}(i,:)$ is normally distributed for n large, it follows immediately from classical pattern recognition (Devroye et al. 1996) that under the normality assumption and a common variance across all k, the linear transformation in Eq. 1 estimates the conditional probability. This is because the LDA transformation directly estimates $Prob(Y|X)$ when $X|Y$ is normally distributed. Specifically,

$$\mathbf{Z}_1(i,k) = \mathbf{Z}(i,k)\Sigma^{+}\mu_k - ((\mu_k'\Sigma^{+}\mu_k) - \log(n_K/n))$$

is the exact LDA transformation for each class $k = 1, \ldots, K$. Writing it into a matrix expression for all k leads to Eq. 1.

Theorem 2 *Suppose the graph is distributed as the stochastic block model with block probability* $\mathbf{B} \in \mathbb{R}^{K\times K}$ *and observed label vector* $\mathbf{Y} \in [1, \ldots, K]$. *Then for any two vertices* i, j, *the encoder embedding* $\mathbf{Z}$ *using observed labels satisfies*:

$$\|\mathbf{Z}(i,:) - \mathbf{Z}(j,:)\|_2 - \|\mathbf{B}(\mathbf{Y}(i),:) - \mathbf{B}(\mathbf{Y}(j),:)\|_2 \xrightarrow{n} 0$$

Suppose the same graph can be viewed as a realization of a latent stochastic block model with $\mathbf{B}_0 \in \mathbb{R}^{K_0\times K_0}$ *and a latent label vector* $\mathbf{Y}_0 \in [1, \ldots, K_0]$ *where* $K_0 > K$. *The for the same two vertices* i, j, *the resulting encoder embedding* $\mathbf{Z}_0$ *using the latent labels satisfies*:

$$\|\mathbf{Z}_0(i,:) - \mathbf{Z}_0(j,:)\|_2 - \|\mathbf{B}_0(\mathbf{Y}_0(i),:) - \mathbf{B}_0(\mathbf{Y}_0(j),:)\|_2 \xrightarrow{n} 0$$

Proof From Theorem 1, it is immediate that the encoder embedding satisfies the law of large numbers, such that

$$\|\mathbf{Z}(i,:) - \mathbf{B}(\mathbf{Y}(i),:)\|_2 \xrightarrow{n} 0.$$

It follows that

$$\begin{aligned}&\|\mathbf{Z}(i,:) - \mathbf{Z}(j,:)\|_2 - \|(\mathbf{B}(\mathbf{Y}(i),:) - \mathbf{B}(\mathbf{Y}(j),:))\|_2\\ \leq &\|\mathbf{Z}(i,:) - \mathbf{Z}(j,:) - (\mathbf{B}(\mathbf{Y}(i),:) - \mathbf{B}(\mathbf{Y}(j),:))\|_2\\ = &\|(\mathbf{Z}(i,:) - \mathbf{B}(\mathbf{Y}(i),:)) - (\mathbf{Z}(j,:) - \mathbf{B}(\mathbf{Y}(j),:))\|_2\\ \leq &\|\mathbf{Z}(i,:) - \mathbf{B}(\mathbf{Y}(i),:)\|_2 + \|\mathbf{Z}(i,:) - \mathbf{B}(\mathbf{Y}(j),:)\|_2\\ \to &0.\end{aligned}$$

Since the graph encoder embedding is fully dependent on the given labels, when the latent labels are used, we also have

$$\|\mathbf{Z}_0(i,:) - \mathbf{B}(\mathbf{Y}_0(i),:)\|_2 \xrightarrow{n} 0,$$

so the same derivation and convergence apply to the encoder embedding using latent labels as well.

C additional experiments

Generally, we observe that varying parameters can lead to differences in results, but for most simulations and real datasets, the impact is minimal. Here, we examine parameter sensitivity, specifically for ϵ and ϵ_n, under four settings:

- Setting 1: $\epsilon = 0.6$ and $\epsilon_n = 50$;
- Setting 2: $\epsilon = 0.4$ and $\epsilon_n = 20$;
- Setting 3: $\epsilon = 0.2$ and $\epsilon_n = 5$;
- Setting 4: $\epsilon = 0.02$ and $\epsilon_n = 2$;

The refinement becomes progressively more aggressive across these settings. For example, consider Eq. 2 with $\sum_{i=1}^{n} \delta_1(i) = 100$ (indicating 100 mismatched vertices based on the LDA self-learned labels versus the previous labels) and $\sum_{i=1}^{n} \delta_1(i) \cdot \delta_2(i) = 70$ (where 70 mismatches remain after refinement). In this case, the difference is 30. Under Settings 1 and 2, refinement would stop, as 30 is smaller than $\max(\sum_{i=1}^{n} \delta_1(i) * \epsilon, \epsilon_n)$. However, under Settings 3 and 4, refinement would continue, as the difference exceeds the thresholds for these more aggressive settings.

Figure 5 illustrates the vertex classification error and the precision of recovering latent communities under the four parameter settings for simulated graph 2 in Fig. 3. In the left panel of Fig. 5, we see that precision is only slightly affected by parameter changes, decreasing slightly as refinement becomes more aggressive. This indicates that a few

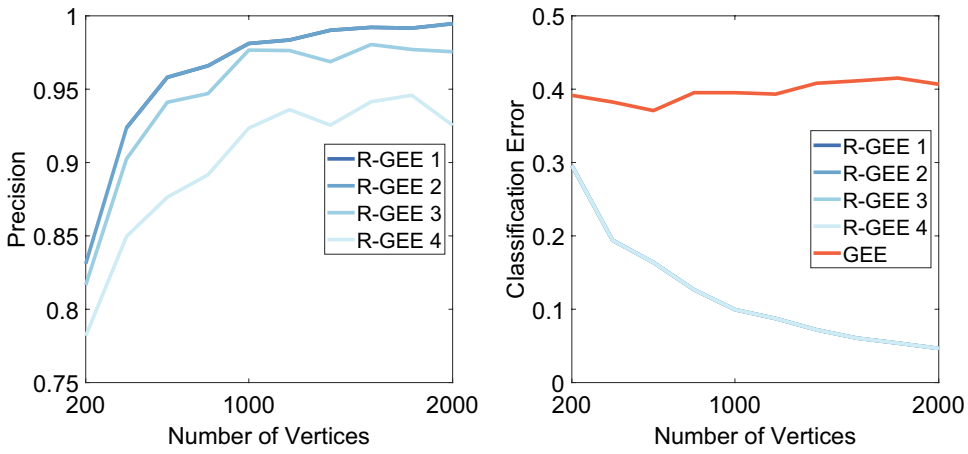


Fig. 5 The left panel displays the precision of the refined GEE in recovering latent communities under four different parameter settings. The right panel shows the vertex classification error

vertices are incorrectly reassigned to new communities due to overly aggressive refinement. However, as shown in the right panel, this has a negligible effect on vertex classification error, where all four settings yield nearly identical results.

Acknowledgements This work was supported by the National Science Foundation DMS-2113099, and by funding from Microsoft Research.

Data availibility The datasets generated and/or analysed during the current study are available in the Github GraphEmd repository, [https://github.com/cshen6/GraphEmd].

Declarations

Conflict of interest The authors declare that they have no Conflict of interest.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

Adamic L, Glance N (2005) The political blogosphere and the 2004 us election: Divided they blog. In: Proceedings of the 3rd International Workshop on Link Discovery. ACM Press, New York

Airoldi EM, Blei DM, Fienberg SE et al (2008) Mixed membership stochastic blockmodels. J Mach Learn Res 9:1981–2014

Arroyo J, Athreya A, Cape J et al (2021) Inference for multiple heterogeneous networks with a common invariant subspace. J Mach Learn Res 22(142):1–49

Athreya A, Fishkind DE, Tang M et al (2018) Statistical inference on random dot product graphs: a survey. J Mach Learn Res 18(226):1–92

Barabasi AL, Oltvai ZN (2004) Network biology: understanding the cell's functional organization. Nature Rev Genet 5(2):101–13

Belkin M, Niyogi P (2003) Laplacian eigenmaps for dimensionality reduction and data representation. Neural Comput 15(6):1373–1396

Boccaletti S, Latora V, Moreno Y et al (2006) Complex networks: structure and dynamics. Phys Rep 424(4–5):175–308

Chung J, Pedigo BD, Bridgeford EW, Varjavand BK, Helm HS, Vogelstein JT (2019) Graspy: graph statistics in python. J Mach Learn Res 20(158):1–7

Dettmers T, Minervini P, Stenetorp P, et al (2018) Convolutional 2d knowledge graph embeddings. In: Proceedings of the AAAI Conference on Artificial Intelligence

Devroye L, Gyorfi L, Lugosi G (1996) A Probabilistic Theory of Pattern Recognition. Springer

Eliasof M, Frasca F, Bevilacqua B, et al (2023) Graph positional encoding via random feature propagation. In: Proceedings of the 40th International Conference on Machine Learning

Gallagher I, Jones A, Bertiger A et al (2023) Spectral clustering of weighted graphs. J Am Stat Assoc 119(547):1923–32

Gallagher I, Jones A, Rubin-Delanchy P (2021) Spectral embedding for dynamic networks with stability guarantees. Adv Neural Inf Proc Syst 34:10158–70

Girvan M, Newman MEJ (2002) Community structure in social and biological networks. Proc Nat Acad Sci 99(12):7821–7826

Grover A, Leskovec J (2016) node2vec: Scalable feature learning for networks. In: Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining

He X, Deng K, Wang X, et al (2020) LightGCN: Simplifying and powering graph convolution network for recommendation. In: Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval

Holland P, Laskey K, Leinhardt S (1983) Stochastic blockmodels: first steps. Social Networks 5(2):109–137

Hu W, Fey M, Zitnik M, Dong Y, Ren H, Liu B, Catasta M, Leskovec J (2020) Open graph benchmark: datasets for machine learning on graphs. Adv Neural Inf Proc Syst 33:22118–33

Karrer B, Newman MEJ (2011) Stochastic blockmodels and community structure in networks. Phys Rev E 83:016107

Kipf TN, Welling M (2017) Semi-supervised classification with graph convolutional networks. In: International Conference on Learning Representations

Leskovec J, Krevl A (2014) SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data

Mehta K, Goldin RF, Marchette D et al (2021) Neuronal classification from network connectivity via adjacency spectral embedding. Netw Neurosci 5(3):689–710

Mu C, Mele A, Hao L et al (2022) On spectral algorithms for community detection in stochastic blockmodel graphs with vertex covariates. IEEE Transa Netw Sci Eng 9(5):3373–3384

Newman MEJ (2003) The structure and function of complex networks. SIAM Rev 45(2):167–256

Ng A, Jordan M, Weiss Y (2001) On spectral clustering: analysis and an algorithm. In: Proceedings of the 14th International Conference on Neural Information Processing Systems

Pavlovic D, Vertes P, Bullmore E et al (2014) Stochastic blockmodeling of the modules and core of the caenorhabditis elegans connectome. PLoS ONE 9(9):e97584

Priebe C, Parker Y, Vogelstein J et al (2019) On a 'two truths' phenomenon in spectral graph clustering. Proc Natl Acad Sci 116(13):5995–5600

Rohe K, Chatterjee S, Yu B (2011) Spectral clustering and the high-dimensional stochastic blockmodel. Ann Stat 39(4):1878–1915

Rossi RA, Ahmed NK (2015) The network data repository with interactive graph analytics and visualization. In: AAAI, https://networkrepository.com

Rozemberczki B, Sarkar R (2020) Characteristic functions on graphs: Birds of a feather, from statistical descriptors to parametric models. In: Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20), ACM

Rubin-Delanchy P, Cape J, Tang M et al (2022) A statistical interpretation of spectral embedding: the generalised random dot product graph. J R Stat Soc Ser B Stat Methodol 84(4):1446–1473

Shen C (2024) Encoder embedding for general graph and node classification. Appl Netw Sci 9:66

Shen C, Larson J, Trinh H et al (2024) Discovering communication pattern shifts in large-scale labeled networks using encoder embedding and vertex dynamics. IEEE Transa Netw Sci Eng 11(2):2100–2109

Shen C, Priebe CE, Larson J et al (2024) Synergistic graph fusion via encoder embedding. Inf Sci 678:120912

Shen C, Sun M, Tang M et al (2014) Generalized canonical correlation analysis for classification. J Multivar Anal 130:310–322

Shen C, Wang Q, Priebe CE (2023) One-hot graph encoder embedding. IEEE Trans Pattern Anal Mach Intell 45(6):7933–7938

Shen C (2024b) Fast and scalable multi-kernel encoder classifier. In: Proceedings of the Future Technologies Conference (FTC) 2024, Springer Nature

Shen C, Park Y, Priebe CE (2023a) Graph encoder ensemble for simultaneous vertex embedding and community detection. In: 2023 2nd International Conference on Algorithms, Data Mining, and Information Technology. ACM

Shen C, Arroyo J, Xiong J, et al (2024a) Community correlations and testing independence between binary graphs. arXiv preprint arXiv:1906.03661

Shen C, Dong Y, Priebe CE, et al (2025a) Principal graph encoder embedding and principal community detection. arXiv preprint arXiv:2501.14939

Shen C, Edge D, Larson J, et al (2025b) Explaining categorical feature interactions using graph covariance and LLMs. arXiv preprint arXiv:2501.14932

Snijders T, Nowicki K (1997) Estimation and prediction for stochastic blockmodels for graphs with latent block structure. J Classif 14(1):75–100

Sussman D, Tang M, Fishkind D et al (2012) A consistent adjacency spectral embedding for stochastic blockmodel graphs. J Am Stat Assoc 107(499):1119–1128

Sussman D, Tang M, Priebe C (2014) Consistent latent position estimation and vertex classification for random dot product graphs. IEEE Trans Pattern Anal Mach Intell 36(1):48–57

Tang M, Sussman DL, Priebe CE (2013) Universally consistent vertex classification for latent positions graphs. Ann Stat 41(3):1406–1430

Tang J, Qu M, Wang M, et al (2015) Line: Large-scale information network embedding. In: Proceedings of the 24th International Conference on World Wide Web (WWW '15)

Ugander J, Karrer B, Backstrom L, et al (2011) The anatomy of the facebook social graph. arXiv preprint arXiv:1111.4503

Varshney L, Chen B, Paniagua E et al (2011) Structural properties of the caenorhabditis elegans neuronal network. PLoS Comput Biol 7(2):e1001066

Wang H, Leskovec J (2022) Combining graph convolutional neural networks and label propagation. ACM Trans Inf Syst 40(4):1–27

Wu Z, Pan S, Chen F et al (2019) A comprehensive survey on graph neural networks. IEEE Transa Neural Netw Learn Syst 32:4–24

Xu K, Hu W, Leskovec J, et al (2019) How powerful are graph neural networks? In: Proc. ICLR

Yan X, Jensen JE, Krzakala F et al (2012) Model selection for degree-corrected block models. J Stat Mech: Theory Exp 05:P05007

Yao T, Wang R, Wang J et al (2024) Efficient supervised graph embedding hashing for large-scale cross-media retrieval. Pattern Recogn 145:109934

Yin H, Benson AR, Leskovec J, et al (2017) Local higher-order graph clustering. In: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining

Zachary W (1977) An information flow model for conflict and fission in small groups. J Anthropol Res 33:452–473

Zhao Y, Levina E, Zhu J (2012) Consistency of community detection in networks under degree-corrected stochastic block models. Ann Stat 40(4):2266–2292

Zhao Z, Wang R, Wang Z et al (2024) Graph joint representation clustering via penalized graph contrastive learning. IEEE Trans Neural Netw Learn Syst 35(12):17650–17661

Zheng R, Lyzinski V, Priebe CE et al (2022) Vertex nomination between graphs via spectral embedding and quadratic programming. J Comput Graph Stat 31(4):1254–1268

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.