Clustering by Deep Latent Position Model with Graph Convolutional Network

Dingge Liang, Marco Corneli, Charles Bouveyron, Pierre Latouche

Clustering by Deep Latent Position Model with Graph Convolutional Network

Dingge Liang[1*], Marco Corneli[1,2], Charles Bouveyron[1] and Pierre Latouche[3]

[1*]Université Côte d'Azur, INRIA, CNRS, Laboratoire J.A.Dieudonné, Maasai team, Nice, France.
[2]Center of modeling, Simulation and Interactions (MSI), Nice, France.
[3]Université Clermont Auvergne, CNRS, LMBP UMR 6620, Aubière, France.

Abstract

With the significant increase of interactions between individuals through numeric means, clustering of nodes in graphs has become a fundamental approach for analyzing large and complex networks. In this work, we propose the deep latent position model (DeepLPM), an end-to-end generative clustering approach which combines the widely used latent position model (LPM) for network analysis with a graph convolutional network (GCN) encoding strategy. Moreover, an original estimation algorithm is introduced to integrate the explicit optimization of the posterior clustering probabilities via variational inference and the implicit optimization using stochastic gradient descent for graph reconstruction. Numerical experiments on simulated scenarios highlight the ability of DeepLPM to self-penalize the evidence lower bound for selecting the number of clusters, demonstrating its clustering capabilities compared to state-of-the-art methods. Finally, DeepLPM is further applied to an ecclesiastical network in Merovingian Gaul and to a citation network Cora to illustrate the practical interest in exploring large and complex real-world networks.

Keywords: network analysis; clustering; unsupervised deep learning; graph neural networks; latent position models

1 Introduction and related work

Networks are employed in a wide range of applications, from social media and email communications to protein-protein interactions, because they are simple structures yet are capable of modeling complex systems. In this context, node clustering is a key branch of clustering which attempts to partition the nodes of the graph into different groups to extract patterns summarizing the data.

A long series of statistical methods [1, 2] have been developed to discover the underlying communities in networks by learning the latent features of graph-structured data. More recently, deep learning based models have emerged as a promising approach for analyzing large-scale networks and they have shown their abilities for representation learning purpose on data with complex structures [3, 4]. We hereafter split the existing approaches for node clustering in networks into two categories and briefly review them.

Statistical models for clustering. On the one hand, the stochastic block model [SBM, 5, 6] is widely used to detect communities or more general clusters of nodes [7]. It assumes that nodes are spread into different latent clusters and that the connection probability between each pair of nodes depends exclusively on their group memberships. Based on SBM, many extensions looking for overlapping clusters have been proposed. For instance, the mixed-membership stochastic blockmodel [MMSBM, 8] introduces a mixing weight vector π_i drawn from the Dirichlet distribution for each node, while the overlapping stochastic blockmodel [OSBM, 9] assumes each node to be characterized by a binary latent vector sampled from a product of Bernoulli distributions, allowing each node to belong to multiple clusters. Other variants consider the processing of valued graphs, such as networks with discrete edges [10], categorical edges [11] or text edges [12]. Moreover, some extensions [13–15] allow to deal with time-evolving networks through dynamic network modeling. On the other hand, a different approach to model network data relies on the potential position of nodes. Originally proposed by [16], the latent position model (LPM) supposes that each node has an unknown position in a latent space and that the probability of a specific link between two nodes is modeled by some function of their positions. Afterwards, the latent position cluster model [LPCM, 17] was introduced to incorporate a clustering structure into LPM by considering that the latent position of each node is drawn from a Gaussian mixture model (GMM). Further developments of LPMs exist and the reader is referred to [18] for an extensive review. Nevertheless, these statistical models face a challenging inference procedure that primarily relies on MCMC or variational approximations and do not scale easily to large and complex networks. A more general overview of statistical models for clustering network data can be explored in [19, Chapter 10].

Deep learning models for clustering. From another aspect, deep neural networks (DNNs) based techniques have recently shown to be effective for feature representation learning and have been actively explored in clustering [20, 21]. Two widely used approaches are deep embedded clustering [DEC,

22] and variational deep embedding [VaDE, 23]. VaDE models the data generative procedure by combining GMM prior distributions with variational auto-encoding [VAE, 24], while DEC learns a mapping function and imposes a soft assignment constraint on the latent features. However, neither VaDE nor DEC are designed for graph-structured data. For the purpose of performing node clustering on networks, new models were introduced based on graph neural networks [GNNs, 25]. In this line of methods, the variational graph auto-encoder [VGAE, 26] adopted a graph convolutional network [GCN, 27] encoder to produce nodes embeddings in the latent space and a simple inner product decoder for graph reconstruction. As an extension, MGAE [28] proposed a marginalization process that adds random noise to the content information of nodes and stacked several single layer graph auto-encoders to reconstruct the node features matrix. By introducing adversarial learning into the generation process, ARVGA [29] enforced the latent representation to match a prior distribution. More recently, AM-GCN [30] proposed a graph convolution both accounting for the network topology and the node features. Moreover, FAGCN [31] introduced an additional high-frequency filter to collaborate with low-frequency filters in conventional GNNs, in order to handle networks with heterogeneous structures. All of the aforementioned and other existing approaches [32–34] adopt a two-step clustering procedure, simply relying on *external* clustering algorithms (e.g. K-means) to group the embedded nodes, independently from the generative model.

Main contributions. In order to overcome the limitations of the methods listed above, while exploring their benefits, we introduce a new deep latent position model (DeepLPM) for network data, allowing to simultaneously learn node representations and obtain node partitions. By combining a GCN encoder with a LPM-based decoder, our model aims at capturing the best of both worlds described so far: it is a flexible representation learning tool based on the deep learning architecture, yet comprehensive and interpretable thanks to the statistical model considered. The DeepLPM we propose here, has the following key-features:

- a LPM-based decoder models the probability of interactions between a pair of nodes as a function of the distance between them in the latent space. Compared with a standard inner-product-based decoder, this choice better preserves the network topology in different scenarios (see Figure 1);
- DeepLPM performs an *end-to-end* clustering of the nodes by estimating the posterior probabilities for cluster memberships. Thus, it can automatically assign each node to its group without using any additional algorithms;
- an original estimation algorithm is designed to integrate the expectation maximization of the posterior clustering probabilities (explicit) and the stochastic gradient descent optimization for graph reconstruction (implicit);
- by combining the substantial representations learned by GCN with the position information, we point out the self-penalizing capability of DeepLPM

in selecting the number of clusters, and demonstrate its effectiveness in performing different clustering tasks.

Organization of the paper. In Section 2, we introduce the generative model behind DeepLPM. The variational inference and the original optimization algorithm are discussed in Section 3. Numerical experiments are provided in Section 4, highlighting the main features of our proposed approach and validating its self-penalization ability in model selection. An application to a real-world network coming from the Medieval history of Europe is presented in Section 5 and an analysis of the citation network Cora is described in Section 6. Section 7 finally concludes with a summary of this work.

2 Deep latent position model

In this section, the DeepLPM for end-to-end node clustering and network representation is first introduced.

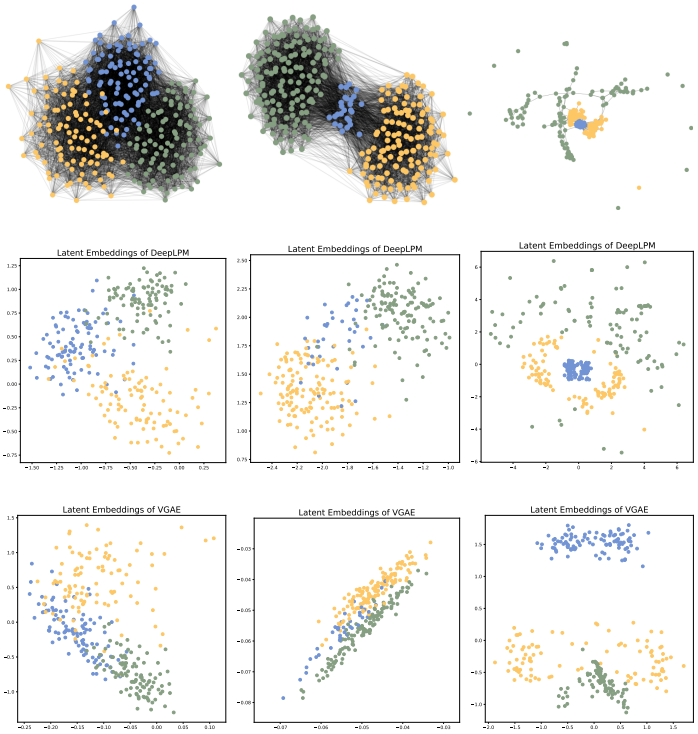

Fig. 1 Simulated networks and learned embeddings in three scenarios. From top to bottom: the original simulated graphs, the latent embeddings learned by DeepLPM and the embeddings learned by VGAE. To facilitate the visualization, the latent dimension is set to 2.

Table 1 List of all model parameters

Notation	Description
A	Adjacency matrix in $[0,1]^{N\times N}$
Y	Edge feature matrix
N	Number of nodes
K	Number of clusters
P	Latent space dimension
D	Edge features dimension
π	Prior cluster probability vector
C	Cluster memberships
Z	Latent node embeddings in $\mathbb{R}^P$
$f_{\alpha,\beta}$	Decoder parametrized by α, β
g_ϕ	GCN encoder parametrized by ϕ
γ_{ik}	Posterior probability that node i is in cluster k

2.1 Notations

In this work, networks are modeled as undirected, unweighted graphs $G = (V;\ E)$ with $N = |V|$ nodes. We introduce an $N \times N$ adjacency matrix A, where $A_{ij} = 1$ if there is a link between node i and node j, 0 otherwise. The set of edges E can be associated with an additional covariate information, collected into matrix $Y \in \mathbb{R}^{|E|\times D}$. The generic entry of Y, denoted y_{ij}, is a D-dimensional feature associated with the edge connecting i to j. For instance, y_{ij} could encode the text that author i sends to author j in a communication network. We aim at learning well-represented, latent, node embeddings Z in a lower dimension P and to partition the nodes into K clusters. Necessary notations are summarized in Table 1.

2.2 Generative model

As in LPM [16], we assume that each node $i = \{1, \cdots, N\}$ has an unknown position $z_i \in \mathbb{R}^P$ in a latent space and that the edges in the network are sampled independently given these positions. Moreover, the probability of a link between two individuals is modeled as a function of the distance between the two nodes, in the latent space. The generative process is as follows.

First, each node is assigned to a cluster via a random variable c_i encoding its cluster membership

$$c_i \overset{\text{i.i.d}}{\sim} \text{Multinomial}(1,\pi), \quad \text{with} \quad \pi \in [0,1]^K,\ \sum_{k=1}^{K} \pi_k = 1. \tag{1}$$

Then, conditionally to its cluster membership, a latent embedding vector z_i is generated

$$z_i | c_{ik} = 1 \sim \mathcal{N}(\mu_k, \sigma_k^2 I_P), \quad \text{with} \quad \sigma_k^2 \in \mathbb{R}^{+*}, \tag{2}$$

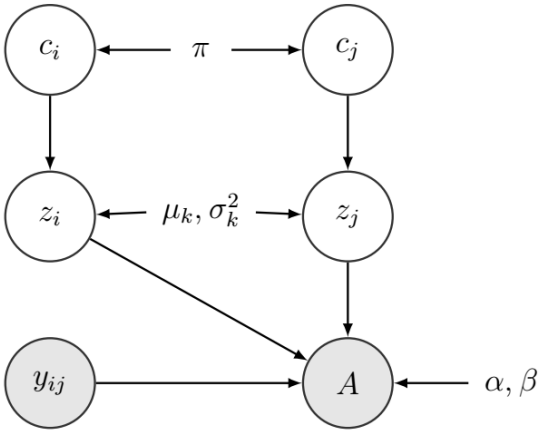

Fig. 2 Graphical representation of DeepLPM (variational parameters are not included).

independently for each node, where μ_k and σ_k^2 denote the mean and variance of each cluster, I_P denotes an identity matrix in $\mathbb{R}^P$.

Finally, the probability of a connection between nodes i and j, as represented by adjacency matrix entry A_{ij}, is modeled through a Bernoulli random variable related to the distance between the corresponding latent positions

$$A_{ij}|z_i, z_j \sim \text{Bernoulli}(f_{\alpha,\beta}(z_i, z_j)), \tag{3}$$

with

$$f_{\alpha,\beta}(z_i, z_j) = \sigma(\alpha + \beta^\top y_{ij} - \|z_i - z_j\|^2), \tag{4}$$

where $f_{\alpha,\beta}$ can be seen as a *decoding*, one-layer, neural network parametrized by α and β. Moreover, σ is the logistic sigmoid function and y_{ij} is the covariate of the edge connecting i with j. A graphical representation of the generative model described so far can be seen in Figure 2.

2.3 Links with related models

At this point, DeepLPM can be linked with the following models:

- In LPCM, a specific prior distribution for the parameters $\beta = (\beta_0^\top, \beta_1)^\top$ is introduced and the estimation is conducted using MCMC sampling. Conversely in DeepLPM, we introduce a decoding neural network $f_{\alpha,\beta}$, where the two parameters α and β are automatically optimized though stochastic gradient descent.
- Both VGAE and DeepLPM rely on the VAE architecture. However, instead of using a simple inner-product decoder as in VGAE, DeepLPM involves a latent position-based $f_{\alpha,\beta}$ decoding strategy and integrates the cluster memberships to achieve an end-to-end clustering.
- Both VaDE and DeepLPM model the data generative procedure with a Gaussian Mixture Model and a deep neural network, whereas in VaDE, both

the decoder and the encoder are convolutional neural networks, limiting the model to image data. The decoding network $f_{\alpha,\beta}$ in DeepLPM is based on latent positions, while the encoder g_ϕ is a two-layer GCN that allows to model graph-structured data.

3 Model inference

This section details the variational auto-encoding inference procedure and proposes an original estimation method which combines the explicit optimization of the posterior clustering probabilities and the implicit optimization of the neural network parameters.

3.1 Variational auto-encoding inference

Before getting into the details of the inference, we first denote by $\Theta = \{\pi, \mu_k, \sigma_k^2, \alpha, \beta\}$ the set of the model parameters introduced so far. A natural procedure would consist in maximizing the integrated log-likelihood of the observed data A with respect to Θ (and, possibly, Y, which is omitted to keep the notation uncluttered)

$$\log p(A|\Theta) = \log \int_Z \sum_C p(A, Z, C|\Theta) dZ. \tag{5}$$

Unfortunately, Eq. (5) is not tractable and we have to rely on a variational approach to approximate it

$$\log p(A|\Theta) = \mathcal{L}(q(Z, C);\ \Theta) + D_{KL}(q(Z, C)|p(Z, C|A, \Theta)), \tag{6}$$

where D_{KL} denotes the Kullback-Leibler divergence between the true and approximate posterior distributions of (Z, C) given the data and model parameters. Then, in order to deal with a tractable family of distributions, $q(Z, C)$ is assumed to fully factorize (*mean-field* assumption)

$$q(Z, C) = q(Z)q(C) = \prod_{i=1}^{N} q(z_i)q(c_i). \tag{7}$$

Moreover, to benefit from the representational learning capabilities of GCN, we assume

$$q(z_i) = \mathcal{N}(z_i;\ \tilde{\mu}_\phi(\overline{A})_i, \tilde{\sigma}^2_\phi(\overline{A})_i I_P), \tag{8}$$

where $\tilde{\mu}_\phi(\cdot) : \mathbb{R}^{N\times N} \mapsto \mathbb{R}^{N\times P}$ (respectively $\tilde{\sigma}^2_\phi(\cdot) : \mathbb{R}^{N\times N} \mapsto \mathbb{R}^{+^N}$) is the function mapping the normalized adjacency matrix $\overline{A} = \hat{D}^{-\frac{1}{2}}(A + I_N)\hat{D}^{-\frac{1}{2}}$ (we denote the degree matrix as $\hat{D}$ here to distinguish it from the edge feature dimension D) into the matrix of the variational means (vector of the standard deviations), as in [27]. In the above equation, $\tilde{\mu}_\phi(\overline{A})_i$ denotes the i-th row

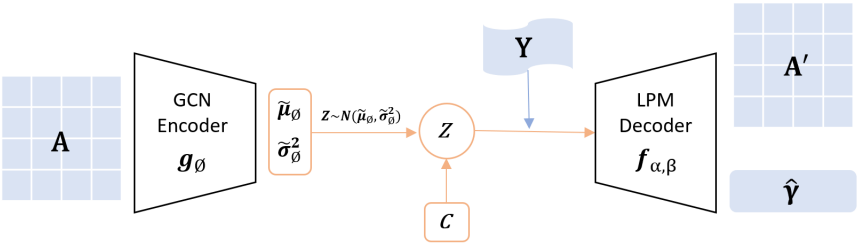

Fig. 3 A deep-learning-like model view of DeepLPM.

of $\tilde{\mu}_\phi(\overline{A})$, corresponding to the variational mean for the latent position z_i (similarly for $\tilde{\sigma}^2_\phi(\overline{A})_i$). The functions $\tilde{\mu}_\phi(\cdot)$ and $\tilde{\sigma}^2_\phi(\cdot)$ are parametrized by the GCN *encoder* g_ϕ.

Finally, a standard assumption is made for the variational clustering probabilities

$$q(C) = \prod_{i=1}^{N} \mathcal{M}(c_i;\ 1, \gamma_i), \tag{9}$$

where γ_{ik} represents the variational probability that node i is in cluster k, with $\sum_{k=1}^{K} \gamma_{ik} = 1, \forall k = 1, \cdots, K$.

Model architecture. The variational structure of DeepLPM is shown in Figure 3. Within the framework of VAE, first the graph adjacency matrix A is taken as the model input and normalized; then, through the two-layer GCN encoder, we obtain the mean and variance of each node; next, by minimizing the Kullback-Leibler divergence between the variational and the posterior distributions, we get the learned latent representations; finally, through the LPM-based decoder, we can reconstruct the matrix A' and obtain the cluster probability matrix $\hat{\gamma}$.

3.2 Optimization

In this part, we focus on maximizing the evidence lower bound (ELBO)

$$\mathcal{L}(A|\Theta) = \int_Z \sum_C q(Z, C) \log \frac{p(A, Z, C|\Theta)}{q(Z, C)} dZ \tag{10}$$

with respect to the model parameters Θ and the variational parameters ϕ. Thanks to Equations (7)-(8)-(9), Eq. (10) can be further developed as

$$\mathcal{L} = \int_Z \sum_C q(Z, C) \log \frac{p(A|Z, \alpha, \beta) p(Z|C, \mu_k, \sigma_k^2) p(C|\pi)}{q(Z, C)} dZ$$

$$
\begin{aligned}
&= \mathbb{E}\left[\log p(A|Z,\alpha,\beta)\right] + \mathbb{E}\left[\log p(Z|C,\mu_k,\sigma_k^2\right] + \mathbb{E}\left[\log p(C|\pi)\right] \\
&\quad - \mathbb{E}\left[\log q(Z|A)\right] - \mathbb{E}\left[\log q(C)\right] \\
&= \mathbb{E}\left[\log p(A|Z,\alpha,\beta)\right] + \mathbb{E}\left[\log \frac{p(Z|C,\mu_k,\sigma_k^2)}{q(Z)}\right] + \mathbb{E}\left[\log \frac{p(C|\pi)}{q(C)}\right] \\
&= \mathbb{E}\left[\sum_{i\neq j} A_{ij}\log\eta_{ij} + (1-A_{ij})\log(1-\eta_{ij})\right] \\
&\quad - \sum_{i=1}^{N}\sum_{k=1}^{K}\gamma_{ik} D_{KL}\left(\mathcal{N}(\tilde{\mu}_\phi(\overline{A})_i, \tilde{\sigma}_\phi^2(\overline{A})_i I_P)\mathcal{N}(\mu_k,\sigma_k^2 I_P)\right) \\
&\qquad + \sum_{i=1}^{N}\sum_{k=1}^{K}\gamma_{ik}\log(\frac{\pi_k}{\gamma_{ik}}),
\end{aligned}
$$

where $\eta_{ij} = \sigma(\alpha + \beta^T y_{ij} - \|z_i - z_j\|^2)$, $D_{KL}(\cdot)$ denotes the KL divergence and the expectation is taken with respect to the variational probability $q(\cdot)$.

Explicit optimization. On the one hand, an *explicit* optimization of the ELBO with respect to the parameters $\gamma_{ik}, \pi_k, \mu_k$ and σ_k can be performed to obtain the following updates:

$$
\hat{\gamma}_{ik} = \frac{\pi_k \mathrm{e}^{-D_{KL}^{ik}}}{\sum_{l=1}^{K}\pi_l \mathrm{e}^{-D_{KL}^{il}}}, \tag{11}
$$

where $D_{KL}^{ik} = \frac{1}{2}\left\{\log\frac{(\sigma_k^2)^P}{(\tilde{\sigma}_\phi^2(\overline{A})_i)^P} - P + \frac{\tilde{\sigma}_\phi^2(\overline{A})_i}{\sigma_k^2} + \frac{1}{\sigma_k^2}\mu_k - \tilde{\mu}_\phi(\overline{A})_i{}^2\right\}$.

Then

$$
\hat{\pi}_k = \sum_{i=1}^{N}\gamma_{ik}/N, \tag{12}
$$

$$
\hat{\mu}_k = \sum_{i=1}^{N}\tilde{\mu}_\phi(\overline{A})_i\gamma_{ik}/\sum_{i=1}^{N}\gamma_{ik}, \tag{13}
$$

and

$$
\hat{\sigma}_k^2 = \frac{\sum_{i=1}^{N}\gamma_{ik}(\sigma_\phi^2(\overline{A})_i + \mu_k - \tilde{\mu}_\phi(\overline{A})_i{}^2)}{P\sum_{i=1}^{N}\gamma_{ik}}. \tag{14}
$$

Detailed derivations are given in the appendix.

Implicit optimization. On the other hand, the *implicit* optimization of the encoder parameter ϕ and decoder parameters α, β is performed via stochastic gradient descent. In this work, it is implemented using the Adam optimizer [35].

Algorithm. In the estimation process, we first conduct a pre-training step to avoid the model getting stuck in a local minima or a saddle point at the beginning of training. Then, the initial weights and biases after pre-training are saved for use in the training phase. Once we obtain the mean $\tilde{\mu}_\phi(\overline{A})_i$ and variance $\tilde{\sigma}^2_\phi(\overline{A})_i$ of each node, we use them to update the cluster information γ_{ik} by minimizing the KL divergence between the variational and the posterior distributions of each node. Next, we adjust the mixture component π_k, mean μ_k and variance σ^2_k for each cluster according to the previous steps. Finally, the total loss is computed and the parameters of the encoder/decoder are optimized via stochastic gradient descent. More details are reported in Algorithm 3.2.

3.3 Model selection

The ELBO introduced in the previous section allows the estimation of the posterior law of (Z, C) for a fixed number of clusters K. If we vary this parameter, the model becomes completely different. Therefore, choosing appropriate values for K is a model selection task.

We emphasize that the self-regularization property of VAEs has already been observed in a number of studies [36, 37]. In the following Section 4.3, we conduct several experiments to show that DeepLPM does indeed benefit in practice from this property and that it induces a penalization on the ELBO, thus allowing to select the number K of clusters.

Algorithm 1 Estimation of DeepLPM

Input: adjacency matrix A, edge features Y
Output: reconstructed graph A', cluster probability matrix $\hat{\gamma}$
$\boldsymbol{\gamma}_{\text{init}} = \textit{pretrain}(A, 50 \text{ epochs})$ ▷ pre-training to initialize cluster parameters
while $\mathcal{L}$ increases **do**
  $\tilde{\mu}_\phi, \log \tilde{\sigma}_\phi = g_\phi(\overline{A})$
  $Z = \tilde{\mu}_\phi + \epsilon \tilde{\sigma}_\phi, \quad$ where $\epsilon \sim \mathcal{N}(0, I_P)$ ▷ reparameterization trick
  explicit optimization:
  update $\hat{\gamma}_{ik}$ by Equation (11)
  update $\hat{\pi}_k, \hat{\mu}_k, \hat{\sigma}^2_k$ by Equations (12)-(13)-(14)
  calculate the training loss (negative ELBO) $-\mathcal{L}$
  implicit optimization:
  update encoder parameter ϕ and decoder parameters α, β
end while

4 Numerical experiments

This section aims at emphasizing the effectiveness of this work on three synthetic datasets and at proving the validity of the estimation algorithm proposed in the previous Section 3.2.

4.1 Simulation setup

In order to simplify the characterization and to facilitate the reproducibility of the experiments, we designed three types of synthetic networks based on the generative models LPCM, SBM and from circle data, respectively:

- scenario A simulates data according to LPCM [17]. 3 communities are considered and edges are generated based on the distance between each node position in dimension $P = 2$. We set a parameter $\delta \in [0.2, 0.95]$ to represent the rate of proximity between the clusters where a larger δ means that the three clusters are better separated. In this experiment, we set the mean of each cluster to
$$\begin{cases} \mu_1 = [0, 0] \\ \mu_2 = [1.5 * \delta, 1.5 * \delta] \\ \mu_3 = [-1.5 * \delta, 1.5 * \delta] \end{cases}$$
- scenario B simulates data according to SBM [6]. It consists of one cluster with large probability of external connectivity and two communities that have a higher tendency to link within subset than across subsets. The connection probabilities are
$$\Pi = \begin{pmatrix} b & a & a \\ a & a & b \\ a & b & a \end{pmatrix}$$
where $a = 0.25$, $b = 0.01 + (1 - \delta^{'}) * (a - 0.01)$. We set another parameter $\delta^{'} \in [0.4, 1.0]$ to measure the degree of closeness where a larger $\delta^{'}$ means less overlap among the three clusters.
- scenario C considers networks created from 3 circular-structured data positions in dimension $P = 2$. Three circles have the same center and the different radius are 1, 5, and 10, respectively. Links are then generated based on the distance between node positions.

By varying the values of δ and $\delta^{'}$ in scenario A (assortative) and scenario B (dissortative), we can model the proximity between each cluster and thus test the robustness of our model in both simple and difficult cases. Then, contrary to standard communities, with strong transitivity (your-friend-is-my-friend effect), scenario C describes the construction of three groups of nodes with little transitivity in each.

Table 2 Experimental clustering results, showing the average ARI plus/minus the standard deviation across ten networks.

	Easy		Hard 1	
Method	Sc.A	Sc.B	Sc.A	Sc.B
SBM	0.945±0.03	**1.000**±0.00	0.683±0.06	0.950±0.09
LPCM	0.922±0.03	0.769±0.15	0.613±0.06	0.540±0.04
VGAE	0.935±0.03	0.999±0.01	0.481±0.07	0.754±0.03
ARVGA	0.884±0.04	0.993±0.00	0.278±0.07	0.792±0.06
AM-GCN	0.824±0.05	0.832±0.03	0.516±0.01	0.565±0.12
FAGCN	0.874±0.04	0.820±0.04	0.626±0.06	0.748±0.04
DeepLPM	**0.959**±0.01	**1.000**±0.00	**0.730**±0.03	**0.984**±0.01

	Hard 2		
Method	Sc.A	Sc.B	Sc.C
SBM	0.305±0.04	0.644±0.08	0.443±0.00
LPCM	0.324±0.07	0.345±0.03	0.415±0.20
VGAE	0.206±0.05	0.386±0.09	0.610±0.03
ARVGA	0.065±0.01	0.239±0.08	**0.631**±0.04
AM-GCN	0.346±0.05	0.304±0.04	0.604±0.08
FAGCN	0.358±0.08	0.341±0.05	0.620±0.08
DeepLPM	**0.373**±0.04	**0.857**±0.02	0.625±0.03

4.2 Benchmark study

In this part, we aim at benchmarking DeepLPM with SBM [6], LPCM [17], VGAE [26], ARVGA [29], AM-GCN [30] and FAGCN [31] on simulated datasets in three scenarios. To facilitate the experiments, we do not consider the covariate information Y in simulated data, thus β in Eq. (4) is set to 0.

Datasets. In the "Easy" situation, scenario A was used with $\delta = 0.95$ and data from scenario B was created with $\delta^{'} = 0.9$. For the "Hard 1" situation, the values of δ and $\delta^{'}$ were set to be 0.6 for both scenario A and B. The value 0.4 was chosen in the situation "Hard 2". The number of nodes for scenario A and B were fixed to 300 and 600, respectively. Finally, in scenario C we simulated networks with 300 nodes.

Results. For each situation, we generated ten different networks and calculated the averaged adjusted rand index [ARI, 38]. On one synthetic network, we ran each model five times and computed the averaged ARI to account for the variance among initializations. Experimental results of clustering are shown in Table 2.

First, focusing on scenario A, we can see that although the networks are simulated according to the LPCM model, LPCM does not exhibit the best performances. It outperforms three deep models ARVGA, AM-GCN and FAGCN in the simple case; in Hard 1, it has better performances than VGAE, ARVGA

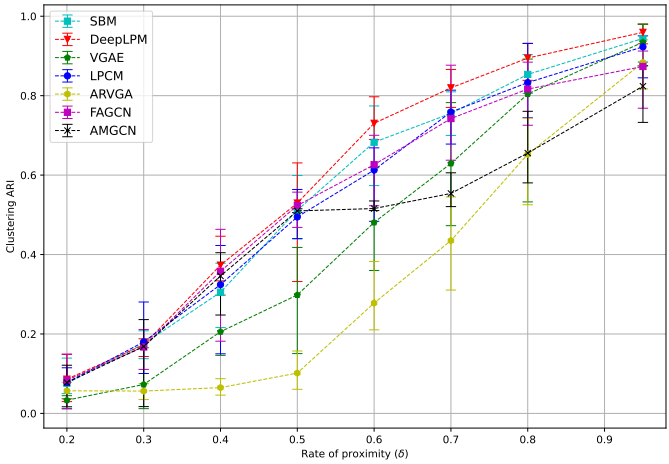

Fig. 4 Clustering ARI with different proximity rate δ in Sc.A. We generated ten different networks for each value of δ and averaged the values of ARI obtained. DeepLPM consistently surpasses its competitors. VGAE and ARVGA perform worse than SBM and LPCM, the other two methods relying on statistical models. FAGCN and AM-GCN present good performance in difficult situations, whereas AM-GCN perform less effectively when δ exceeds 0.5

.

and AM-GCN; and in the more difficult Hard 2 case, it outperforms SBM, VGAE and ARVGA. Among deep models, AM-GCN and FAGCN perform less effectively in the easy situation. ARVGA obtains the worst performance in the hard situation. Instead, DeepLPM always outperforms other competitors with different rate of proximity δ.

Second, considering scenario B, SBM is expected to have good performances in all conditions since the networks are simulated according to SBM. Indeed, it shows better performances than LPCM, VGAE, ARVGA, AM-GCN and FAGCN in the three situations. As a matter of fact, LPCM cannot find clusters on dissortative network structures and the other four deep models only work well in the simple situation. Again, DeepLPM shows the best performance in all cases with high ARI values. AM-GCN and FAGCN do not demonstrate comparable performances.

Lastly, on the circular-structured data, all deep learning-based methods perform better than the ones based on statistical models. ARVGA presents the highest ARI compared to the other deep models. DeepLPM and FAGCN have a slightly lower ARI.

Robustness. To further demonstrate the robustness of DeepLPM compared to other competitors, Figures 4 and 5 illustrate the evolution of the clustering ARI in scenario A and scenario B. For all models, we varied the parameter δ from 0.2 to 0.95 and $\delta^{'}$ from 0.4 to 1 to compare the clustering performances. We can see that DeepLPM has the highest ARI with a small variance in all situations. Moreover, Figure 6 shows the embeddings learned by ARVGA,

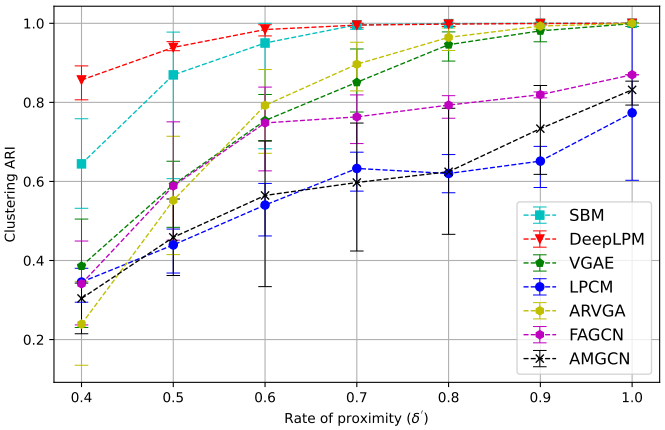

Fig. 5 Clustering ARI with different proximity rates $\delta^{'}$ in Sc.B. We generated ten different networks for each value of $\delta^{'}$ and averaged the values of ARI obtained. When $\delta^{'}$ is greater than 0.6, both DeepLPM and SBM can recover the true node partitions perfectly (ARI=1), whereas SBM cannot maintain its robustness when $\delta^{'}$ is less than 0.6. LPCM and AM-GCN are unsuitable for this type of data, performing worse than VGAE, ARVGA and FAGCN.

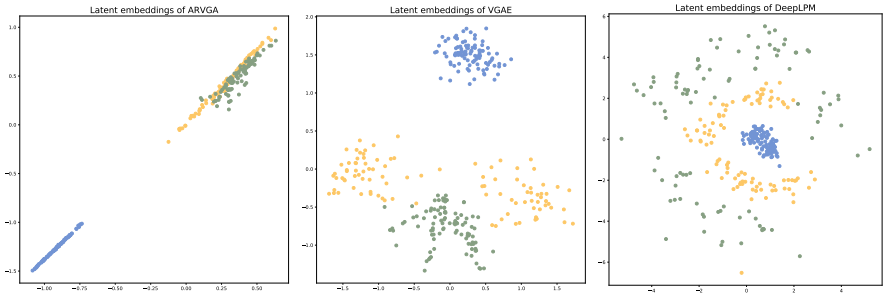

Fig. 6 From left to right: embeddings learned by ARVGA, VGAE and DeepLPM with latent dimension equal to 2 in Sc.C. Only DeepLPM preserves the data circle structure by employing a latent position-based decoding strategy.

VGAE and DeepLPM with latent dimension equal to 2 in scenario C. It can be seen that DeepLPM better preserves the network topology.

4.3 Model selection

A key element of an unsupervised learning technique such as DeepLPM is to be able to automatically determine the number of clusters (K). We highlight here the ability of our methodology to auto-penalize the ELBO for selecting the number of clusters appropriately. Regarding the determination of the latent, intrinsic dimensionality of a graph, we refer to the work of Lelu [39] which employs a randomization method. While this methodology provides valuable insights, our present study focuses on the clustering of the nodes and therefore on the selection of the number of clusters. A comprehensive exploration of the latent dimension is an avenue we plan to investigate in future work.

Firstly, by varying the number of clusters from 2 to 6, Figure 7 illustrates how the training loss (negative ELBO) can be used to find the appropriate number of clusters. In this experiment, for each number of clusters, we trained 10 synthetic data in scenario B ($\delta^{'} = 0.5$) with the latent dimension $P = 16$. The results show that when $K = 3$, the training loss is minimal, thus recovering the actual value of K for the simulation setting.

Similarly, to explore the robustness of our approach with respect to the choice of P, Table 3 presents the averaged training loss and standard derivations across 10 networks simulated according to scenario A ($\delta = 0.6$), involving various latent dimensions ($P = \{2, 4, 8, 16\}$) and varying cluster numbers ($K \in [2, 6]$). DeepLPM was able to recover the true value $K = 3$ by displaying a clear minimum of the negative ELBO whatever the value of P considered.

5 Analysis of a medieval network

As an illustration of the practical application of DeepLPM, it is used here on a real-world dataset coming from historical science.

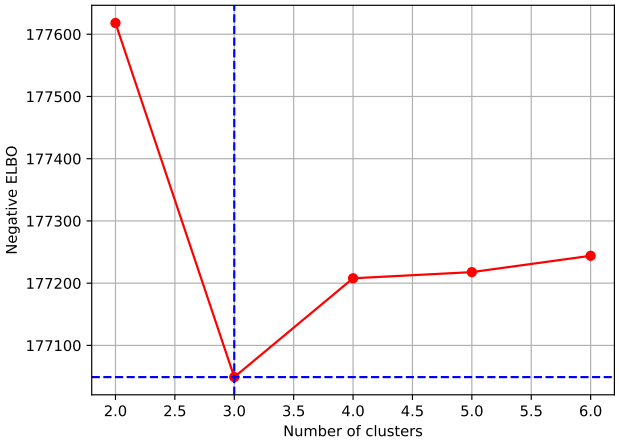

Fig. 7 Averaged training loss (negative ELBO) with different number of clusters on 10 synthetic data in scenario B. DeepLPM was able to estimate $K = 3$ by displaying a clear minimum of the negative ELBO, which recovered the actual number of clusters.

Table 3 Average training loss (negative ELBO) ± standard deviation ($\times 10^{-2}$) for various numbers of clusters and latent dimensions across 10 simulated networks from Scenario A.

	K=2	K=3	K=4	K=5	K=6
P=2	500.93±2.92	500.26±2.86	500.49±3.01	500.48±3.00	500.52±3.02
P=4	504.35±2.95	502.26±2.87	502.35±2.92	502.31±2.89	502.34±2.90
P=8	507.08±2.90	503.15±2.73	503.30±2.85	503.16±2.83	503.17±2.86
P=16	509.01±3.08	505.29±3.75	505.77±3.76	505.78±3.85	505.93±3.85

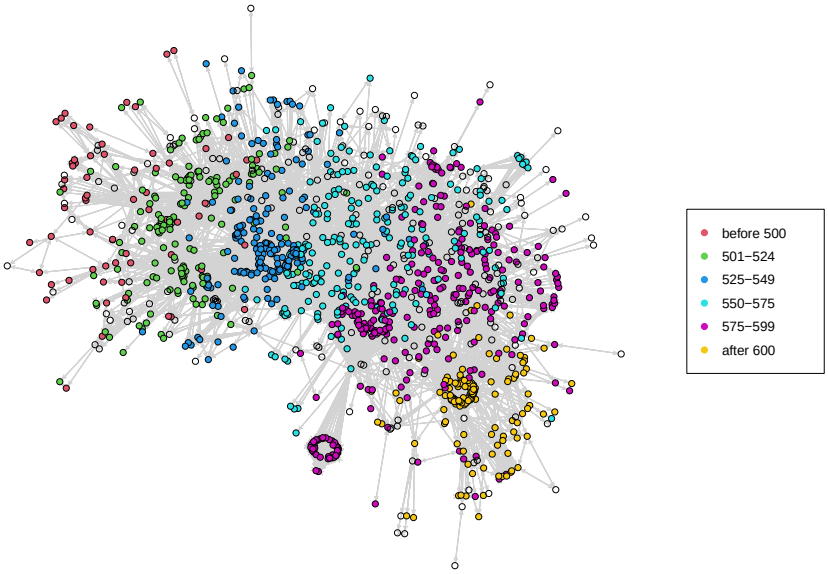

Fig. 8 Visualization of the ecclesiastical network, highlighting the temporality of the relationships. People living in distinct time periods during the 5th and 6th centuries are represented by different colors.

5.1 Dataset

We consider the data set proposed by [11], which reports the ecclesiastical councils that took place in Merovingian Gaul during the 5th and 6th centuries. A council is an ecclesiastical meeting, usually called by a bishop, where issues regarding the church or the faith are addressed. The composition of these councils is known thanks to the acts written at the end of the meeting, and which were signed by all the attending members. The network contains $N = 1,287$ individuals who held one or several offices in Gaul between the years 480 and 614, and who either have been related or have at least met during their lifetime. The number of edges is equal to 33,384. Figure 8 shows a visualization of the network highlighting the importance of the temporality in the relationships. The standard network visualization tool *gplot* of the *sna*[1] library in R is used. Clusters are represented in distinct colors with a layout using a variant of Fruchterman and Reingold force-directed placement algorithm [40] by default.

[1] https://CRAN.R-project.org/package=sna

In addition to the interaction data, the data set also contains information about the individuals: period of activity, type of position and location. From this covariate information, we were able to build a 3-dimensional tensor Y encoding the similarities and differences between individuals. Thus, $Y_{ij}^{(1)}$ is equal to the number of years for which i and j have been active at the same time or, alternatively, the negative time lag (in years) between their period of activity; $Y_{ij}^{(2)} = 1$ if i and j were in the same region, -1 otherwise; $Y_{ij}^{(3)} = 1$ if i and j held a similar position (noble, ecclesiastical or other), -1 otherwise. As a result, those who share a greater number of active years, live in the same location, or hold similar positions are more likely to interact.

5.2 Results without covariates

We first analyze hereafter the clustering results without taking into account the covariate information encoded in Y. DeepLPM was applied to this network for various numbers of groups K (varying from 2 to 10) and a fixed number of latent space dimensions ($P = 16$). When we ignore the covariate information, the evolution of the training loss shows a clear minimum at $K = 9$. Figure 9 depicts the visualization of node partitioning into 9 groups found by DeepLPM without the use of covariates. It is worth noticing that DeepLPM has not been influenced by the temporality since it was able to detect communities that played a similar role in the network at different periods. For instance, the groups #3 and #5 gather people who lived at different and not overlapping time periods. In Figure 10, we also show the distributions in each group when personal roles are taken into consideration. In particular, we can notice that the group #5 is essentially made of ecclesiastics contrary to all the other clusters. We also point out that most nobles (in proportion) were found in the group #4. In addition, it can be seen that citizens played a significant role in the group #2.

5.3 Results with covariates

To demonstrate the effectiveness of the covariates, we integrated the covariates encoded in a 3-dimensional vector Y into DeepLPM and performed clustering on this network. The number of groups also varied between 2 to 10, with a same intrinsic dimension equals to 16. When edge features are added to the methodology, the number of groups is estimated to be $K = 8$ with minimal loss. Compared to previous Section 5.2, the number of groups is reduced by one.

Confusion matrix. We first plot the confusion matrix between the predicted labels at $K = 9$ without covariates and $K = 8$ with covariates to investigate the fusion or dispersion between multiple clusters, as shown in Figure 11. We can see that, by introducing the covariate, the cluster $N1$ gathers people from clusters $O3$, $O6$ and $O8$ together; then, it separates the individuals from $O8$ into $N1$, $N3$ and $N6$; and all the people in $N5$ and $N8$ come from $O4$ and $O9$, respectively. Therefore, the exploitation of the covariate information

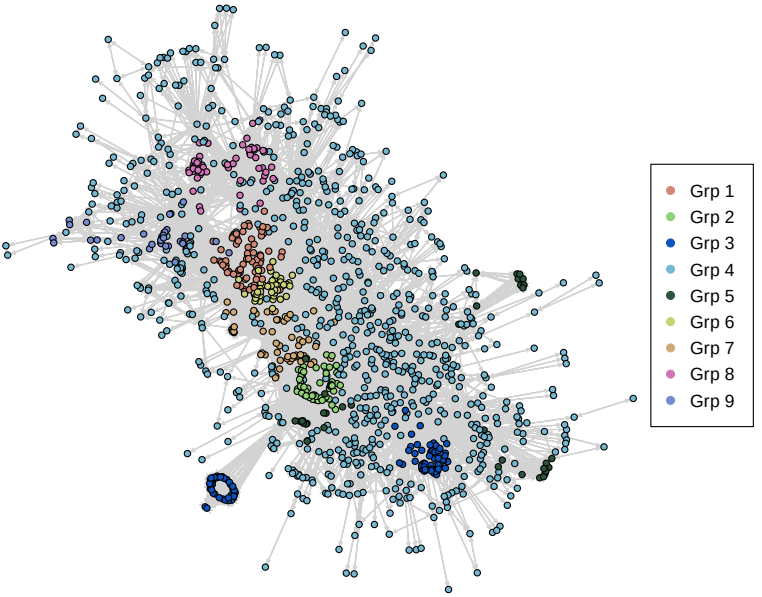

Fig. 9 Node partitions without the covariate information on medieval data. Nine clusters are represented in different colors. DeepLPM was able to find communities at multiple time periods without being influenced by the network temporality; groups #3 (dark green), #4 (cyan), and #5 (dark blue) in particular are clusters that share the same color while lying in distinct periods.

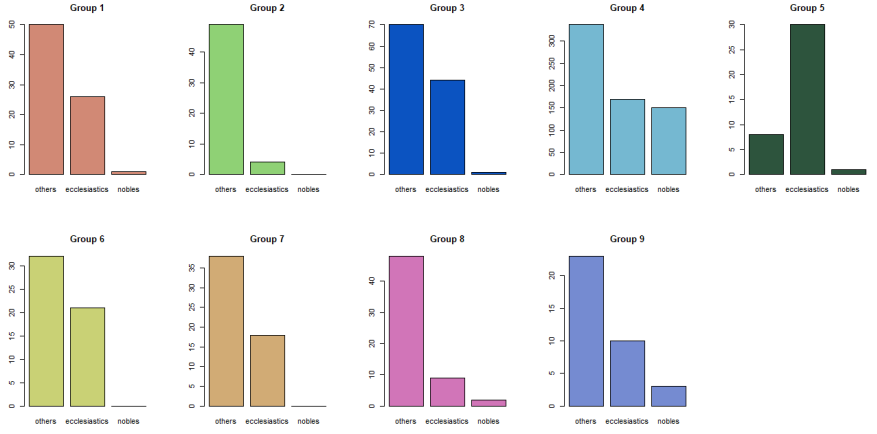

Fig. 10 Distributions in each group based on personal roles on medieval data. Nobles are generally found in groups #4 and #9, ecclesiastics play a large role in group #5, and civilians predominate in group #2.

results in personnel switching between groupings and a reduction in the number of clusters, allowing DeepLPM to focus on patterns that are not explained by the covariates alone.

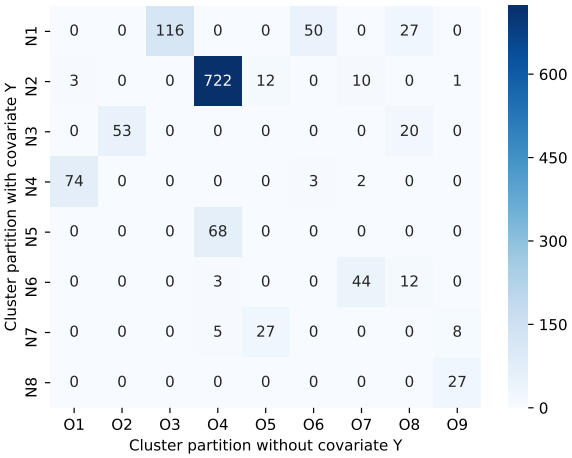

Fig. 11 Confusion matrix between cluster partitions with and without covariate. Individuals from $O3$, $O6$, and almost half of $O8$ congregated in $N1$, reducing one cluster.

Visualization and analysis. Figures 12 shows the clustering visualization obtained by DeepLPM for 8 groups with covariates. We can see that DeepLPM was able to detect communities that played a similar role in the network at different periods. Indeed, the groups #1 (black), #2 (red), #3 (green), #5 (cyan), #6 (pink) and #7 (yellow), all gather people who lived at different and not overlapping time periods. The red group #2 is particularly representative of this since it covers the whole period (480-614 of our era). In addition, distributions within each group based on personal functions are also shown in Figure 13. Firstly, we can observe that the majority of people in groups #3 and #8 were civilians; then, it can be noticed that ecclesiastics was a significant component of the group #7; Furthermore, we point out that groups # 2 and #8 included individuals from all strata of society.

Comparisons of results without and with Y. We further analyze the results by comparing the partitions without and with covariates in Figures 9 and 12. Firstly, we can see that the group #1 (black) in Figure 12 extracted some individuals from groups $O6$ and $O8$ in Figure 9 and kept people in the group $O3$. Combining with Figures 10 and 13, the group #1 in Figure 12 is specific since it only gathers people from clergy and civilians, who were probably discussing some central questions about the faith during different periods. Then, the group #3 (green) retained individuals from cluster $O2$ and retrieved some civilians from cluster $O8$, indicating that they were possibly addressing civilian issues. Similarly, the group #6 (pink) kept the majority of its members in $O7$ while also bringing in people from $O4$ and $O8$. They may share religious concerns in relation to their positions. Likewise, the group #5 (cyan) in Figure 12 is specially separated from the big group $O4$ in Figure 9, which is made of a relatively significant proportion of nobles, in particular

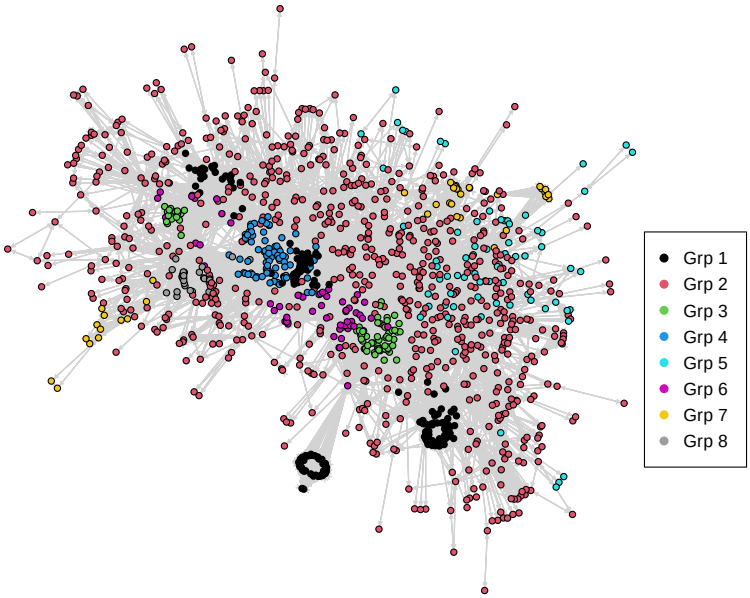

Fig. 12 Visualization of cluster partitions with covariates on medieval data. Eight clusters are represented in different colors. DeepLPM was able to find communities at multiple time periods without being influenced by the network temporality; groups #1, #2, #3, #5, #6 and #7, for example, are clusters with the same colors but located at different times.

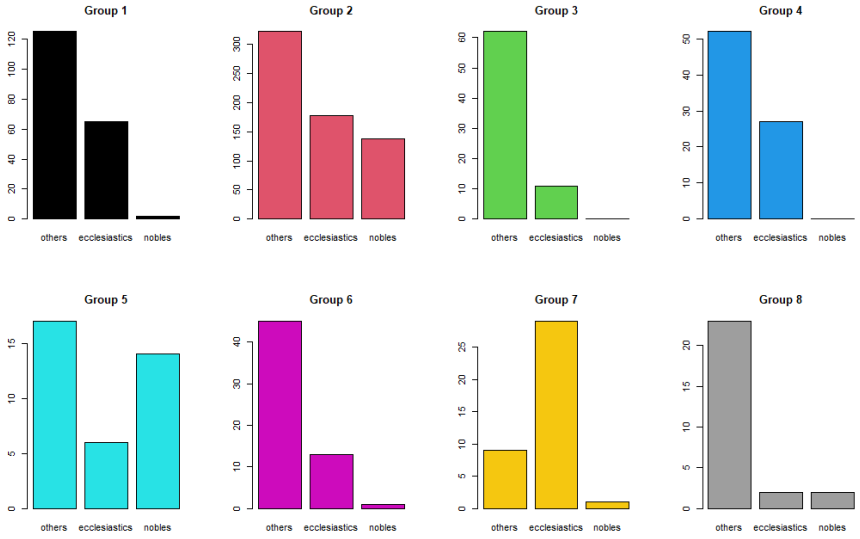

Fig. 13 Distributions in each group based on personal roles on medieval data. Civilians are generally found in groups #3 and #8, ecclesiastics play a large role in group #7, and all social classes are represented in groups #2 and #5.

kings and queens, implying that this group was discussing political or nobility matters.

6 Cora citation network

In this section, we also conduct an unsupervised analysis on a widely used scientific citation network.

6.1 Dataset

The Cora dataset has been analyzed with several embedding and clustering (deep) methods. The dataset contains 2,708 scientific publications classified in seven classes: *case based*, *genetic algorithms*, *neural networks*, *probabilistic methods*, *reinforcement learning*, *rule learning* and *theory*. The citation network consists of 5,429 links and each publication is described by a 0/1-valued word vector indicating the absence/presence of the corresponding word from a dictionary.

Most related works [29, 41] assume that the number of clusters is equal to the number of classes used in supervised classification tasks, whereas we argue that the class labels might not be in a one-to-one relation with the detected communities in unsupervised clustering. Instead, an appropriate cluster number should be obtained through model selection. Thus, we decided to use the class membership of each paper to build a tensor Y of dimension $D = 7 \times 7$ encoding the similarities and differences between articles. For each pair of papers i and j with category labels s_i and s_j, $Y_{s_i s_j} = 1$ indicates that paper i belongs to the class s_i and j belongs to the class s_j, 0 otherwise.

6.2 Results without covariates

We first performed clustering without considering the covariate information. DeepLPM was fitted to this network for different numbers of groups, ranging between 5 and 11, and fixed latent dimension ($P = 16$) for the latent space. The number of groups is estimated to be $K = 9$ with minimum loss.

A visualization of Cora, generated by applying PCA to the latent embeddings learned by DeepLPM, is shown in Figure 14. We can see that even though groups #4 and #5 are very close to each other, and there are some overlaps between groups #2 and #9, DeepLPM globally produces discriminant embeddings. Then, to get an idea of the composition of each group, the distribution of the papers in the nine clusters according to the seven categories is given in Figure 15. In particular, groups #1, #4, #6, #7 and #8 focus on subjects related to *neural networks*, *theory*, *probabilistic methods*, *case based* and *genetic algorithms*, respectively; the group #9 is mainly based on *reinforcement learning*.

Without considering covariates, it is clear that most clusters, such as #1, #4, #6, #7, #8 and #9 contain primarily one category of papers, which coincides with the known supervised information. However, we also see groups #2,

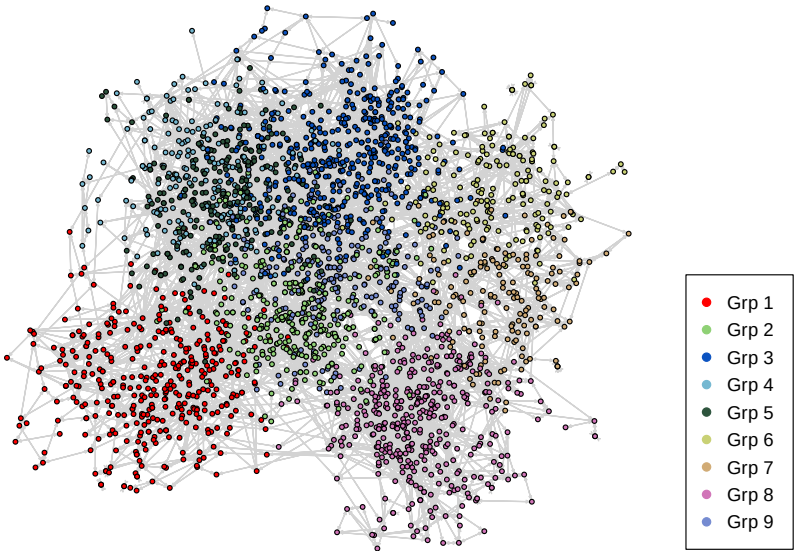

Fig. 14 PCA visualization of the clustered embeddings without covariates on Cora. Nine clusters are represented in different colors.

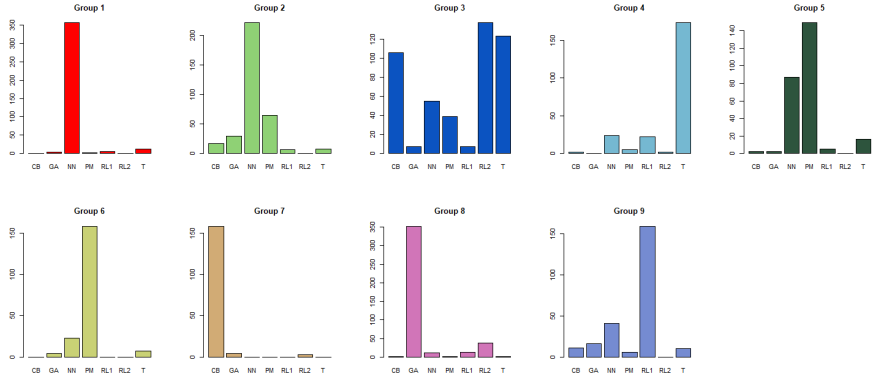

Fig. 15 Partitions without covariates taking into account the classes in each group on Cora. Most groups consist primarily of one type of paper, while a few groups include multiple categories. Here CB: Case_Based, GA: Genetic_Algorithms, NN: Neural_Networks, PM: Probabilistic_Methods, RL1: Reinforcement_Learning,RL2: Rule_Learning, T: Theory.

#3 and #5 containing more categories. This is not surprising. Indeed, when looking at papers from some peculiar categories (e.g. neural nets, probabilistic methods or theory) we discover that they cover topics from other categories. For instance, several probabilistic approaches are build upon neural networks, or some theoretical papers can refer to neural network-based techniques. The clusters containing papers from different categories, clearly account for this

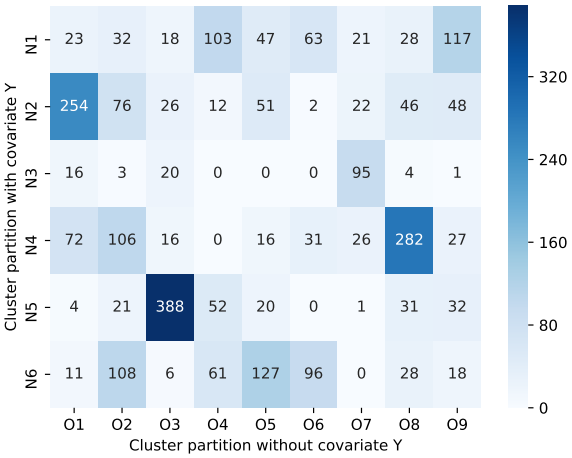

Fig. 16 Confusion matrix between the estimated clusters with and without covariates. The previous nine clusters are reduced to six with the addition of covariates.

"contaminations". Therefore, we want to give DeepLPM the known class labels and let the model dig for more information hidden behind them.

6.3 Results with covariates

To show the impact of the covariate information, we now adopt the covariates Y. The model selection was also conducted by varying the number of clusters from 5 to 11, with the dimensionality of the latent space equal to 16. Based on the evolution of the training loss, the number of groups was estimated to be $K = 6$. Thus, with this additional covariate, the clusters number is reduced by three.

Confusion matrix. We first plot the confusion matrix between the predicted labels at $K = 9$ (without covariates) and $K = 6$ (with covariates) to investigate the fusion or dispersion between multiple clusters, as shown in Figure 16. As we can see, the cluster $N1$ extracted papers from 9 different clusters $O1$ to $O9$, especially from $O4$ and $O9$; $N2$ assembled publications mainly from clusters $O1$ and $O2$; besides, most of the items in $N3$ and $N5$ come from $O7$ and $O3$, respectively; finally, $N4$ consists of quantitative papers from $O1$, $O2$ and $O8$, $N6$ is mainly composed of articles of $O2$, $O5$ and $O6$. The fact that each cluster now contains multiple categories of publications demonstrates that the addition of covariates helps to reveal hidden patterns behind supervised class information when performing clustering.

Visualization and analysis. A visualization of Cora learned by DeepLPM is presented in Figure 17, where PCA has been applied to the latent embeddings. Figure 18 shows the paper distributions when considering the class labels for six groups. In contrast to Figure 15 where each group contains principally one or two different classes, it is clear that, due to the introduction of paper

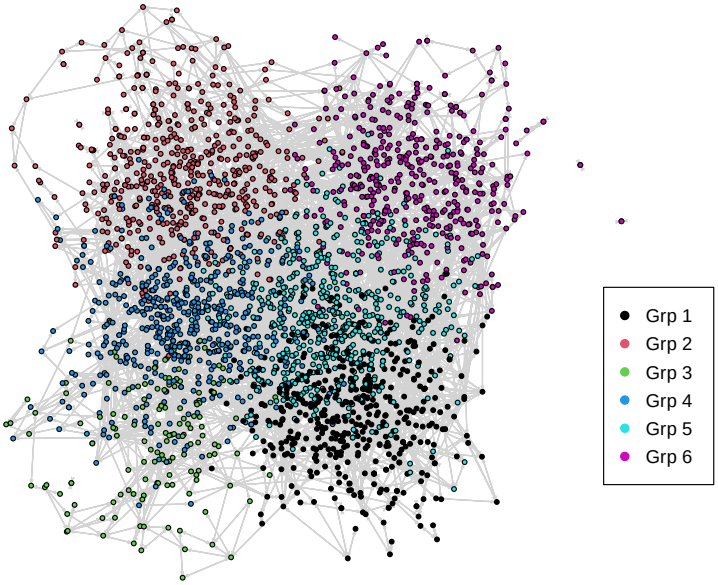

Fig. 17 PCA visualization of the clustered embeddings with covariates on Cora. Six clusters are represented in distinct colors.

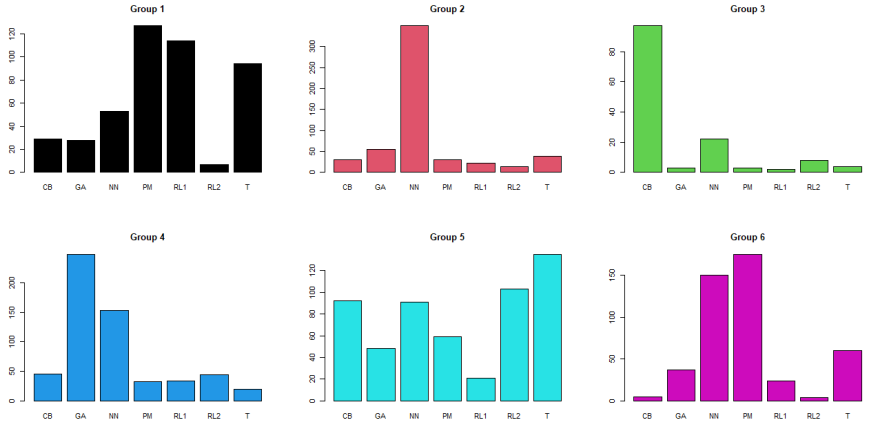

Fig. 18 Partitions with covariates taking into account classes in each group on Cora. Each group now contains a variety of categories that represent the hidden patterns discovered through the addition of covariates.

labels as covariates, new similarities between papers in different categories emerge.

Next, to better understand the clustering results, more analysis on the obtained clusters are performed. We first plotted the latent positions learned

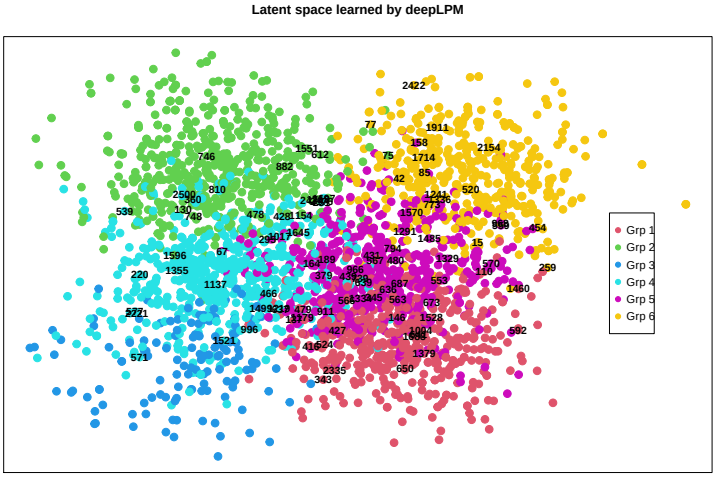

Fig. 19 Learned hidden space (PCA compression on the first two principal components), highlighting the nodes with degrees higher than 10.

by DeepLPM using PCA with a projection of the first two principal eigenvectors in Figure 19, highlighting nodes with degrees higher than 10. Those papers are more often cited by other papers and can be more representative. Based on the publications ID, we selected several articles with relatively large degree from each group and reported the information in Table 4. According to paper titles, it can first be seen that group #1 (red) focuses on dynamic or temporal learning algorithms using probabilistic methods or reinforcement learning; group #2 (green) then discusses different aspects of neural networks, such as self-organization, adjusting or rules; in group #3 (blue), the papers are largely based on the analysis and development of case studies; next, group #4 (cyan) contains articles on applications of genetic algorithms and neural networks; while in group #5 (purple), papers consist of rule learning and inductive methods; finally, group #6 (yellow) typically involves statistical and machine learning models.

Interestingly, when looking at Figure 19 from left to right, the content is changing from applied research to more theoretical learning, and then from bottom to top, the topic of the articles is changing from case-based methods and reinforcement learning to genetic algorithms, and finally to neural networks and statistical models.

Table 4: Inspection of some nodes/documents having large degree

Groups	Node IDs	Paper titles	Degrees
	#524	*Studies in machine learning using the game of checkers*	30

Grp 1

	#553	*Learning to act using real-time dynamic programming*	42
	#566	*Learning to predict by the methods of temporal differences*	78
	#567	*Integrated architectures for learning, planning, and reacting based on approximating dynamic programming*	32
	#673	*Cryptographic limitations on learning boolean formulae and finite automata*	21
Grp 2	#130	*Evolving networks: using the genetic algorithm with connection learning*	15
	#295	*Neuronlike adaptive elements that can solve difficult learning control problems*	32
	#746	*Self-organized formation of topologically correct feature maps*	33
	#748	*Self-organization and associative memory*	74
	#810	*Self-adjusting dynamic logic module*	14
	#882	*Proben1 | A set of neural network benchmark problems and benchmarking rules*	14
Grp 3	#137	*Theory refinement combining analytical and empirical methods*	19
	#1499	*Inferential theory of learning: developing foundations for multistrategy learning*	12
Grp 4	#164	*Genetic algorithms in search, optimization and machine learning*	168
	#428	*Introduction to the theory of neural computation*	65
	#571	*A new learning algorithm for blind signal separation*	19
	#1355	*The structure-mapping engine: algorithm and examples*	23
	#1521	*Adaptive nonlinear PCA algorithms for blind source separation without prewhitening*	18
Grp 5	#345	*Learning logical relations from definitions*	31
	#379	*An empirical comparison of selection measures for decision-tree induction*	26
	#431	*Irrelevant features and the subset selection problem*	36
	#636	*Learning with many irrelevant features*	21
	#911	*Learning sequential decision rules using simulation models and competition*	22

Grp 6	#15	*Hidden Markov models in computational biology: applications to protein modeling*	19
	#42	*Markov chain Monte Carlo convergence diagnostics: a comparative review*	14
	#75	*Hierarchical mixtures of experts and the EM algorithm*	40
	#77	*A view of the EM algorithm that justifies incremental, sparse, and other variants*	16
	#454	*How to use expert advice*	23
	#794	*A survey of evolution strategies*	23

We close this section emphasizing once more that, in unsupervised problems, we cannot determine the number of clusters solely based on the number of the classes that are used in supervised tasks. Conversely, when selecting the number of clusters via model selection (that VAEs seem to perform intrinsically), we are able to discover interesting new similarities between the nodes of a graph.

7 Conclusion

We introduced DeepLPM to perform node clustering on network data in an end-to-end manner. By integrating the GCN encoder with the LPM-based decoder, we retain the interpretability of the statistical model while also enjoying the excellent performance of neural networks in representation learning. An original estimation procedure combined the explicit optimization via variational inference and the implicit optimization using stochastic gradient descent. Numerical experiments show that DeepLPM outperforms state-of-the-art methods and highlight its capabilities in terms of model selection. Real-world applications on a historical network and a scientific citation network were also proposed to illustrate the interest of the method for unsupervised analysis.

To assist users further, we have found that a latent dimension of $P = 16$ generally yields meaningful outcomes across various datasets in practice. Therefore, we suggest this as a starting point for users. For the number of clusters K, we recommend initially exploring the range of 2-10. This range is a good balance between computational efficiency and the ability to capture diverse data structures. However, if the minimum of the negative ELBO is not distinctly achieved within this range, we advise extending the range of K for further exploration.

For future work, we are interested into analyzing textual edges by incorporating topic modeling. Moreover, the primary focus of our current study is on the identification and analysis of non-overlapping clusters. Further

research could certainly explore extending DeepLPM to more effectively address overlapping scenarios.

Acknowledgments. This work has been supported by the French government, through the 3IA Côte d'Azur Investment in the Future Project managed by the National Research Agency (ANR) with the reference numbers ANR-19-P3IA-0002.

Declarations

- Funding: this work was funded by the French National Research Agency (ANR) with the reference numbers ANR-19-P3IA-0002.
- Conflict of interest/Competing interests: the authors have no relevant financial or non-financial interests to disclose.

Appendix A Derivatives of the ELBO

We perform the explicit optimization of the ELBO with respect to the parameters $\gamma_{ik}, \pi_k, \mu_k$ and σ_k by calculating the derivatives of the ELBO.

Under the equality constraint $\sum_{k=1}^{K} \gamma_{ik} = 1, \forall k$, we use the method of Lagrange multipliers. Firstly, we introduce a Lagrange multiplier λ_i

$$\tilde{\mathcal{L}} = \mathcal{L} - \sum_{i=1}^{N} \lambda_i \left(\sum_{k=1}^{K} \gamma_{ik} - 1 \right), \tag{A1}$$

then, we derive $\tilde{\mathcal{L}}$ according to γ_{ik}

$$\frac{\partial \tilde{\mathcal{L}}}{\partial \gamma_{ik}} = \log \pi_k - \log \gamma_{ik} - \frac{\gamma_{ik}}{\gamma_{ik}} - D_{KL}^{ik} - \lambda_i = 0, \tag{A2}$$

thus, we have

$$\log \gamma_{ik} = \log \pi_k - 1 - D_{KL}^{ik} - \lambda_i \tag{A3}$$

$$\gamma_{ik} = \mathrm{e}^{\{\log \pi_k - 1 - D_{KL}^{ik} - \lambda_i\}} = \frac{\mathrm{e}^{\{\log \pi_k - D_{KL}^{ik}\}}}{\mathrm{e}^{\{1+\lambda_i\}}}. \tag{A4}$$

By using the constraint on $\sum_{k=1}^{K} \gamma_{ik}$, we can get

$$\sum_{k=1}^{K} \gamma_{ik} = \frac{\sum_{k=1}^{K} \mathrm{e}^{\{\log \pi_k - D_{KL}^{ik}\}}}{\mathrm{e}^{\{1+\lambda_i\}}} = 1 \tag{A5}$$

$$\log \sum_{k=1}^{K} \mathrm{e}^{\{\log \pi_k - D_{KL}^{ik}\}} = \log \mathrm{e}^{\{1+\lambda_i\}} \tag{A6}$$

$$\lambda_i = \log \sum_{k=1}^{K} \mathrm{e}^{\{\log \pi_k - D_{KL}^{ik}\}} - 1. \tag{A7}$$

After putting the value of λ_i into Eq. A4

$$\gamma_{ik} = \frac{\mathrm{e}^{\{\log \pi_k - D_{KL}^{ik}\}}}{\mathrm{e}^{\{1+\log \sum_{k=1}^{K} \mathrm{e}^{\{\log \pi_k - D_{KL}^{ik}\}} - 1\}}} = \frac{\mathrm{e}^{\{\log \pi_k - D_{KL}^{ik}\}}}{\sum_{k=1}^{K} \mathrm{e}^{\{\log \pi_k - D_{KL}^{ik}\}}}. \tag{A8}$$

Finally, we obtain

$$\hat{\gamma}_{ik} = \frac{\pi_k \mathrm{e}^{-D_{KL}^{ik}}}{\sum_{l=1}^{K} \pi_l \mathrm{e}^{-D_{KL}^{il}}}, \tag{A9}$$

where $D_{KL}^{ik} = \frac{1}{2}\left\{\log \frac{(\sigma_k^2)^P}{(\tilde{\sigma}_\phi^2(\overline{A})_i)^P} - P + \frac{\tilde{\sigma}_\phi^2(\overline{A})_i}{\sigma_k^2} + \frac{1}{\sigma_k^2}\mu_k - \tilde{\mu}_\phi(\overline{A})_i{}^2\right\}$.

Similarly, since $\sum_{k=1}^{K} \pi_k = 1, \forall k$, we introduce another Lagrange multiplier β

$$\tilde{\mathcal{L}} = \mathcal{L} - \beta\left(\sum_{k=1}^{K} \pi_k - 1\right), \tag{A10}$$

then, we derive $\tilde{\mathcal{L}}$ according to π_k

$$\frac{\partial \tilde{\mathcal{L}}}{\partial \pi_k} = \sum_{i=1}^{N} \frac{\gamma_{ik}}{\pi_k} - \beta = 0, \tag{A11}$$

next, we use the equality constraint to solve β

$$\sum_{k=1}^{K}\sum_{i=1}^{N} \gamma_{ik} = \sum_{k=1}^{K} \pi_k \beta \tag{A12}$$

$$\beta = N, \tag{A13}$$

and finally, we have

$$\hat{\pi}_k = \sum_{i=1}^{N} \gamma_{ik}/N. \tag{A14}$$

Lastly, we need to calculate the derivatives for μ_k and σ_k^2. We start by deriving $\tilde{\mathcal{L}}$ according to μ_k

$$\frac{\partial \tilde{\mathcal{L}}}{\partial \mu_k} = -\frac{1}{2}\sum_{i=1}^{N}\gamma_{ik}\{\frac{1}{\sigma_k^2}(2\mu_k - 2\tilde{\mu}_\phi(\overline{A})_i\} = 0, \tag{A15}$$

then, we obtain

$$\mu_k \sum_{i=1}^{N}\gamma_{ik} = \sum_{i=1}^{N}\tilde{\mu}_\phi(\overline{A})_i\gamma_{ik} \tag{A16}$$

$$\hat{\mu}_k = \frac{\sum\limits_{i=1}^{N}\tilde{\mu}_\phi(\overline{A})_i\gamma_{ik}}{\sum\limits_{i=1}^{N}\gamma_{ik}}, \tag{A17}$$

and finally for σ_k^2, we have

$$\begin{aligned}
\frac{\partial \tilde{\mathcal{L}}}{\partial \sigma_k^2} = -\frac{1}{2}\sum_{i=1}^{N}\gamma_{ik}\{\frac{P}{\sigma_k^2} - \frac{1}{\sigma_k^4}(\sigma_\phi^2(\overline{A})_i + \mu_k - \tilde{\mu}_\phi(\overline{A})_i{}^2)\} = 0 \\
P\sum_{i=1}^{N}\frac{\gamma_{ik}}{\sigma_k^2} = \sum_{i=1}^{N}\frac{\gamma_{ik}}{\sigma_k^4}(\sigma_\phi^2(\overline{A})_i + \mu_k - \tilde{\mu}_\phi(\overline{A})_i{}^2) \\
P\sum_{i=1}^{N}\gamma_{ik}\sigma_k^2 = \sum_{i=1}^{N}\gamma_{ik}(\sigma_\phi^2(\overline{A})_i + \mu_k - \tilde{\mu}_\phi(\overline{A})_i{}^2) \\
\hat{\sigma}_k^2 = \frac{\sum\limits_{i=1}^{N}\gamma_{ik}(\sigma_\phi^2(\overline{A})_i + \mu_k - \tilde{\mu}_\phi(\overline{A})_i{}^2)}{P\sum\limits_{i=1}^{N}\gamma_{ik}}.
\end{aligned} \tag{A18}$$

Appendix B Implementation details and computation time

In DeepLPM, the GCN encoder has 64 neurons in the first hidden layer and 16 neurons in the second hidden layer respectively, equipped with a Relu activation for the first layer. The decoder is a one-layer neural network which maps the latent embeddings into a reconstructed graph, following by a sigmoid activation function. Adam optimizer is used instead of the classical stochastic gradient descent procedure to update network weights. The computation time on the ecclesiastical network with 1,287 nodes and 33,384 edges is about 0.12s/epoch, for a total of 107s for 800 epochs on a GeForce RTX 2070 GPU. On the same GPU, training on the citation network Cora with 2,708 nodes and 5,429 edges takes about 0.18s/epoch and a total of 444.37s for 2,000 epochs.

In this paper, LPCM is implemented by *VBLPCM* pakage in R, SBM, VGAE, ARVGA, AM-GCN and FAGCN are conducted using the available Python code in github: https://github.com/Remi-Boutin/SBM_package, https://github.com/DaehanKim/vgae_pytorch, https://github.com/GRAND-Lab/ARGA, https://github.com/zhumeiqiBUPT/AM-GCN, https://github.com/bdy9527/FAGCN, respectively.

References

[1] Schaeffer, S.E.: Graph clustering. Computer science review **1**(1), 27–64 (2007)

[2] Snijders, T.A.: Statistical models for social networks. Annual review of sociology **37**, 131–153 (2011)

[3] Hamilton, W.L., Ying, R., Leskovec, J.: Inductive representation learning on large graphs. In: Proceedings of the 31st International Conference on Neural Information Processing Systems, pp. 1025–1035 (2017)

[4] Zhang, D., Yin, J., Zhu, X., Zhang, C.: Network representation learning: A survey. IEEE transactions on Big Data **6**(1), 3–28 (2018)

[5] Wang, Y.J., Wong, G.Y.: Stochastic blockmodels for directed graphs. Journal of the American Statistical Association **82**(397), 8–19 (1987)

[6] Nowicki, K., Snijders, T.A.B.: Estimation and prediction for stochastic blockstructures. Journal of the American statistical association **96**(455), 1077–1087 (2001)

[7] Lee, C., Wilkinson, D.J.: A review of stochastic block models and extensions for graph clustering. Applied Network Science **4**(1), 1–50 (2019)

[8] Airoldi, E.M., Blei, D.M., Fienberg, S.E., Xing, E.P.: Mixed membership stochastic blockmodels. Journal of machine learning research (2008)

[9] Latouche, P., Birmelé, E., Ambroise, C.: Overlapping stochastic block models with application to the french political blogosphere. The Annals of Applied Statistics, 309–336 (2011)

[10] Mariadassou, M., Robin, S., Vacher, C.: Uncovering latent structure in valued graphs: a variational approach. The Annals of Applied Statistics **4**(2), 715–742 (2010)

[11] Jernite, Y., Latouche, P., Bouveyron, C., Rivera, P., Jegou, L., Lamassé, S.: The random subgraph model for the analysis of an ecclesiastical network in merovingian gaul. The Annals of Applied Statistics **8**(1), 377–405

(2014)

[12] Bouveyron, C., Latouche, P., Zreik, R.: The stochastic topic block model for the clustering of vertices in networks with textual edges. Statistics and Computing **28**(1), 11–31 (2018)

[13] Xu, K.S., Hero, A.O.: Dynamic stochastic blockmodels for time-evolving social networks. IEEE Journal of Selected Topics in Signal Processing **8**(4), 552–562 (2014)

[14] Matias, C., Miele, V.: Statistical clustering of temporal networks through a dynamic stochastic block model. Journal of the Royal Statistical Society: Series B (Statistical Methodology) **79**(4), 1119–1141 (2017)

[15] Corneli, M., Bouveyron, C., Latouche, P., Rossi, F.: The dynamic stochastic topic block model for dynamic networks with textual edges. Statistics and Computing **29**(4), 677–695 (2019)

[16] Hoff, P.D., Raftery, A.E., Handcock, M.S.: Latent space approaches to social network analysis. Journal of the american Statistical association **97**(460), 1090–1098 (2002)

[17] Handcock, M.S., Raftery, A.E., Tantrum, J.M.: Model-based clustering for social networks. Journal of the Royal Statistical Society: Series A (Statistics in Society) **170**(2), 301–354 (2007)

[18] Raftery, A.E.: Comment: Extending the latent position model for networks. Journal of the American Statistical Association **112**(520), 1531–1534 (2017)

[19] Bouveyron, C., Celeux, G., Murphy, T.B., Raftery, A.E.: Model-based Clustering and Classification for Data Science: with Applications in R vol. 50. Cambridge University Press, ??? (2019)

[20] Aljalbout, E., Golkov, V., Siddiqui, Y., Strobel, M., Cremers, D.: Clustering with deep learning: Taxonomy and new methods. arXiv preprint arXiv:1801.07648 (2018)

[21] Zhang, Z., Cui, P., Zhu, W.: Deep learning on graphs: A survey. IEEE Transactions on Knowledge and Data Engineering (2020)

[22] Xie, J., Girshick, R., Farhadi, A.: Unsupervised deep embedding for clustering analysis. In: International Conference on Machine Learning, pp. 478–487 (2016). PMLR

[23] Jiang, Z., Zheng, Y., Tan, H., Tang, B., Zhou, H.: Variational deep embedding: An unsupervised and generative approach to clustering. In:

International Joint Conference on Artificial Intelligence (IJCAI-2017) (2016)

[24] Kingma, D.P., Welling, M.: Stochastic gradient vb and the variational auto-encoder. In: Second International Conference on Learning Representations, ICLR, vol. 19, p. 121 (2014)

[25] Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., Sun, M.: Graph neural networks: A review of methods and applications. AI Open **1**, 57–81 (2020)

[26] Kipf, T.N., Welling, M.: Variational graph auto-encoders. In: NeurIPS Workshop on Bayesian Deep Learning (NeurIPS-16 BDL) (2016)

[27] Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: 5th International Conference on Learning Representations (ICLR-17) (2016)

[28] Wang, C., Pan, S., Long, G., Zhu, X., Jiang, J.: Mgae: Marginalized graph autoencoder for graph clustering. In: Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, pp. 889–898 (2017)

[29] Pan, S., Hu, R., Long, G., Jiang, J., Yao, L., Zhang, C.: Adversarially regularized graph autoencoder for graph embedding. In: International Joint Conference on Artificial Intelligence (IJCAI-18), pp. 2609–2615 (2018)

[30] Wang, X., Zhu, M., Bo, D., Cui, P., Shi, C., Pei, J.: Am-gcn: Adaptive multi-channel graph convolutional networks. In: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 1243–1253 (2020)

[31] Bo, D., Wang, X., Shi, C., Shen, H.: Beyond low-frequency information in graph convolutional networks. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 35, pp. 3950–3957 (2021)

[32] Tian, F., Gao, B., Cui, Q., Chen, E., Liu, T.-Y.: Learning deep representations for graph clustering. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 28 (2014)

[33] Nie, F., Zhu, W., Li, X.: Unsupervised large graph embedding. In: Thirty-first AAAI Conference on Artificial Intelligence (2017)

[34] Zhang, X., Liu, H., Li, Q., Wu, X.-M.: Attributed graph clustering via adaptive graph convolution. arXiv preprint arXiv:1906.01210 (2019)

[35] Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv

preprint arXiv:1412.6980 (2014)

[36] Kingma, D.P., Salimans, T., Jozefowicz, R., Chen, X., Sutskever, I., Welling, M.: Improved variational inference with inverse autoregressive flow. Advances in neural information processing systems **29**, 4743–4751 (2016)

[37] Dai, B., Wang, Y., Aston, J., Hua, G., Wipf, D.: Hidden talents of the variational autoencoder. arXiv preprint arXiv:1706.05148 (2017)

[38] Hubert, L., Arabie, P.: Comparing partitions. Journal of classification **2**(1), 193–218 (1985)

[39] Lelu, A.: Relevant eigen-subspace of a graph: A randomization test. In: CAP 2011, p. 4 (2011)

[40] Fruchterman, T.M., Reingold, E.M.: Graph drawing by force-directed placement. Software: Practice and experience **21**(11), 1129–1164 (1991)

[41] Mehta, N., Duke, L.C., Rai, P.: Stochastic blockmodels meet graph neural networks. In: International Conference on Machine Learning, pp. 4466–4474 (2019). PMLR