# Joint Embedding of Graphs

Shangsi Wang, Carey E. Priebe, Joshua T. Vogelstein, Avanti Athreya, Minh Tang, Vince Lyzinski, Youngser Park

**Abstract**—In this paper, we propose an algorithm to jointly embed multiple graphs into low dimensional space. The embeddings can be used to cluster or classify graphs and vertices by various standard learning algorithms. We demonstrate through simulation studies and real data experiments that our joint embedding algorithm leads to the state of the art clustering and classification performance.

**Index Terms**—Graphs, Embed, Cluster.

## 1 Introduction

In many problems arising in science, graphs appears naturally as data to model complex relationship between a set of objects. Graphs have been used in various application domains as diverse as social networks, internet map, brain connectomics, political voting networks, and many others. The graphs are very high dimensional object with complicated topological structure which makes graphs clustering and classification a challenge to traditional machine learning algorithms. Therefore, a dimension reduction technique is in desperate need to study graphs.

There are a few approaches to reduce the dimension of graphs. First, classical PCA and CCA can operate on edges of graphs. The problem with this approach is that they totally ignore the graphical structure and ends up with features which are not interpretable. Second, features can be extract by computing topological and label statistics from graphs. These statistics commonly include number of edges, number of triangles, average clustering coefficient, maximum effective eccentricity, etc. The problem with this approach is that computing some statistics can be quite expensive and there is no guarantee the statistics computed will be useful for subsequent inference. The third approach is adjacency spectral embedding (ASE) which factorizes the adjacency matrix of the graph. It has good theoretical property for large graphs; however, ASE operates on each graph separately, and to align embeddings of different graphs introduces extra variance.

In this paper, we propose an algorithm to jointly embed multiple graphs into low dimensional space. The algorithm tries to simultaneously factorize all adjacency matrices into symmetric product of low rank matrices. The joint embedding algorithm can be understood as a generalization of ASE. We then can cluster or classify embeddings using standard learning algorithms. We demonstrate through simulation experiments that our graph embedding algorithm leads to top performance for a variety of inference tasks.


- *Shangsi Wang was with the Department of Applied Mathematics and Statistics, Johns Hopkins University, MD, 21218. E-mail: swang127@jhu.edu*
- *Shangsi Wang was with the Department of Applied Mathematics and Statistics, Johns Hopkins University, MD, 21218.*


In the next section, we define some models for random graphs. In section 3, we define joint embeddings of graphs and describe an algorithm to find it. We then provide some theoretical analysis of joint embeddings. Section 5 some data experiments are performed to demonstrate the utility of joint embeddings.

## 2 Setting

The joint embedding algorithm consumes a set of graphs with the same number of vertices. In this paper, we focus on unweighted and undirected graphs for simplicity, although the algorithm works on weighted and directed graphs as well. Let $\{G_i = (V_i, E_i)\}_{i=1}^m$ be $m$ graphs each with $n$ vertices. We require the vertices in these graphs to be matched which means that all the graphs have a common vertex set $V$, so $V_i = V$. The joint embedding algorithm tries to embed all $G_i$s simultaneously into $\mathbb{R}^d$ and represent $G_i$ by a vector $\lambda_i \in \mathbb{R}^d$. Before discussing the joint embedding algorithm, we first introduce three random graph models which provide basis for statistical analysis, and are also used to generate graphs in the simulation experiment.

**Definition** (Stochastic Block Model (SBM)) Let $\pi$ be a prior probability vector for block membership. Denote $\tau = (\tau_1, \tau_2, ..., \tau_n) \sim\in [K]^n$ be a block membership vector, where $\tau_i$ are independent and identically distributed according to a probability vector $\pi \in \mathbb{R}^K$. Denote $B \in [0,1]^{K\times K}$ to be the block connecting probability matrix. Suppose $A$ is a random adjacency matrix given by,

$$P(A|\tau, B) = \prod_{i<j} B_{\tau_i,\tau_j}^{A_{i,j}} (1 - B_{\tau_i,\tau_j})^{(1-A_{i,j})}$$

Then we say $A$ is an adjacency matrix of a K-block stochastic block model graph, and we write $A \sim SBM(\pi, B)$. Sometimes, we may treat $\tau$ as the parameter of interest, then we write $A \sim SBM(\tau, B)$.

**Definition** (Random Dot Product Graph (RDPG)) Let $F$ be a distribution on a set $\mathcal{X} \in \mathbb{R}^d$ satisfying $x^T y \in [0,1]$ for all $x, y \in \mathcal{X}$. Let $X = [X_1^T, X_2^T, ..., X_n^T] \in \mathbb{R}^{n\times d}$. We say $(X, A) \sim RDPG(F)$ if $X_i$ is independent and identically distributed according to $F$ and condition on $X$, $A_{ij}$ is an independent Bernoulli random variable.

$$A_{ij} \sim Bernoulli(X_i^T X_j)$$

Alternatively,

$$P(A|X) = \prod_{i<j} X_i^T X_j^{A_{ij}} (1 - X_i^T X_j)^{1-A_{ij}}$$

Also, we define $P := XX^T$ to be edge probability matrix. When we are actually interested in estimating latent positions $X$, we treat $X$ as parameters and write $A \sim RDPG(X)$.

**Definition** (Multiple Random Eigen Graphs (MREG) ) Let $\{h_k\}_{k=1}^d$ be a set of norm-1 vectors in $\mathbb{R}^n$, and $F$ be a distribution on a set $\mathcal{X} \in \mathbb{R}^d$, satisfying $\sum \lambda[k] h_k h_k^T \in [0,1]^{n\times n}$ for all $\lambda \in \mathcal{X}$, where $[k]$ means the kth entry of vector. We say $\{(\lambda_i, A_i)\}_{i=1}^m$ follows a m-graph d-dimensional multiple random eigen graphs model,

$$\{(\lambda_i, A_i)\}_{i=1}^m \sim MREG(F, h_1, ..., h_d)$$

if $\lambda_i$ is independent and identically distributed according to $F$, and condition on $\lambda_i$, the $(s,t)$ entry of $A_i$ follows independent Bernoulli distribution,

$$A_i[s,t] \sim Bernoulli(\sum_{k=1}^d \lambda_i[k] h_k[s] h_k[t])$$

We will call $P_i := \sum \lambda_i[k] h_k h_k^T$ the edge probability matrix for graph $i$. When $\lambda_i \geq 0$, the equation above is equivalent to $A_i \sim RDPG(X)$, where the kth column of $X$ is $\sqrt{\lambda_i[k]} h_k$. In many applications, we treat $\{\lambda_i\}_{i=1}^m$ as parameters, and write $\{A_i\}_{i=1}^m \sim MREG(\lambda_1, ..., \lambda_m, h_1, ..., h_d)$.

If an adjacency matrix $A \sim SBM(\pi, B)$ and the block connecting matrix $B$ is semi-positive definite, $A$ can also be written as a $RDPG(F)$ with $F$ being a finite mixture of point masses. If an adjacency matrix $A \sim RDPG(X)$, then it is also a 1-graph $MREG(\lambda_1, h_1, ..., h_d)$ with $h_k$ being the normalized kth column of $X$ and $\lambda[k]$ being the vector containing the corresponding square root norm of kth column of $X$. However, 1-graph $MREG(\lambda_1, h_1, ..., h_d)$ is not necessarily a RDPG graph since $\lambda_1$ could contain negative entries which may result an indefinite edge probability matrix. If $m$ RDPG graphs have the edge probability matrices share the same eigenspace, they also follow a m-graph MREG model.

## 3 METHODOLOGY

### 3.1 Joint Embedding of Graphs

Our joint embedding methodology considers a collection of vertex-aligned graphs, and estimates a common embedding space across all graphs and a loading for each graph. It tries to project each adjacency matrix $A_i$ linearly into a space spanned by a set of rank one symmetric matrices $\{\hat{h}_k \hat{h}_k^T\}_{k=1}^d$. The linear coefficient of $A_i$ corresponding to $\hat{h}_k \hat{h}_k^T$ is denoted by $\hat{\Lambda}_{ik}$ which will be called kth loading for graph $i$. To find the rank one symmetric matrices and loadings for graphs, we try to minimize the sum of squared distances between adjacency matrices and their projections. The definitions of joint embeddings of graphs is provided below.

**Definition** (Joint Embeddings of Graphs) Given $m$ graphs $\{G_i\}_{i=1}^m$ with $n$ vertices, denote $A_i \in \{0,1\}^{n\times n}$ the associated adjacency matrices of graph $G_i$. The d-dimensional joint embeddings of graphs $\{G_i\}_{i=1}^m$ are represented by $\hat{\Lambda} \in R_{m\times d}$ and $\hat{H} \in R_{n\times d}$ and they are given by,

$$(\hat{\Lambda}, \hat{H}) = \underset{\Lambda, ||h_k||=1}{\operatorname{argmin}} \sum_{i=1}^m \|A_i - \sum_{k=1}^d \Lambda_{ik} h_k h_k^T\|^2$$

Here, $\|\cdot\|$ means the Frobenius norm, and $\hat{h}_k$ is the kth column of $\hat{H}$.

We will denote the function on the left hand side of the equation by $f(\Lambda, H)$. Alternatively, we may consider another formulation of the problem,

$$\underset{H, D_i}{\operatorname{argmin}} \sum_{i=1}^m \|A_i - H D_i H\|^2 \text{ , subject to } D_i \text{ is diagonal}$$

For identifiability issue, we also require columns of $H$ to be norm 1. When the solution is unique, the two optimization problem should return the same $\hat{H}$ up to a reordering and sign flip. If we put the ith row of $\Lambda$ into a diagonal matrix, it will serve as the minimizer for $D_i$ in the second formulation. In joint embeddings, $\{\hat{h}_k \hat{h}_k^T\}_{k=1}^d$ can be understood a basis for graphs. All adjacency matrix $A_i$ is approximated by a linear combination of the basis. The corresponding loadings are stored in the ith row of $\hat{\Lambda}$. Next, we describe an algorithm to simultaneously find $hat\Lambda$ and $\hat{H}$.

### 3.2 Alternating Descent Algorithm

Joint embeddings of $\{G_i\}_{i=1}^m$ can be recovered by solving the optimization problem. The difficulty here is that $\|A_i - \sum_{k=1}^d \Lambda_{ik} h_k h_k^T\|^2$ is not convex in $h_k$. Therefore, we develop an alternating gradient descent algorithm to minimize the $f(\Lambda, H)$. The algorithm iteratively updates $H$ and $\Lambda$ while treating the other parameters fixed. The algorithm stops when the objective cease to decrease. We will see that updating $\Lambda$ when fixing $H$ is easy, since it is essentially a linear regression problem. However, updating $H$ when fixing $\Lambda$ is hard, due to the problem being non-convex and no closed form solution available. In this case, we utilize gradient information and take a line search strategy to minimize the objective function.

Instead of performing gradient descent on all columns $H$ simultaneously, we consider a greedy algorithm which solves the optimization problem dimension by dimension. Specifically at iteration $k_0$, we fix all estimates of previous dimensions. Objective function is then minimized the by searching $h_{k_0}$ and $k_0$th column of $\Lambda$, that is

$$(\hat{h}_{k_0}, \hat{\Lambda}_{k_0}) = \underset{\Lambda_{k_0}, ||h_{k_0}||=1}{\operatorname{argmin}} \sum_{i=1}^m \|A_i - \sum_{k=1}^{k_0-1} \hat{\Lambda}_{ik} \hat{h}_k \hat{h}_k^T - \Lambda_{ik_0} h_{k_0} h_{k_0}^T\|^2$$

Again, we use $\Lambda_{k_0}$ to denote $k_0$th column of matrix $\Lambda$, and we will denote the function on the left hand side of the equation by $f(\Lambda_{k_0}, h_{k_0})$. In general, to compute d-dimensional embeddings $(\hat{\Lambda}, \hat{H})$, we recursively solve the one dimensional optimization problem above by letting $k_0$ to increase from 1 to d.

There are a few advantages in recursively solving one dimensional problem. First, there are less parameters to fit at each iteration since we are only allowed to change $\Lambda_{k_0}$ and $h_{k_0}$. It makes initialization and optimization procedure much easier compared to optimizing all columns of $H$ the same time. Second, it implicitly enforces an ordering on columns of $H$. Adding $\hat{h}_i$ naturally yields more reduction in the objective function than $\hat{h}_j$ when $i$ is less than $j$. Third, it allows incremental computation. If we fit two joint embeddings; one with $d = 10$ and another one with $d = 20$, the first 10 columns of $H$ estimates would be the same. The disadvantage of performing gradient descent on each dimension separately is that we are more likely to end up at a local minimum, when the problem is structured not in favor of embedding dimension by dimension. In practice, this can be mitigated by running the joint embedding algorithm several times with random initialization.

To update $\Lambda_{k_0}$ and $h_{k_0}$ in one dimensional problem, we need to evaluate two derivatives: $\frac{\partial f}{\partial h_{k_0}}$ and $\frac{\partial f}{\partial \Lambda_{ik_0}}$. Denote the residual matrix $R_{ik_0} = A_i - \sum_{k=1}^{k_0-1} \hat{\Lambda}_{ik}\hat{h}_k\hat{h}_k^T$. The gradient of the objective with respect to $h_{k_0}$ are given by,

$$\frac{\partial f}{\partial h_{k_0}} = \sum_{i=1}^{m} 4\Lambda_{ik_0}(R_{ik} - \Lambda_{ik_0}h_{k_0}h_{k_0}^T)h_{k_0}$$

When updating $\Lambda_{ik_0}$, we can just regress $R_{ik_0}$ on $h_{k_0}h_{k_0}^T$. It yields,

$$\arg\min_{\Lambda_{ik_0}} f(\Lambda_{k_0}, h_{k_0}) =< R_{ik}, h_{k_0}h_{k_0}^T >$$

Algorithm 1 describes the general procedure to compute joint embeddings of graphs $\{G_i\}_{i=1}^n$.

**Algorithm 1** Joint Embedding Algorithm

**procedure** FIND JOINT EMBEDDINGS $\hat{\Lambda}, \hat{H}$ OF $\{A_i\}_{i=1}^m$
  Set residuals: $R_{i1} = A_i$
  **for** $k = 1 : d$ **do**
    Initialize $h_k$ and $\Lambda_k$
    **while** not convergent **do**
      Compute objective $\sum_{i=1}^{n} \|R_{ik} - \Lambda_{ik}h_kh_k^T\|_F^2$
      Fixing $h_k$, update $\Lambda_k$ by regression
      Fixing $\Lambda_k$, update $h_k$ by gradient descent
    **end while**
    Update residuals: $R_{i(k+1)} = R_{ik} - \Lambda_{ik}h_kh_k^T$
  **end for**
  Output $\hat{\Lambda} = [\Lambda_1, \Lambda_2, ..., \Lambda_d]$ and $\hat{H} = [h_1, h_2, ..., h_d]$
**end procedure**

The basic idea of the algorithm is similar to forward stepwise selection in linear regression setting. The difference between forward stepwise selection and our algorithm is that joint embedding needs to compute its predictors which are $\{\hat{h}_k\hat{h}_k^T\}_{k=1}^d$. The algorithm is also similar to principle component analysis in the sense of minimizing squared reconstruction error. However, there are extra symmetricity and rank assumptions on the components. In case of embedding only one graph, the joint embedding is equivalent to the adjacency spectral embedding.

### 3.3 Variations

The joint embedding algorithm described above can be modified to accommodate several settings.

**Variation 1.** When we are confident all graphs comes from the same model, we can force all $\Lambda_{ik}$s to be equal across graphs. This is useful when primary inference task is to learn embeddings of vertices. Since all graphs share the same loadings, we only need $\Lambda \in \mathbb{R}^d$ and the optimization problem becomes,

$$(\hat{\Lambda}, \hat{H}) = \underset{\Lambda, ||h_k||=1}{\operatorname{argmin}} \sum_{i=1}^{m} \|A_i - \sum_{k=1}^{d} \Lambda_k h_k h_k^T\|^2$$

In this case, the optimization problem can be solved exactly by singular value decomposition of the average matrix of $A_i$.

**Variation 2.** When we have a label $Y_i \in \mathbb{Y}$ associated with graph $G_i$, we may require all $\Lambda_{ik}$s to be equal within class. That is $\Lambda_{ik} = \Lambda_{y_ik}$ for all $i$ with $Y_i = y_i$. If we let $\Lambda \in \mathbb{R}^{|\mathbb{Y}|\times d}$, the optimization problem becomes,

$$(\hat{\Lambda}, \hat{H}) = \underset{\Lambda, ||h_k||=1}{\operatorname{argmin}} \sum_{i=1}^{m} \|A_i - \sum_{k=1}^{d} \Lambda_{y_ik} h_k h_k^T\|^2$$

In this case, when updating $\Lambda$ algorithm 1 should to regress all $R_{ik}$ from the same class simultaneously.

**Variation 3.** We may require all $\Lambda_{ik}$s to be greater than 0 like in the non-negative matrix factorization. The advantage of this constrain is that we may automatically cluster graph $G_i$ based on largest $\Lambda_{ik}$.

$$(\hat{\Lambda}, \hat{H}) = \underset{\Lambda \geq 0, ||h_k||=1}{\operatorname{argmin}} \sum_{i=1}^{m} \|A_i - \sum_{k=1}^{d} \Lambda_{ik} h_k h_k^T\|^2$$

To solve this problem, the algorithm should be modified to apply shrinkage thresholding on $\Lambda$.

## 4 THEORY

With previous simulations, we know $\hat{h} \to h$ is not possible even in fix $lambda$ and only one $h$ cases. However, we think these two conjectures maybe proved,

***Theorem 4.1 (conjecture).*** Under MREG model, $\hat{h}^m \to h'$ as $m$ goes to infinity. (For fixed point mass $\lambda$ or random $\lambda$.)

***Theorem 4.2 (conjecture).*** Under MREG model, $||\hat{h}^m - h|| < e(n, h)$ as $m$ goes to infinity. (We do not require to $e$ converge to 0.)

## 5 EXPERIMENT

Before going into details of experiments, we want to discuss about how to select the dimensionality of embedding $d$. Estimating $d$ is an important model selection question which has been studied for years under various settings. It is not the focus of this paper, but we still face this decision in every experiments we run. In the simulation experiments of this section, we simply just set our dimension estimates $\hat{d}$ equal to the true dimension $d$. In the real data experiment, we recommend two approaches to determine $\hat{d}$. Both

approaches require first run the $d'$ dimensional joint embedding algorithm, where $d'$ is large and we are confident $d$ is less $d'$. Then, we can plot the decrease in the objective function after adding each dimension, and determine $\hat{d}$ where the objective stop to decrease. Alternatively, we can plot $\frac{\sum_{i=1}^{m}|\hat{\Lambda}_{ik}|}{m}$ for $k = 1, ...d'$, and decide where the average loadings start to drop off. These two approaches should yield similar $\hat{d}$ for the data.

### 5.1 Simulation Experiment 1: Joint Embeddings in Simple Settings

This is a numerical example to demonstrate some properties of joint embeddings as the number of graphs goes to infinity. We repeatedly generate graphs with 5 vertices from MREG where $\lambda_i = 1$ for all $i$ and $h_1 = [0.1, 0.2, 0.3, 0.4, 0.5]^T/0.74$. We keep doubling the number of graphs $m$, as it increases from $2^4$ to $2^{16}$. At each value of $m$, we compute 1-dimensional joint embeddings of $m$ graphs through algorithm 1. Denote the estimated parameters based on $m$ graphs by $\hat{\lambda}_i^m$ and $\hat{h}_1^m$. Two quantities are calculated using $\hat{h}_1^m$. The first one is the norm difference between the current $h_1$ estimates and the previous estimates, namely $||\hat{h}_1^m - \hat{h}_1^{\frac{m}{2}}||$. It tries to seek numerical evidences for the convergence of our principled estimation procedure. The second quantity is $||\hat{h}_1^m - h_1||$. It tries to see whether $\hat{h}_1$ is an unbiased estimator for $h_1$. The plot at bottom demonstrates the result.

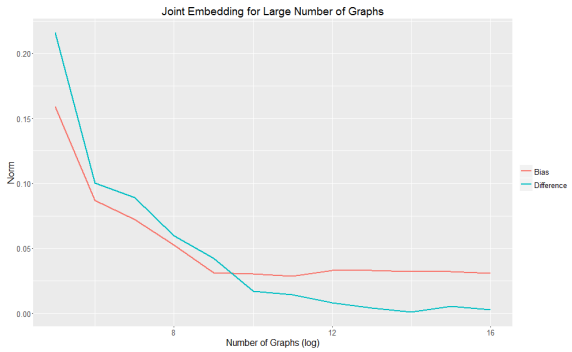


Fig. 1. Bias and Convergence of Joint Embedding

From the plot, we can see that $||\hat{h}_1^m - \hat{h}_1^{\frac{m}{2}}||$ converges to 0 as $m$ increases. It suggests the convergence of $\hat{h}_1^m$. Secondly, we should notice that the bias $||\hat{h}_1^m - h_1||$ does not converge to 0; instead, it stops to decrease at around 0.03. This suggests that $\hat{h}_1$ is an biased and asymptotically inconsistent estimator for $h_1$. The $\hat{h}_1^m$ with $m = 2^{16}$ is $[0.083, 0.186, 0.296, 0.406, 0.503]^T/0.74$. Compared to $h_1$, there are negative biases for small entries of $h_1$, and positive biases for large entries of $h_1$. Actually, there are some theoretical evidences for this phenomenon as well. In general, when there are infinitely many parameters present, in our case $\lambda_i$, we do not expect the unbiasedness of our estimators.

In applications like clustering or classifying graphs, we may not care about $\hat{h}_1$ at all. We are more interested in $\hat{\lambda}_i$ which specifically provides information about graph $i$. The next plot shows boxplots of $\hat{\lambda}_i$ for different values of $m$ using two methods. For each value of m, the first boxplot shows estimates $\hat{\lambda}_i$ which are computed from joint embedding by

$$\hat{\lambda}_i = \langle A_i, \hat{h}_1^m \hat{h}_1^{mT} \rangle$$

The second boxplot shows $\hat{\lambda}_i$ computed by forcing $\hat{h}_1$ equal to $h_1$, that is

$$\hat{\lambda}_i = \langle A_i, h_1 h_1^T \rangle$$

$\hat{\lambda}_i$ calculated this way can be thought of as 'oracle' estimates, since it assumes $h_1$ is known. Not surprisingly, the two boxplots look similar since $||\hat{h}_1^m - h_1||$ is small for large $m$.

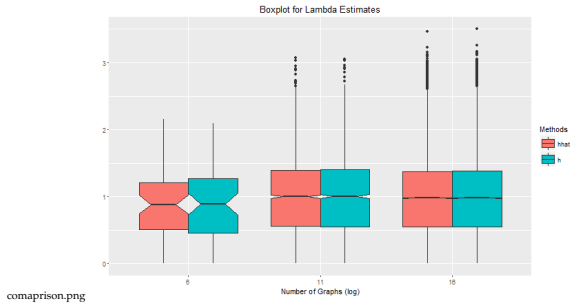


Fig. 2. Boxplot for Lambda Estimates

### 5.2 Simulation Experiment 2: Joint Embedding to Classify Graphs

In this experiment, we consider the inference task of classifying graphs. In general, $m$ pairs $\{(A_i, Y_i)\}_{i=1}^m$ are observed. Each pair consists of an adjacency matrix $A_i \in \{0,1\}^{n\times n}$ and a label $Y_i \in [K]$. Furthermore, all pairs are assumed to be independent and identically distributed according to an unknown distribution $\mathbb{F}_{A,Y}$, that is

$$(A_1, Y_1), (A_2, Y_2), ..., (A_m, Y_m) \overset{i.i.d.}{\sim} \mathbb{F}_{A,Y}$$

Our goal is to find a classification function $g : \{0,1\}^{n\times n} \to [K]$ which has small classification error $L_g = P(g(A) \neq Y)$.

Our experiment is set up to be a binary classification problem where $Y$ can only take value 0 or 1. We independently generate 200 graphs with 100 vertices from two MREG models each. Let $h_1$ and $h_2$ be two vectors in $\mathbb{R}^{100}$, and

$$h_1 = [\frac{1}{10}, \frac{1}{10}, ..., \frac{1}{10}]^T \text{ , and } h_2 = [-\frac{1}{10}, -\frac{1}{10}, ..., \frac{1}{10}]^T$$

Here, $h_2$ has $-\frac{1}{10}$ as its first 50 entries and $\frac{1}{10}$ as its last 50 entries. Condition on the value of $Y$, we generate graphs according to two MREG models, that is

$$A_i|Y_i = 0 \sim MREG(F_0, h_1, h_2) \text{ , here } F_0 = \mathbb{1}_{[25,5]}$$

$$A_i|Y_i = 1 \sim MREG(F_1, h_1, h_2) \text{ , here } F_1 = \mathbb{1}_{[22.5,2.5]}$$

In terms of SBM, this graph generation scheme is also equivalent to,

$$A_i|Y_i = 0 \sim SBM((1, ..., 1, 2, ..., 2), \begin{bmatrix} 0.3 & 0.2 \\ 0.2 & 0.3 \end{bmatrix})$$

$$A_i|Y_i = 1 \sim SBM((1, ..., 1, 2, ..., 2), \begin{bmatrix} 0.25 & 0.2 \\ 0.2 & 0.25 \end{bmatrix})$$

To classify graphs, we first apply joint embedding algorithm to 200 graphs. The embeddings are shown in the figure below. We can see two classes are clearly separated after being jointly embedded. Then, a 1-nearest neighbor classifier is

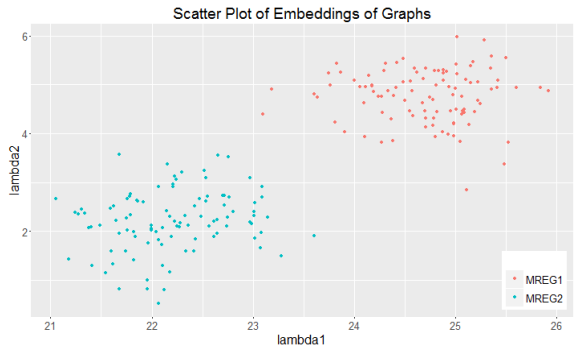


Fig. 3. $\Lambda_i$ of jointly embedded graphs

constructed based on loadings $\hat{\Lambda}$ by treating ith row as a feature vector for graph $i$. The 1-NN classification error with 200 graphs is 0.

We compare classification performance using joint embedding and ASE. In paper ***, *** introduces a semi-parametric statistic on adjacency spectral embeddings of two graphs. It is used to test the hypothesis that whether a graph follows a SBM model or not and is shown to be consistent when the number of vertices goes to infinity. Here, we treat each semi-parametric statistic as distance between two graphs and also apply 1-NN to classify graphs. We let the number of graphs $m$ to increase from 4 to 200. For each value of $m$, we repeat simulations 20 times. The result is shown in the next figure. We see joint embedding takes the advantage of increasing sample size and dominates ASE at all values of $m$.

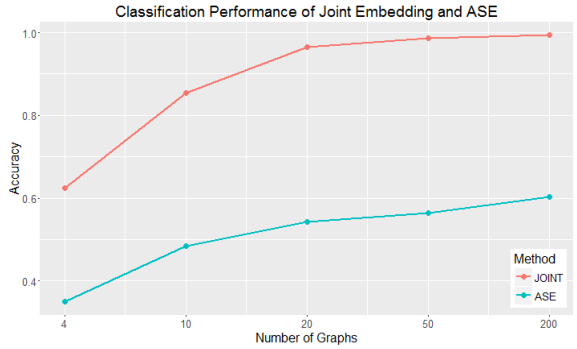


Fig. 4. Classification Performance of Joint Embedding and ASE

### 5.3 Real Data Experiment: CCI

## REFERENCES

[1]