Alma Mater Studiorum
Università di Bologna

DOTTORATO DI RICERCA IN
SCIENZE STATISTICHE

CICLO XXXII

Settore Concorsuale: 13/D1
Settore Scientifico Disciplinare: SECS-S/01

MULTIPLE GRAPH STRUCTURE LEARNING: A COMPARATIVE ANALYSIS

Presentata da: Violetta Zoffoli

Coordinatrice Dottorato:
Prof.ssa Alessandra Luati

Supervisore:
Prof. Alberto Roverato

Co-supervisore:
Prof. Stefano Lodi

Esame Finale Anno 2020

Abstract

In the context of analysing multivariate Gaussian distributions under different experimental conditions, recent studies have focused on retrieving the patterns of the conditional independences between pairs of variables for each condition. Given the representation of non-zero partial correlations as edges in a graph, we refer to this domain as *Multiple Graph Structure Learning*.

In application problems that assume some similarity between the graph structures, it has been suggested in the literature that learning the graphs jointly would be advantageous with respect to learning them separately. As an alternative, the graphs can be learnt directly from the difference of the concentration matrices.

The aim of this thesis is to understand the advantages and limitations of such learning methods. In order to do so, we compare these strategies by constructing a comprehensive and detailed simulation study analysis that includes different graph structures, different sample sizes, different dimensions and different levels of similarity between the experimental conditions. We evaluate the performance of the methods using the precision and recall indexes.

From the results of our simulation, it is evident that the underlying limitation of all the graph structure learning methods resides in the model selection, which corresponds to the choice of ℓ_1-norm penalty terms. This leads to the identification of graphs with highly variable densities, which hinders the method comparison.

We then impose that the models reproduce the true graph densities and we explore how different the resulting graphs are with respect to each learning method and simulation scenario.

Contents

Chapter 1

Introduction

Preface

In the recent decades, the research interest in biological, social as well as technological matters regarding the connectivity of items has increased vastly. According to the domain, a network of connections can be constructed in a variety of ways, for instance based on how individuals interact with each other, how genes in our DNA are expressed, how our brain activates, or even how web pages are linked to each other to create the World Wide Web.

Since this problem may be applicable to a multitude of real-world situations, the study of graphical models allows analysing the way interactions influence life as we know it. How are we connected as social beings? How do the genes encoding our body interact with each other? How does our brain activate under different conditions? These are all questions that may be answered by investigating the object that we are inclined to model through a graph.

Graphical models

A graph is a structure made of a set of vertices V and a set of edges E connecting pairs of the vertices. The reason why it a useful and versatile tool to many real-world applications is the ability to associate vertices to random variables and edges to their interactions. In this sense, we talk about a *graphical model* representing a statistical one. The focus for the statistical analysis is on how to recover the interactions between the variables and the graphical model comes in to associate these interactions to edges and has the advantage of portraying these pieces of information in a visually intuitive way.

Applications

For instance, we can think of a few examples which attribute to vertices and edges different meanings. In the domain of social networks, vertices are indi-

viduals and edges can be a degree of acquaintance, such as mutual friendship (e.g. Facebook) or one-sided following (e.g. Twitter). In the genetic field, we can think of vertices as positions (*loci*) on the DNA strands, in which expression levels for genes are recorded and the edges are to be intended as gene co-expression levels. Furthermore, in the analysis of neuronal activity, the vertices are associated to regions of the brain and if the interest is in how the regions co-activate among each other, the edges are retrieved by recording oxygenation levels in the blood.

According to the problem at hand, the type of graph associated to the statistical model changes. Graphs where edges include directionality, for instance, are suited for causal inference, whereas correlation is more easily-interpretable using graphs that do not portray a direction on the edges. Furthermore, the random variables associated to the vertices follow a specific probability distribution. The distribution itself determines the way in which a graph is estimated from the observed data.

High dimensional data

In the context of graphical modelling, not all applications face the same issues. Most of them, however, share the necessity of reducing the dimensionality of the data. For instance, in the analysis of DNA, a biologist may collect thousands of variables, but only for a few individuals. And even when the variables are less than the observed samples, there could still be too many interactions being shown by the model, which impacts negatively on the ability to interpret of the results of the graph estimation.

For this reason, penalized regression approaches have been largely employed in the last decades and various methods have been suggested, especially in the context of graphical models where the random variables are assumed to be Gaussian-distributed.

Multiple Graph Structure Learning

While graphical models have been popular for decades ([Lauritzen, 1996]) and applied to several research topics, a more specific interest has recently risen upon their employment in analyses regarding the differentiation of connectivity across multiple conditions. As an example, let us consider the case where we are interested in the interactions between six variables that are observed under two different conditions, assuming that among the two, the interactions are subject to modifications.

Figure 1.1 shows two graphs of six vertices and six edges each, three being common, three being different. In the literature, there have been several strategies in learning multiple graphs. The most immediate one is to consider each graph separately and apply a graph learning model to each.

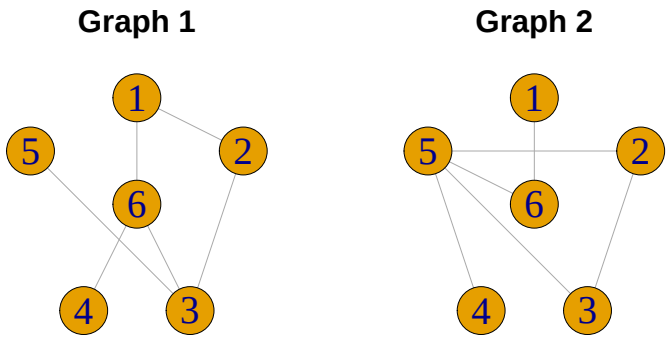


Figure 1.1: Two six-vertex graphs

Another strategy consists in learning the graphs jointly, by exploiting the commonality of their structures. This is shown in Figure 1.2: each of the graphs is learnt by borrowing strength from the common structure (red edges). Alternatively, one could focus directly on the graph of the different edges between the two conditions, as shown in Figure 1.3. This strategy does not retrieve the single graphs, but rather their differences and for this reason it is referred to as a *differential* graph structure learning procedure.

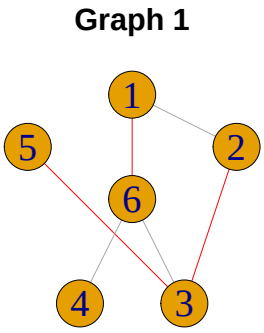
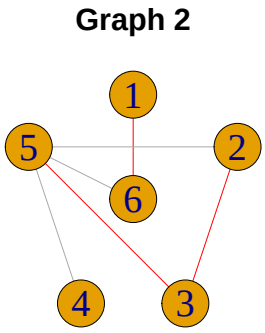


Figure 1.2: Graphs learnt jointly (common edges in red)



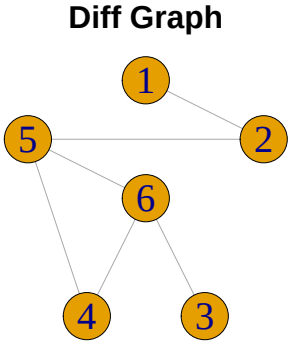


Figure 1.3: Graphs learnt differentially

To give some context to Multiple Graph Structure Learning, we may think of the previously-mentioned example of the analysis of DNA strands. To investigate the co-expression of genes for one set of individuals, a single graph is employed. However, there could be an interest in comparing the conditional dependence structure between genes of individuals that differ for some characteristic, such as the presence or absence of a tumor. In that case, we can expect most of the interactions between the genes to be common under the two conditions, but some could co-express differently. It could be

convenient, then, to either employ a joint learning technique or a differential one.

Comparison through simulation

Given the variety of estimation techniques for Multiple Graph Structure Learning models under the assumption of Gaussianity, this work aims at giving a useful comparison of such methods under different scenarios. Since many different applications may be characterized by varying graph structure, as well as varying sample size and dimension, we construct a simulation that comprehends a set of scenarios that may suit different datasets. The aim of this simulation analysis is to understand whether there are scenarios under which one graph structure learning method would be preferred to the others and how well they are able to learn the different features of multiple graphs with varying characteristics.

Chapter 2

Graphical Models

2.1 Fundamentals

A graph is a structure that we denote as a pair $\mathcal{G} = (V, E)$, where V is a finite set of vertices and E is a set of edges. The set of edges is a subset of $V \times V$, i.e. it is a set of pairs of vertices. For a pair of vertices $(i, j) \in V \times V$, we indicate with $i \sim_{\mathcal{G}} j$ the presence of an edge between i and j under graph $\mathcal{G}$, which we also refer to as an *adjacency* between the pair of vertices.

It should be specified that, depending on the directionality of the edges, there exists a variety of graph types such as directed, undirected or bidirected [Lauritzen, 1996], but for the purpose of this research only undirected graphs are relevant. We shall then refer to $i \sim_{\mathcal{G}} j$ solely as a non-directional connection between the two vertices. Furthermore, we do not consider the effect of vertices being related to themselves, thus eliminating loops.

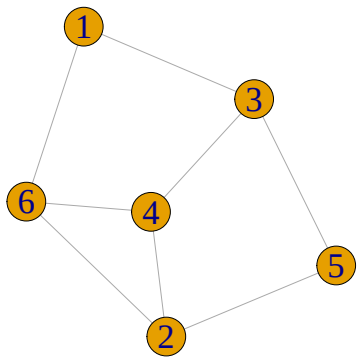


Figure 2.1: A simple undirected graph with six vertices

Figure 2.1 shows a simple undirected graph with six vertices and eight edges. We use the term *simple* to indicate the property of not having multiple edges and not having loops, i.e. no edges of the form $(i, i), i \in V$ [Lauritzen, 1996]. From here onwards, we will always assume graphs to be undirected and simple.

The interpretation of a graph is versatile to various statistical applications. In this thesis, a graph structure finds its statistical equivalent by considering the vertices as random variables which we are able to observe and analyse and edges as interactions. Since we do not consider directionality, we may interpret the presence of an edge as a non-zero partial correlation.

Depending on the application field and on the research problem itself, the variables may have different characteristics, mainly in terms of distributional properties. In other words, for a set of vertices V, we observe a vector $(X_1, \ldots, X_{|V|})$. We then refer to its distribution as

$$\mathbf{X}_V = (X_1, \ldots, X_{|V|}) \sim P \tag{2.1}$$

where P is some $|V|$-variate distribution function. Throughout this work, we denote vectors and matrices with bold letters.

At this point, we define how these variables entail a graphical model. The fundamental basis upon which a graphical model is built is the concept of conditional independence.

Conditional independence: Given three random variables X_i, X_j, X_k, $\{i, j, k\} \subseteq V$ that admit a joint distribution P, we say that X_i is conditionally independent of X_j given X_k if for any measurable set S_i in the sample space of X_i there exists a conditional probability $\mathbb{P}(S_i|X_j, X_k)$ which is only a function of X_k. We use the notation $X_i \perp\!\!\!\perp X_j|X_k$ to indicate conditional independence.

For discrete random variables, conditional independence can be expressed as:

$$\mathbb{P}(X_i = x_i, X_j = x_j|X_k = x_k) = \mathbb{P}(X_i = x_i|X_k = x_k)\mathbb{P}(X_j = x_j|X_k = x_k)$$

Where x_i, x_j, x_k are any realizations of X_i, X_j, X_k respectively and the equation holds for any value x_k for which $\mathbb{P}(X_k = x_k) > 0$.

If the variables admit joint density with respect to a product measure μ, we can write $X_i \perp\!\!\!\perp X_j|X_k$ if

$$f_{X_iX_j|X_k}(x_i, x_j|x_k) = f_{X_i|X_k}(x_i|x_k)f_{X_j|X_k}(x_j|x_k) \qquad \text{for all } x_i, x_j, x_k$$

The equation must hold almost surely with respect to P and for any realization x_k of X_k such that $f_{X_k}(x_k) > 0$ [Lauritzen, 1996].

We can extend the definition of conditional independence to random vectors. Let A, B, C be three subsets of V. For discrete random vectors, we say that $\mathbf{X}_A \perp\!\!\!\perp \mathbf{X}_B | \mathbf{X}_C$ if

$$\mathbb{P}(\mathbf{X}_A = \mathbf{x}_A, \mathbf{X}_B = \mathbf{x}_B | \mathbf{X}_C = \mathbf{x}_C) = \mathbb{P}(\mathbf{X}_A = \mathbf{x}_A | \mathbf{X}_C = \mathbf{x}_C)\mathbb{P}(\mathbf{X}_B = \mathbf{x}_B | \mathbf{X}_C = \mathbf{x}_C)$$

For any value of the realizations $\mathbf{x}_A, \mathbf{x}_B, \mathbf{x}_C$ of $\mathbf{X}_A, \mathbf{X}_B, \mathbf{X}_C$ respectively. For continuous random vectors, we have conditional independence if:

$$f_{\mathbf{X}_A \mathbf{X}_B | \mathbf{X}_C}(\mathbf{x}_A, \mathbf{x}_B | \mathbf{x}_C) = f_{\mathbf{X}_A | \mathbf{X}_C}(\mathbf{x}_A | \mathbf{x}_C) f_{\mathbf{X}_B | \mathbf{X}_C}(\mathbf{x}_B | \mathbf{x}_C) \qquad \text{for all } \mathbf{x}_A, \mathbf{x}_B, \mathbf{x}_C$$

The equation must hold almost surely with respect to P.

Moreover, in regard to three subsets A, B, C of V in a graph $\mathcal{G}$, we also introduce the concept of graph separation.

Graph separation: Let us define a *path* (a, b) of length r as a sequence $\gamma_0 = a, \ldots, \gamma_r = b$ such that $(\gamma_{i-1}, \gamma_i) \in E,\ i = 1, \ldots, r$. The subset C is an (a, b)-separator if all paths from a to b intersect C. C separates A from B if it is an (a, b)-separator for any $a \in A$, $b \in B$. We indicate graph separation as $A \overset{\mathcal{G}}{\perp} B | C$ [Lauritzen, 1996].

In Figure 2.1, if $A = \{2, 5\}$, $B = \{1\}$ and $C = \{3, 4, 6\}$, C separates A from B.

In addition to this, define the concept of boundary of a vertex.

Boundary: The boundary of a vertex i, indicated as $\mathrm{bd}(i)$, is the set of vertices adjacent to i. In Figure 2.1, for instance, $\mathrm{bd}(4) = \{2, 3, 6\}$.

To conclude, we give a definition for graph density.

Graph density: For a simple undirected graph, the density corresponds to the proportion of present edges over the total possible edges. Since loops are not considered and the direction is not influential, this can be computed for a graph $\mathcal{G} = (V, E)$ as

$$GD = \frac{|\{(i, j) : i \sim_{\mathcal{G}} j;\ i, j = 1, \ldots, V;\ i < j\}|}{|V|(|V| - 1)/2}$$

2.2 Markov properties

A graphical model is a representation, through a graph, of the set of conditional dependencies between the variables associated to V. The conditional independence between two variables $X_i, X_j,\ \forall\{i,j\} \subseteq V$, is represented by a missing edge in $\mathcal{G}$ only if the distribution of $\mathbf{X}_V$ satisfies the *pairwise Markov property* with respect to $\mathcal{G}$.

***Pairwise Markov property*:**
$\{i,j\} \subseteq V,\ i \nsim_{\mathcal{G}} j \implies X_i \perp\!\!\!\perp X_j | \mathbf{X}_{V\setminus\{i,j\}}$

The pairwise Markov property therefore describes the conditional independence relationship between pairs of non-adjacent vertices.

Other two Markov properties which we will refer to are the Local Markov Property and the Global Markov Property.

***Local Markov property*:** $\forall\ i \in V,\ X_i \perp\!\!\!\perp \mathbf{X}_{V\setminus \mathrm{cl}(i)} | \mathbf{X}_{\mathrm{bd}(i)}$
Where $\mathrm{cl}(i) = \{i\} \cup \mathrm{bd}(i)$.
This property thus concerns the conditional independence between a vertex and the set of its non-adjacent vertices.

***Global Markov property*:** For any disjoint subsets $A, B, C \subseteq V$ such that $A \overset{\mathcal{G}}{\perp} B | C$, it holds that $\mathbf{X}_A \perp\!\!\!\perp \mathbf{X}_B | \mathbf{X}_C$.

The global property then does not pertain a single vertex but rather the conditional independence between sets of vertices.

These properties are not equivalent in general for any graph and we remark that the global property implies the local, which implies the pairwise one, while the inverse is not always true. The converse implication, however, does hold for conditional independence when P has a positive density [Lauritzen, 1996].

2.3 Gaussian Graphical Models

When it comes to the distributional assumptions of these variables, we merely stated a joint distribution P in Equation (2.1). While there is no theoretical constraint on P, we focus on the case where the distribution is a multivariate Gaussian one. We do stress the relevance of Gaussian distributions for biological research topics, such as gene co-expression analysis on microarray data [Babu, 2004].

However, many problems do not assume gaussianity. Among these, we mention the analysis of co-expression data arising from RNA-sequencing. In this case, each variable $X_j, \forall j \in V$, is usually modelled according to a Poisson distribution or a negative-binomial one [Anders and Huber, 2010].

Once we established our distributional interest, we may explore the advantages of assuming Gaussian distribution for the variables making the vertices of a graph.

For notation purposes, we indicate that $|V| = p$. Moreover, we assume that the p-variate distribution is observed across a sample of n individuals. Therefore, we can store the information given by the sample in a $n \times p$ matrix, that we indicate with $\mathbf{x}$. Each row $\mathbf{x}^{(i)}$ is an independent observation of the p-variate Gaussian distribution, $i = 1, \dots, n$. Each column $\mathbf{x}_j$ is a set of n observations of the j-th random variable, $j = 1, \dots, p$.

We can then compute the sample covariance matrix as
$\mathbf{S} = \frac{1}{n} \sum_{i=1}^{n} (\mathbf{x}^{(i)} - \bar{\mathbf{x}})^T (\mathbf{x}^{(i)} - \bar{\mathbf{x}})$, being $\bar{\mathbf{x}}$ the sample mean vector.

A fundamental property of the Gaussian distribution that relates to graphical models is the correspondence between conditionally independent pairs of variables and zero elements of the inverse of the covariance matrix.

More precisely, let us consider a p-variate normal distribution for the i-th element of the sample, $i = 1, \dots, n$: $\mathbf{X}^{(i)} \sim \mathcal{N}_p(\boldsymbol{\mu}, \boldsymbol{\Sigma})$.

Let us consider two variables X_i and X_j, $\{i, j\} \subseteq V$, and define the bi-dimensional random vector $\mathbf{X}_{ij} = (X_i, X_j)$. We shall denote $V \backslash \{i, j\}$ as a and partition the mean vector and covariance matrix as:

$$\boldsymbol{\mu} = \begin{bmatrix} \boldsymbol{\mu}_{ij} \\ \boldsymbol{\mu}_a \end{bmatrix} \qquad \boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{\Sigma}_{ij,ij} & \boldsymbol{\Sigma}_{ij,a} \\ \boldsymbol{\Sigma}_{a,ij} & \boldsymbol{\Sigma}_{a,a} \end{bmatrix}$$

Where $\boldsymbol{\Sigma}_{ij,ij}$ is a 2×2 matrix and $\boldsymbol{\Sigma}_{a,a}$ is a $(p-2) \times (p-2)$ matrix.

The conditional distribution of $\mathbf{X}_{ij} | \mathbf{X}_a = \mathbf{x}_a$ is given by

$$\mathbf{X}_{ij} | \mathbf{X}_a = \mathbf{x}_a \sim \mathcal{N}_2(\boldsymbol{\mu}_{ij|a}, \boldsymbol{\Sigma}_{ij|a})$$

Where $\boldsymbol{\mu}_{ij|a} = \boldsymbol{\mu}_{ij} + \boldsymbol{\Sigma}_{ij,a} (\boldsymbol{\Sigma}_{a,a})^{-1} (\mathbf{x}_a - \boldsymbol{\mu}_a)$ and
$\boldsymbol{\Sigma}_{ij|a} = \boldsymbol{\Sigma}_{ij,ij} - \boldsymbol{\Sigma}_{ij,a} (\boldsymbol{\Sigma}_{a,a})^{-1} \boldsymbol{\Sigma}_{a,ij}$.

The covariance matrix of the conditional distribution is related to the concentration matrix of the joint distribution, $\boldsymbol{\Omega} = \boldsymbol{\Sigma}^{-1}$. Using the same partitioning,

$$\boldsymbol{\Omega} = \begin{bmatrix} \boldsymbol{\Omega}_{ij,ij} & \boldsymbol{\Omega}_{ij,a} \\ \boldsymbol{\Omega}_{a,ij} & \boldsymbol{\Omega}_{a,a} \end{bmatrix}$$

By using block-matrix inversion, it is easy to prove that
$\mathbf{\Omega}_{ij,ij} = [\mathbf{\Sigma}_{ij,ij} - \mathbf{\Sigma}_{ij,a}(\mathbf{\Sigma}_{a,a})^{-1}\mathbf{\Sigma}_{a,ij}]^{-1} = [\mathbf{\Sigma}_{ij|a}]^{-1}$.

Let us assume from now onwards, for simplicity, that the data is centred around zero: $\mathbf{X}^{(i)} \sim \mathcal{N}_p(\mathbf{0}_p, \mathbf{\Sigma})$. The concentration matrix of
$\mathbf{X}_{ij}|\mathbf{X}_a = \mathbf{x}_a$ is

$$\mathbf{\Omega}_{ij,ij} = \begin{bmatrix} \omega_{ii} & \omega_{ij} \\ \omega_{ji} & \omega_{jj} \end{bmatrix}$$

Keeping in mind that the matrix is symmetric, we have that:

$$COV(X_i, X_j|\mathbf{X}_a) = \{(\mathbf{\Omega}_{ij,ij})^{-1}\}_{i,j} = -\frac{\omega_{ij}}{\omega_{ii}\omega_{jj} - \omega_{ij}^2}$$

From which follows that $\omega_{ij} = 0 \iff X_i \perp\!\!\!\perp X_j|\mathbf{X}_{V\setminus\{i,j\}}$.

In this sense we can use a graph $\mathcal{G}(\mathbf{\Omega})$ based on the concentration matrix, also known as a *concentration graph* [Cox and Wermuth, 1996], to represent a multivariate Gaussian distribution, since $i \nsim_{\mathcal{G}} j \iff \omega_{ij} = 0$. This graph not only satisfies the pairwise Markov property, but also the global one, as the density of the normal distribution is positive.

2.4 Maximum Likelihood Estimate

When it comes to estimating the conditional independence patterns of data that are p-variate normally distributed, the maximum likelihood estimator for the covariance matrix shows convenient properties that facilitate retrieving such structure.

The maximum likelihood estimator $\hat{\mathbf{\Sigma}}$ for a covariance matrix $\mathbf{\Sigma}$, given a sample matrix $\mathbf{S}$ and with respect to a concentration graph $\mathcal{G}(\mathbf{\Omega})$, is such that:

$$\begin{cases} \hat{\sigma}_{ij} = s_{ij} & \text{if } i \sim_{\mathcal{G}} j \text{ or } i = j \quad (2.2) \\ \hat{\omega}_{ij} = 0 & \text{if } i \nsim_{\mathcal{G}} j \quad (2.3) \end{cases}$$

Where $\mathbf{\Omega} = \mathbf{\Sigma}^{-1}$ and consequently $\hat{\mathbf{\Omega}} = \hat{\mathbf{\Sigma}}^{-1}$.

The MLE also satisfies two properties [Dempster, 1972]:

- *Existence and uniqueness*: if there exists a positive definite symmetric matrix that satisfies (2.2) then there exists only one such matrix $\hat{\mathbf{\Sigma}}$ such that $\hat{\mathbf{\Sigma}}^{-1}$ also satisfies (2.3).

- *Maximum entropy*: among the matrices that satisfy (2.2), the choice $\hat{\mathbf{\Sigma}}$ has maximum entropy.

  While in general, for a density function $f(x)$ the entropy is defined as $-\int f(x)\log f(x)dx$, in this case, the entropy for the multivariate normal distribution results in $\log\det\mathbf{\Sigma}$.

Therefore, to find an estimate for the covariance matrix that also allows reconstructing a graph through its inverse, one should find its maximum likelihood estimate.

Given a number of observations n, let us express the likelihood as a function of the concentration matrix:

$$\begin{aligned}
L(\mathbf{\Omega}) &\propto \det(\mathbf{\Omega})^{\frac{n}{2}}\exp\left\{-\frac{1}{2}\sum_{i=1}^{n}\mathbf{x}^{(i)T}\mathbf{\Omega}\mathbf{x}^{(i)}\right\}\\
&= \det(\mathbf{\Omega})^{\frac{n}{2}}\exp\left\{-\frac{1}{2}\sum_{i=1}^{n}tr(\mathbf{\Omega}\mathbf{x}^{(i)}\mathbf{x}^{(i)T})\right\}\\
&= \det(\mathbf{\Omega})^{\frac{n}{2}}\exp\left\{-\frac{1}{2}tr\left(\mathbf{\Omega}\sum_{i=1}^{n}\mathbf{x}^{(i)}\mathbf{x}^{(i)T}\right)\right\}\\
&= \det(\mathbf{\Omega})^{\frac{n}{2}}\exp\left\{-\frac{n}{2}tr(\mathbf{\Omega}\mathbf{S})\right\} = \det(\mathbf{\Omega})^{\frac{n}{2}}\exp\left\{-\frac{n}{2}tr(\mathbf{S}\mathbf{\Omega})\right\}
\end{aligned}$$

By taking the logarithm, the function becomes:

$$\begin{aligned}
l(\mathbf{\Omega}) = \log L(\mathbf{\Omega}) &\propto \frac{n}{2}\log\det(\mathbf{\Omega}) - \frac{n}{2}tr(\mathbf{S}\mathbf{\Omega})\\
&\propto \log\det(\mathbf{\Omega}) - tr(\mathbf{S}\mathbf{\Omega})
\end{aligned}$$

Therefore, the concentration graph $\mathcal{G}(\mathbf{\Omega})$ can be retrieved by solving the maximization problem:

$$\hat{\mathbf{\Omega}} = \arg\max_{\mathbf{\Omega}}\left\{\log\det(\mathbf{\Omega}) - tr(\mathbf{S}\mathbf{\Omega})\right\} \tag{2.4}$$

For the saturated model, i.e. $|E| = p^2$, we may differentiate with respect to each element of $\mathbf{\Omega}$ and set the derivative to zero, which leads to:

$$\omega_{ij}^{-1} - s_{ij} = 0 \qquad \forall\ i,j = 1,\ldots,p \tag{2.5}$$

From which it is easy to conclude that for the saturated model, the maximum likelihood estimate is $\hat{\mathbf{\Sigma}} = \mathbf{S}$.

In general, when $|E| < p^2$, (2.5) does not hold for every (i,j). In order to calculate the maximum likelihood estimate satisfying (2.2) and (2.3), one

needs to perform a *matrix completion* problem. For instance, given a graph $\mathcal{G} = (V, E)$ such that $\{(1,3),(2,3),(2,p)\} \notin E$, the maximum likelihood estimate complies with the following structure:

$$\mathbf{\Sigma}^{\mathcal{G}} = \begin{bmatrix} s_{11} & s_{12} & * & \dots & s_{1p} \\ s_{21} & s_{22} & * & \dots & * \\ * & * & s_{33} & \dots & s_{3p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ s_{p1} & * & s_{p3} & \dots & s_{pp} \end{bmatrix}$$

The elements of $\mathbf{\Sigma}^{\mathcal{G}}$ corresponding to an edge in $\mathcal{G}$ or a diagonal element are obtained from their corresponding elements in $\mathbf{S}$. The remaining ones are not found directly in $\mathbf{S}$ and are indicated with $*$ to denote that their values are to be computed.

Therefore, when $\mathcal{G}$ is known and $\mathbf{S}$ is observed, the maximum likelihood estimation problem reduces to completing $\mathbf{\Sigma}^{\mathcal{G}}$, i.e. retrieving the unknown elements of $\hat{\mathbf{\Sigma}}$ in order to satisfy (2.3) as well as (2.2).

Several iterative algorithms have been developed to obtain such estimate, many of which follow a coordinate descent approach [Uhler, 2017]. Among the most widely-known, we mention the Iterative Proportional Scaling (IPS) algorithm. The MLE is obtained by iterating on the cliques of the graph (i.e. maximal complete subsets of vertices), using the reciprocal implications of the pairwise, local and global Markov properties for multivariate Gaussian data. After starting from an initial estimate $\hat{\mathbf{\Omega}} = \mathbf{I}_p$ whose entries are adjusted at each step of the algorithm to satisfy both (2.2) and (2.3), the algorithm converges to the MLE, as is proved in [Speed and Kiiveri, 1986].

On the other hand, one could perform the coordinate descent on the dual problem, starting from the sample covariance matrix and cycling through the entries of $\mathbf{S}$ that correspond to a missing edge and update their values in order to maximize the entropy and simultaneously satisfy (2.3) [Uhler, 2017].

2.5 Graphical Lasso

As discussed in Section 2.4, the maximum likelihood estimate is found by solving the maximization problem (2.4). In the saturated model, since $\hat{\mathbf{\Sigma}} = \mathbf{S}$, the estimate for the concentration matrix can be found as $\hat{\mathbf{\Omega}} = \mathbf{S}^{-1}$, assuming that the sample covariance matrix is invertible.

In general, for the maximum likelihood estimate to exist, the sample covariance matrix needs to be positive definite. In practice, this condition may not always hold, as whenever the dimension of data p is larger than the

sample size n the data matrix $\mathbf{x} \in \mathbb{R}^{n \times p}$ is almost surely not full rank. It is a common scenario in applications such as the genomic field, where thousands of genes are analysed simultaneously and only few observations are available due to the cost of genome sequencing. Furthermore, even in cases where $n > p$, the eigenvalues of $\mathbf{S}$ may be distorted to the point of making the matrix is ill-conditioned. While this does not compromise positive-definiteness, it does imply that the estimates tend to suffer from large estimation error [Hannart and Naveau, 2014].

One of the first approaches to retrieve the structure of high-dimensional graphs was suggested by [Meinshausen and Bühlmann, 2006]. The problem they suggest is *neighbourhood selection*: reconstructing for each node the set of its neighbours, i.e. its adjacent vertices, so that iteratively the whole graph is obtained. They suggest that neighbourhood selection is performed for each node through an ℓ_1- norm penalty. Using the fact that the missing edges correspond to elements of the concentration matrix being zero, they estimate for each node j the vector of coefficients $\hat{\boldsymbol{\theta}}_j$ that predict $\mathbf{X}_j$ with a penalty term ρ on the ℓ_1-norm, as in the following equation:

$$\hat{\boldsymbol{\theta}}_j = \arg\min_{\boldsymbol{\theta}} (n^{-1} \|\mathbf{X}_j - \mathbf{x}\boldsymbol{\theta}\|_2^2 + \rho \|\boldsymbol{\theta}\|_1)$$

The elements of $\hat{\boldsymbol{\theta}}_j$ being non-zero correspond to neighbours of vertex j. This is repeated for all vertices. It is clear then that neighbourhood selection operates via a Lasso-type [Tibshirani, 1996] variable selection strategy. Furthermore, they show that the neighbourhood selection estimate is consistent for high-dimensional sparse graphs [Meinshausen and Bühlmann, 2006].

The covariance selection problem was later on analysed by [Banerjee et al., 2008] and [Friedman et al., 2008], who proposed two iterative approaches that start from a modification of the sample covariance matrix to ensure positive definiteness and subsequently update the entries to satisfy (2.3), in a coordinate-descent fashion.

The objective of both the algorithms is to retrieve the concentration matrix (and consequently, graph) by maximizing the penalized log-likelihood function:

$$\hat{\boldsymbol{\Omega}} = \arg\max_{\boldsymbol{\Omega}} \left[\log\det(\boldsymbol{\Omega}) - tr(\mathbf{S}\boldsymbol{\Omega}) - \rho \|\boldsymbol{\Omega}\|_1 \right], \tag{2.6}$$

where $\rho > 0$ is the penalty term, which needs to be strictly greater than zero when $p > n$. While the two approaches by [Banerjee et al., 2008] and [Friedman et al., 2008] are structurally similar, the implementation of the latter is an iterative penalized regression problem and it has become

more widely-known as the *Graphical Lasso*.

Let us denote the iterative estimate for the Lasso as $\mathbf{W}$ and partition it as

$$\mathbf{W} = \begin{bmatrix} \mathbf{W}_{11} & \mathbf{w}_{12} \\ \mathbf{w}_{12}^T & w_{22} \end{bmatrix}$$

Where $\mathbf{W}_{11}$ is a $(p-1) \times (p-1)$ matrix, $\mathbf{w}_{12}$ is a $(p-1) \times 1$ vector and w_{22} is a scalar. Let us use the same partition also for the sample covariance matrix $\mathbf{S}$.

In the iterative algorithm, the objective is to estimate, for each vertex, the corresponding entries of $\mathbf{w}_{12}$. It is to be intended that the partition shifts according to the vertex. [Banerjee et al., 2008] find the solution as

$$\mathbf{w}_{12} = \arg\min_{\mathbf{y}} \{\mathbf{y}^T \mathbf{W}_{11}^{-1} \mathbf{y} : \|\mathbf{y} - \mathbf{s}_{12}\|_\infty \leq \rho\}$$

This is however equivalent to solving the problem

$$\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} \left\{ \frac{1}{2} \|\mathbf{W}_{11}^{1/2} \boldsymbol{\beta} - \mathbf{W}_{11}^{-1/2} \mathbf{s}_{12}\|_2^2 + \rho \|\boldsymbol{\beta}\|_1 \right\} \tag{2.7}$$

And then multiplying $\mathbf{w}_{12} = \mathbf{W}_{11} \hat{\boldsymbol{\beta}}$. Equation (2.7) is the penalized regression approach employed by [Friedman et al., 2008]. The algorithm for the Graphical Lasso goes as follows:

1. Initialize: $\mathbf{W} = \mathbf{S} + \rho \mathbf{I}_p$
2. For $j = 1, \ldots, p$:
   Solve (2.7) and fill in the corresponding row-column using $\mathbf{w}_{12} = \mathbf{W}_{11} \hat{\boldsymbol{\beta}}$
3. Repeat step 2 until convergence.

The first step of the algorithm ensures the positive definiteness of the modified sample covariance matrix, provided $\rho > 0$. By using block coordinate descent, the algorithm isolates one row-column at a time (diagonal excluded) and fits a Lasso regression using the remaining $(p-1) \times (p-1)$ block of the matrix. This maintains the positive definiteness of the estimate at any point of the algorithm. Since the algorithm was proved to be convergent [Banerjee et al., 2008] [Friedman et al., 2008], the estimate for $\boldsymbol{\Sigma}$ is obtained at the conclusion of step 3.

As a further remark on the Graphical Lasso, we discuss the relationship between the penalty term and the dimension of the data. As previously mentioned, ρ needs to be strictly greater than zero whenever dimension reduction is to be imposed. The larger this value is, the stronger the regularization operates on the concentration matrix, which as a consequence controls the sparsity of the estimate. It should be also noted that the level of regularization is not scale-invariant, in that the order of magnitude of the data influences the effect that the regularization parameter has on the shrinkage of the estimates. If we considered two sets of data that are equal other than a multiplying factor, the larger data would require a higher regularization parameter ρ than the smaller data to obtain the same estimate.

Moreover, it could be discussed whether the ℓ_1- norm could be substituted to other ones, such as ℓ_0 or ℓ_2. The ℓ_0 norm does not guarantee convexity of the objective function, while ℓ_2 does not have the parsimony property of ℓ_1. Therefore, it is the only viable solution that guarantees variable selection while maintaining the convexity needed for high-dimensional problems [Meinshausen and Bühlmann, 2006].

Chapter 3

Multiple Graph Structure Learning

3.1 Developments in the Early 2000s

The learning of concentration graphs discussed in Chapter 2 may be applied to a single population, but it may also be involved in the analysis of multiple populations to be compared. More precisely, we refer to multiple experimental conditions over sets of statistical units for which we wish to learn the p-variate conditional dependence structure.

For instance, in biology, a large quantity of studies focus on comparing a healthy condition and a diseased one. In this case, if the aim of an analysis were to learn the conditional independence structure of the p variables under the two conditions, one would need to learn both graphs. In a more general setting, this could be applied to any number of conditions for which the dependency structure is to be compared.

This research subject originates in the early 2000s, when in application fields such as the genomic one, researchers attempted to compare the distribution of p variables (e.g. genes) observed under two settings (e.g. healthy and tumor-affected subjects). The main concern lied in the dimension of the problem, having a large number of variables for a small sample size, due to the cost of retrieving the data.

Let us denote with H the number of conditions under analysis. For each one, the same p variables are recorded over a set of n_h subjects, $h = 1, \dots, H$. We can also refer to these conditions as *groups*, since the characteristic defining the experimental condition is shared among the subjects in each one.

Therefore, we are interested in the distributions of $\mathbf{X}_V^{(1)}, \dots, \mathbf{X}_V^{(H)}$, where the pairs $(\mathbf{X}_V^{(h)}, \mathbf{X}_V^{(h')})$, $h, h' = 1, \dots, H$ are mutually independent. As stated

in Section 2.3, we do focus on the case of Gaussian distributions, therefore we assume that

$$\mathbf{X}_V^{(h)} \sim \mathcal{N}_p(\boldsymbol{\mu}^{(h)}, \boldsymbol{\Sigma}^{(h)}) \qquad h = 1, \ldots, H \tag{3.1}$$

The simplest proposition to detect variables that behave differently in different conditions was to use a t-test for comparing means in a two-group scenario or ANOVA for multiple groups [Cui and Churchill, 2003]. One limitation of this procedure is that the t-test has a low power, especially in cases where the amount of variables greatly exceeds the sample size [Jeanmougin et al., 2010]. Furthermore, there is a need to correct for multiple testing using methods such as the Bonferroni or the Benjamini-Hochberg correction [Bogdan et al., 2015].

In the mid 2000s, other techniques developed as a complement to analysing the difference in means between varying experimental conditions. More specifically, there was a supplementary focus on the dependency between couples of random variables conditioned on the remaining ones.

A suggestion by [Choi et al., 2005] consists in constructing a graph based on the Pearson correlation coefficient for pairs of variables under two different conditions (healthy against tumor), where only the top 0.5% of correlations in absolute value are kept as significant. The graph for each experimental condition is constructed by introducing an edge between nodes for which the correlation coefficients are above the given threshold. This approach is followed similarly by [Reverter et al., 2006], who join the analysis of differences in means of the variables under two conditions and the connectivity of pairs of variables based on the Pearson correlation coefficient. Another suggestion based on the correlation between variables is followed by [Hu et al., 2009], who perform a test to compare the Fisher-transformed vectors of the correlations between two conditions and develop a nonparametric test to detect differentially-correlated variables. On the other hand, for Gaussian-distributed data, another idea is given by [Schäfer and Strimmer, 2004], who test the partial correlation between pairs of variables, exploiting the properties of the maximum likelihood estimate, as in Section 2.4.

These suggestions consider the correlation between pairs of variables under different conditions and they are highly based on the threshold determining the magnitude of correlation to be considered significant. However, they do not aim at reconstructing the dependency structure for the whole set of variables, which is instead done in [Zhang et al., 2009] by formulating a linear model for the local dependency structure based on the dependence of pairs of variables conditioned on the remaining ones. The parameters for

the model are estimated through an ℓ_1 constrained regression. They detect statistically-significant topological changes by comparing the variability in prediction using the local dependency structure estimated for the two conditions.

Furthermore, there have been propositions to compare the concentration matrices between two conditions. For instance, [Massa et al., 2010] suggest constructing a likelihood ratio test for the hypothesis that the concentration matrices are equal among the two conditions, thus giving a global test for *differential connectivity*, i.e. whether the strength of connections between variables is different between experimental conditions. [Kiiveri, 2011] give a detailed procedure to test for differential connectivity by evaluating the nullity of the matrix of differences between two or more conditions.

In the case of Gaussian-distributed data, focusing on the pairs of correlated variables might be less informative than reconstructing the whole network based on the concentration matrix. Furthermore, their effectiveness highly depends on the magnitude of the correlations, as well as the dimension of the data and the sample size, since many of these approaches are based on fixing thresholds on the correlations to detect differentially connected pairs of variables. However, test statistics for differential connectivity based on the nullity of the whole concentration matrices may not be useful, as it would not give information on which elements of the matrix are leading to differential connectivity.

For this reason, we remark the development in the early 2010s of the idea that identifying the different patterns of conditional dependence under two or more experimental conditions could be done by learning their underlying graph structures. We refer to this statistical problem as ***Multiple Graph Structure Learning*** (MGSL). Following the development of the Graphical Lasso algorithm [Friedman et al., 2008], this was consequently done by obtaining the estimate for the concentration matrices and thus identifying the graphs. In the case where there are multiple conditions to be compared, researchers have approached the issue by exploiting the nature of the problem, either by learning the graphs jointly or by learning directly the difference between conditions.

3.2 Separate Graph Structure Learning

Let us assume that we observe p variables over a set of H experimental conditions. In each one, we deal with a Gaussian distribution, as in Equation (3.1). As done for a single population, we can assume that the data is centred

around zero, so that for each condition, $\mathbf{X}_V^{(h)} \sim \mathcal{N}_p(\mathbf{0}_p, \mathbf{\Sigma}^{(h)}),\ h = 1, \dots, H$.

Clearly the objective of a MGSL procedure is to learn the H graphs and with Gaussian data, this amounts to estimating the concentration matrices $\mathbf{\Omega}^{(h)} = [\mathbf{\Sigma}^{(h)}]^{-1}$. The simplest way to do so is to estimate each of the matrices alone, as if they were H separate graph learning problems.

Let us assume that we observe the p variables n_h times for each condition. Let us then denote the $n_h \times p$ centred data matrix for the h-th condition as $\mathbf{x}^{(h)}$ and the corresponding sample covariance matrix as $\mathbf{S}^{(h)} = \frac{1}{n_h}\mathbf{x}^{(h)T}\mathbf{x}^{(h)}$.

Using the notions of the Graphical Lasso, as in Section 2.5, we can apply the method for learning multiple graphs by recovering an estimate for the concentration matrix associated to each graph.

If we wish to learn H graphs based on each $\mathbf{S}^{(h)}$, $h = 1, \dots, H$, we will have H separate Graphical Lasso problems of the form:

$$\hat{\mathbf{\Omega}}^{(h)} = \underset{\mathbf{\Omega}^{(h)}}{\arg\max} \left[\log\det(\mathbf{\Omega}^{(h)}) - tr(\mathbf{S}^{(h)}\mathbf{\Omega}^{(h)}) - \rho^{(h)}\|\mathbf{\Omega}^{(h)}\|_1\right] \qquad h = 1, \dots, H$$

The tuning parameters $\rho^{(h)}$, $h = 1, \dots, H$, are specific to each penalized likelihood maximization problem. Therefore, there is no regard for the possible similarity between graphs that may happen in studies where the variables are observed on different experimental conditions.

3.3 Joint Graph Structure Learning

In many application fields, there are situations in which there is a prior understanding of some degree of similarity between the graphs. For instance, if a phenomenon is observed under a healthy-patient condition against a diseased-patient one, it is reasonable to assume that many of the conditional dependencies between the observed variables do not vary. Such is the case for gene expression data, as it is assumed that most gene interactions are similar across human beings and only a few vary under different experimental conditions [Guo et al., 2011]. Another example would be to consider the brain activity between two patients: while it is likely that two individuals do not share completely the same connections, it is on the other hand unlikely that their brains function in a substantially different manner.

To overcome the limit of fitting separate graphical models that do not consider any degree of similarity between the conditions, an idea is to operate a joint fitting. We indicate this approach as *Joint Graph Structure Learning*

(JGSL). We consider two methods to estimate multiple graphs jointly. We refer to the first as Reparametrization Joint Graphical Lasso, proposed by [Guo et al., 2011]. The second one entails two procedures, the Fused Joint Graphical Lasso and the Group Joint Graphical Lasso, both introduced by [Danaher et al., 2014].

We remark that many applications compare the structure of two populations, due to the multitude of studies that aim at learning the difference between healthy and diseased patients. However, the methods explained in Sections 3.3.1 and 3.3.2 can be applied to estimate the graphs for any number groups.

3.3.1 Reparametrization Joint Graphical Lasso

One of the first approaches that employed joint graph structure learning for the model in (3.1) via the Graphical Lasso was suggested by [Guo et al., 2011]. Their motivation for the joint learning of the graphs is that it is able to exploit the common nature between them and obtain better estimates of the graph structures by using a penalty that controls the sparsity in the common structure and in the differentiated part.

Formally, they parametrize the off-diagonal elements of the concentration matrices as $\omega_{ij}^{(h)} = \theta_{ij}\gamma_{ij}^{(h)},\ \forall\ i \neq j,\ h = 1, \ldots, H$. The common part is represented by θ_{ij}, whereas $\gamma_{ij}^{(h)}$ may take on different values for each graph, $h = 1, \ldots, H$.

By allowing this factorization, they define a log-likelihood minimization criterion for estimating the concentration matrices:

$$\min_{\boldsymbol{\Theta}, (\boldsymbol{\Gamma}^{(h)})_{h=1}^{H}} \left\{ \sum_{h=1}^{H} \frac{n_h}{2} \left[tr(\mathbf{S}^{(h)}\boldsymbol{\Omega}^{(h)}) - \log(\det(\boldsymbol{\Omega}^{(h)})) \right] + \eta_1 \sum_{i \neq j} \theta_{ij} + \eta_2 \sum_{i \neq j} \sum_{h=1}^{H} |\gamma_{ij}^{(h)}| \right\} \tag{3.2}$$

In this setting, the sparsity is controlled by η_1 for the common components and by η_2 for the different ones. Note that they operate hierarchically, in that elements shrunk to zero by η_1 will be zero in all the graphs and those that are not can still be shrunk to zero by η_2.

The authors then restrain from the computational cost of tuning two parameters and reformulate (3.2) by inducing a penalty $\eta = \eta_1\eta_2$ only on differentiated elements.

$$\min_{\boldsymbol{\Theta}, (\boldsymbol{\Gamma}^{(h)})_{h=1}^{H}} \left\{ \sum_{h=1}^{H} \frac{n_h}{2} \left[tr(\mathbf{S}^{(h)}\boldsymbol{\Omega}^{(h)}) - \log(\det(\boldsymbol{\Omega}^{(h)})) \right] + \sum_{i \neq j} \theta_{ij} + \eta \sum_{i \neq j} \sum_{h=1}^{H} |\gamma_{ij}^{(h)}| \right\} \tag{3.3}$$

For computational advantages, they search for the local minimizer

$$\min_{(\mathbf{\Omega}^{(h)})_{h=1}^{H}} \left\{ \sum_{h=1}^{H} \frac{n_h}{2} \left[tr(\mathbf{S}^{(h)}\mathbf{\Omega}^{(h)}) - \log(\det(\mathbf{\Omega}^{(h)})) \right] + \rho \sum_{i \neq j} \left(\sum_{h=1}^{H} |\omega_{ij}| \right)^{\frac{1}{2}} \right\}, \tag{3.4}$$

where $\rho = 2\sqrt{\eta}$. Finally, they solve the local minimization problem by approximating (3.4) in an iterative form and decomposing it in H optimization problems. At each step t, to estimate the h-th concentration matrix for step $t+1$, the minimization problem is:

$$(\mathbf{\Omega}^{(h)})^{(t+1)} = \arg\min_{\mathbf{\Omega}^{(h)}} \left\{ \sum_{h=1}^{H} \frac{n_h}{2} \left[tr(\mathbf{S}^{(h)}\mathbf{\Omega}^{(h)}) - \log(\det(\mathbf{\Omega}^{(h)})) \right] + \rho \sum_{i \neq j} \tau_{ij}^{(t)} |\omega_{ij}^{(h)}| \right\}, \tag{3.5}$$

where $\tau_{ij}^{(t)} = \{\sum_{h=1}^{H} |(\omega_{ij}^{(h)})^{(t)}|\}^{-\frac{1}{2}}$. The penalty ρ is therefore multiplied, at each step, by an element-wise penalty based on the entries $(\omega_{ij}^{(h)})^{(t)}$ of the concentration matrix at the previous step. Since these can be very small numbers, their minimum value is set to 10^{-10} for numerical stability.

When it comes to the implementation of this method, the procedure is structured as an iterative Graphical Lasso with updated penalizations, as in Algorithm 1.

Algorithm 1: Reparametrization Joint Graphical Lasso

Initialize: $(\hat{\mathbf{\Sigma}}^{(h)})^{(0)} = \mathbf{S}^{(h)} + \nu\mathbf{I}$, $(\hat{\mathbf{\Omega}}^{(h)})^{(0)} = [(\hat{\mathbf{\Sigma}}^{(h)})^{(0)}]^{-1}$, $\forall\ h$

$\tau_{ij}^{(0)} = \{\sum_{h=1}^{H} |(\hat{\omega}_{ij}^{(h)})^{(0)}|\}^{-\frac{1}{2}}$, $\quad \mathbf{T}^{(0)} = [\tau_{ij}^{(0)}]_{i,j=1,\dots,p}$

while not convergent **do**

Obtain Graphical Lasso estimate $(\hat{\mathbf{\Omega}}^{(h)})^{(t)}$ following (3.5)

with penalty $\rho\mathbf{T}^{(t-1)}$,

Update: $\tau_{ij}^{(t)} = \{\sum_{h=1}^{H} |(\hat{\omega}_{ij}^{(h)})^{(t)}|\}^{-\frac{1}{2}}$, $\quad \mathbf{T}^{(t)} = [\tau_{ij}^{(t)}]_{i,j=1,\dots,p}$

end while

Convergence is achieved when $\frac{\sum_h \|(\hat{\mathbf{\Omega}}^{(h)})^{(t)} - (\hat{\mathbf{\Omega}}^{(h)})^{(t-1)}\|_1}{\sum_h \|(\hat{\mathbf{\Omega}}^{(h)})^{(t-1)}\|_1} < 10^{-5}$.
The initialization $(\hat{\mathbf{\Sigma}}^{(h)})^{(0)} = \mathbf{S}^{(h)} + \nu\mathbf{I}$ is made, for some $\nu > 0$, to ensure positive definiteness at the beginning of the algorithm. At the conclusion of the algorithm, H estimates for the H concentration matrices are obtained.

In order to determine which level of regularization ρ is more suited to the

data, the authors suggest two model selection criteria. The first chooses the parameter ρ that minimizes the Bayes Information Criterion:

$$BIC_\rho = \sum_{h=1}^{H} \left\{ \frac{n_h}{2} \left[tr(\mathbf{S}^{(h)} \hat{\mathbf{\Omega}}_\rho^{(h)}) - \log(\det(\hat{\mathbf{\Omega}}_\rho^{(h)})) \right] + log(n_h) df_\rho^{(h)} \right\}, \quad (3.6)$$

where n_h is the sample size for the h-th group, $\hat{\mathbf{\Omega}}_\rho^{(h)}$ is the estimate for the h-th concentration matrix with penalty ρ and $df_\rho^{(h)}$ are the degrees of freedom for the h-th graph with penalty ρ. The degrees of freedom are to be understood as the number of non-zero elements in the estimated concentration matrix. More formally, $df_\rho^{(h)} = |\{(i,j) : i < j, [\hat{\mathbf{\Omega}}_\rho^{(h)}]_{ij} \neq 0\}|$.

The second method they suggest to select the model is to use crossvalidation, by splitting the dataset into D segments of equal size. For the d-th segment, they recover the sample covariance matrix for each group, $\mathbf{S}^{(h,d)}$. The concentration matrix is estimated on the remaining data using tuning parameter ρ, which is denoted as $\hat{\mathbf{\Omega}}_\rho^{(h,-d)}$. The optimal parameter is chosen to minimize the average predictive negative log-likelihood:

$$CV_\rho = \sum_{d=1}^{D} \sum_{h=1}^{H} \left[tr(\mathbf{S}^{(h,d)} \hat{\mathbf{\Omega}}_\rho^{(h,-d)}) - \log(\det(\hat{\mathbf{\Omega}}_\rho^{(h,-d)})) \right] \quad (3.7)$$

The advantage in choosing the crossvalidation selection method comes from its accuracy with respect to the BIC. On the other hand, the latter is computationally more intensive.

3.3.2 Fused and Group Joint Graphical Lasso

The method proposed by [Danaher et al., 2014] for the distribution model in (3.1) is based on finding the estimates for the concentration matrices $\{\mathbf{\Omega}\} = \{\mathbf{\Omega}^{(1)}, \ldots, \mathbf{\Omega}^{(H)}\}$ by maximizing a penalized log-likelihood function. This method is based on similar premises as the one by [Guo et al., 2011], as shown in Section 3.3.1. The regularization is required especially in cases where the number of variables exceeds the number of observations, but also when the number of observations is not large enough to have stable estimates. Moreover, this method is built to possibly improve the performance of fitting H graphical Lasso models separately, by exploiting the similarity between the H conditions.

The estimates for the concentration matrices $\{\mathbf{\Omega}\}$ are found by finding

the maximum of the penalized log-likelihood:

$$\{\hat{\mathbf{\Omega}}\} = \arg\max_{\{\mathbf{\Omega}\}} \left\{ \sum_{h=1}^{H} \frac{n_h}{2} \left[\log(\det(\mathbf{\Omega}^{(h)})) - tr(\hat{\mathbf{\Sigma}}^{(h)}\mathbf{\Omega}^{(h)}) \right] - P(\{\mathbf{\Omega}\}) \right\} \quad (3.8)$$

Where $P(\{\mathbf{\Omega}\})$ denotes a convex penalty function, so that the maximization problem is strictly concave with respect to $\{\mathbf{\Omega}\}$. The authors choose a penalty function of the form $P(\{\mathbf{\Omega}\}) = \rho_1 \sum_{h=1}^{H} \sum_{i \neq j} |\omega_{ij}^{(h)}| + \tilde{P}(\{\mathbf{\Omega}\})$. The first element of this function imposes a traditional ℓ_1-norm regularization on the elements of each of the concentration matrices and when $\tilde{P}(\{\mathbf{\Omega}\}) = 0$, this corresponds to performing H separate Graphical Lasso problems. They suggest two possibilities for $\tilde{P}(\{\mathbf{\Omega}\})$: a fused Lasso penalty [Tibshirani et al., 2005] and a group Lasso one [Yuan and Lin, 2006].

Fused Joint Graphical Lasso.
The Fused Joint Graphical Lasso (Fused JGL) is a method that aims at solving Equation (3.8) with penalty

$$P(\{\mathbf{\Omega}\}) = \rho_1 \sum_{h=1}^{H} \sum_{i \neq j} |\omega_{ij}^{(h)}| + \rho_2 \sum_{h<h'} \sum_{i,j} |\omega_{ij}^{(h)} - \omega_{ij}^{(h')}| \quad (3.9)$$

In the Fused Graphical Lasso, ρ_1 controls the general sparsity of the estimates for the H concentration matrices $\hat{\mathbf{\Omega}}^{(h)}$ and the parameter ρ_2 controls the similarity of edge values. When ρ_1 is large, the H matrices will be sparse and furthermore, when ρ_2 is large, many elements of the matrices will be equal across the groups.

Group Joint Graphical Lasso.
The Group Joint Graphical Lasso (Group JGL) is the solution to Equation (3.8) with penalty

$$P(\{\mathbf{\Omega}\}) = \rho_1 \sum_{h=1}^{H} \sum_{i \neq j} |\omega_{ij}^{(h)}| + \rho_2 \sum_{i \neq j} \sqrt{\sum_{h=1}^{H} \omega_{ij}^{(h)^2}} \quad (3.10)$$

Here again, ρ_1 controls the sparsity of the concentration matrices, whereas ρ_2 regulates the similarity of pattern across the H matrices. As ρ_1 increases, they become overall sparser and as ρ_2 increases, the elements of each matrix tend to be all either zero or non-zero. When $\rho_1 = 0$ and $\rho_2 > 0$, the H concentration matrices will all have the same sparsity structure. In other words, the Group JGL induces a similar sparsity pattern, whereas the Fused JGL induces similar *values* and therefore promotes a stronger similarity.

As for the method to select the regularization parameters, we remark that the characteristic of having two parameters to tune allows a large flexibility of

the model, but it implies a much heavier computational effort. This was the motivation to re-parametrize $\eta = \eta_1\eta_2$ in Section 3.3.1. In [Danaher et al., 2014], the authors select the optimal model in terms of Akaike's Information Criterion:

$$AIC_{(\rho_1,\rho_2)} = \sum_{h=1}^{H} \left\{ \frac{n_h}{2} \left[tr(\mathbf{S}^{(h)}\hat{\mathbf{\Omega}}^{(h)}_{(\rho_1,\rho_2)}) - \log\det(\hat{\mathbf{\Omega}}^{(h)}_{(\rho_1,\rho_2)}) \right] + 2df^{(h)}_{(\rho_1,\rho_2)} \right\}, \tag{3.11}$$

where $\hat{\mathbf{\Omega}}^{(h)}_{(\rho_1,\rho_2)}$ is the h-th matrix estimated with penalties (ρ_1, ρ_2) and the degrees of freedom $df^{(h)}_{(\rho_1,\rho_2)} = |\{(i,j) : i < j, [\hat{\mathbf{\Omega}}^{(h)}_{(\rho_1,\rho_2)}]_{ij} \neq 0\}|$ correspond to the number of non-zero values in such estimated concentration matrices.

Selecting over plausible ranges for ρ_1 and ρ_2 would imply a grid search which quickly becomes infeasible for high-dimensional datasets. The authors suggest performing a dense search over ρ_1 with fixed low ρ_2 and then a quick search over ρ_2 at the selected value for ρ_1.

3.4 Differential Graph Structure Learning

A different approach to graph structure learning problems based on the model in (3.1) consists in estimating directly the graphs of differences between the two (or more) populations to be compared. We refer to this as *Differential Graph Structure Learning* (DGSL).

For instance, given two concentration matrices $\mathbf{\Omega}^{(1)}$ and $\mathbf{\Omega}^{(2)}$ we can construct the conditional independence graphs associated to them as $\mathcal{G}_1 = (V, E_1)$ and $\mathcal{G}_2 = (V, E_2)$. We clearly have $E_1 = \{(i,j) : \omega^{(1)}_{ij} \neq 0; i, j \in V; i < j\}$ and similarly for E_2. We can then construct a graph of differences as $\mathcal{G}_d = (V, E_d)$, where $E_d = \{(i,j) : \omega^{(1)}_{ij} - \omega^{(2)}_{ij} \neq 0; i, j \in V; i < j\}$.

The main advantage of this kind of methods is in the lack of sparsity assumption for the individual concentration matrices associated to each of the graphs. Since only the concentration matrix of differences is estimated, the sparsity to retrieve the estimates is needed only on the graph of differences.

The key point when choosing between a joint learning method and a differential one is the possibility to recover estimates for each of the population graphs that is given by the former, as opposed to the single estimate that is produced by the latter.

We show a DGSL method proposed by [Zhao et al., 2014], that essentially performs an ℓ_1-norm regularization on the concentration matrix of differences between two populations. Another suggestion by [Liu et al., 2014] consists

in estimating the ratio between the probability density functions of the two groups. The latter is applicable to any distribution of the random variables to be analysed, but we do focus only on the former, since this work mainly deals with Gaussian-distributed data.

3.4.1 Direct Estimation of Graphs of Differences

This method proposed by [Zhao et al., 2014] relies on the estimation of the differences between the concentration matrices of multiple populations. Even though a multi-group expansion is mentioned, the method is built primarily for comparing two groups.

Therefore, we focus on two concentration matrices $\mathbf{\Omega}^{(1)}$ and $\mathbf{\Omega}^{(2)}$. Let us define $\mathbf{\Delta}_0 = \mathbf{\Omega}^{(1)} - \mathbf{\Omega}^{(2)}$, which will be the object of estimation in this procedure.

Let us also define a few operators on matrices. Given a $n \times p$ matrix $\mathbf{A}$, we indicate as $\text{vec}(\mathbf{A})$ the vectorization, i.e. the operation of stacking the columns of $\mathbf{A}$ to obtain a np-dimensional vector. Similarly, for a symmetric $p \times p$ matrix $\mathbf{B}$ we define the half-vectorization operator $\text{vech}(\mathbf{B})$ that stacks the columns of the lower-triangular part of $\mathbf{B}$ (diagonal included) onto a $p(p+1)/2$-dimensional vector.

Furthermore, for a pair of matrices $\mathbf{C}, \mathbf{D}$, we define $\mathbf{C} \otimes \mathbf{D}$ as the Kronecker product of the two matrices. Finally, we indicate $|\mathbf{A}|_\infty$ as the sup-norm of any matrix $\mathbf{A}$.

In order to find the estimate for $\mathbf{\Delta}_0$, given the sample covariance matrices, the objective is to find a solution $\hat{\mathbf{\Delta}}$ to the following equation:

$$\mathbf{S}^{(1)}\mathbf{\Delta}\mathbf{S}^{(2)} - (\mathbf{S}^{(1)} - \mathbf{S}^{(2)}) = 0 \qquad \text{for } \mathbf{\Delta}$$

Notice that the equation always holds for $\mathbf{\Delta} = \mathbf{\Delta}_0$. However, we incur in the same existence issues when $n < p$, since there would be infinite solutions to this problem. Therefore, an ℓ_1-norm constrained minimization is proposed. A formulation as a linear problem is also operated so that the minimization becomes

$$\hat{\mathbf{\Delta}} = \arg\min |\mathbf{\Delta}|_1 \quad \text{s.t. } |(\mathbf{S}^{(2)} \otimes \mathbf{S}^{(1)})\text{vec}(\mathbf{\Delta}) - \text{vec}(\mathbf{S}^{(1)} - \mathbf{S}^{(2)})|_\infty \leq \rho_n \quad (3.12)$$

Note that the kind of regularization in Equation (3.12) imposes sparsity directly on $\mathbf{\Delta}_0$ rather than on $\mathbf{\Omega}^{(1)}$ and $\mathbf{\Omega}^{(2)}$ individually.

Furthermore, we define a $p^2 \times p(p+1)/2$ duplication matrix $\mathcal{D}$ and a vector $\boldsymbol{\beta} = \text{vech}(\mathbf{\Delta})$. The authors suggest exploiting the equality $(\mathbf{S}^{(2)} \otimes \mathbf{S}^{(1)})\text{vec}(\mathbf{\Delta}) = (\mathbf{S}^{(2)} \otimes \mathbf{S}^{(1)})\mathcal{D}\boldsymbol{\beta}$. Equation (3.12) is then reformulated as:

$$\hat{\boldsymbol{\beta}} = \arg\min |\boldsymbol{\beta}|_1 \quad \text{s.t. } |\mathcal{D}^T(\mathbf{S}^{(2)} \otimes \mathbf{S}^{(1)})\mathcal{D}\boldsymbol{\beta} - \mathcal{D}^T\text{vec}(\mathbf{S}^{(1)} - \mathbf{S}^{(2)})|_\infty \leq \rho_n$$

Finally, they modify the regularization strategy by dividing two cases, for the off-diagonal elements of the matrices and the diagonal ones, the latter having halved constraints:

$$\hat{\boldsymbol{\beta}} = \arg\min |\boldsymbol{\beta}|_1, \quad \text{s.t.} \begin{cases} |\mathcal{D}^T \mathbf{S}^{(2)} \otimes \mathbf{S}^{(1)} \mathcal{D}\boldsymbol{\beta} - \mathcal{D}^T \text{vec}(\mathbf{S}^{(1)} - \mathbf{S}^{(2)})|_{O\infty} \leq \rho_n \\ |\mathcal{D}^T \mathbf{S}^{(2)} \otimes \mathbf{S}^{(1)} \mathcal{D}\boldsymbol{\beta} - \mathcal{D}^T \text{vec}(\mathbf{S}^{(1)} - \mathbf{S}^{(2)})|_{D\infty} \leq \frac{\rho_n}{2} \end{cases}$$

where the norms $|\cdot|_{O\infty}$ and $|\cdot|_{D\infty}$ denote the sup-norm of the entries corresponding to off-diagonal and diagonal elements respectively.

It should be noted that this procedure is currently developed and implemented for DGSL analyses involving only two groups, though the expansion two multiple group comparison is considered by [Zhao et al., 2014]. While it could be possible to perform all the pairwise comparisons, it could be time consuming as the number of variables p increases. Therefore, they suggest comparing each of the populations to a common one built on a pooled covariance matrix. They define the pooled sample covariance matrix $\mathbf{S}^{(P)} = \sum_{h=1}^{H} w_h \mathbf{S}^{(h)}$ as a weighted average of the H sample covariance matrices with weights $w_h,\ h = 1, \ldots, H$.

The problem to be solved for each group $h = 1, \ldots, H$ becomes

$$\hat{\boldsymbol{\Delta}}^{(h)} = \arg\min |\boldsymbol{\Delta}|_1 \quad \text{s.t. } |\mathbf{S}^{(h)} \boldsymbol{\Delta} \mathbf{S}^{(P)} - \mathbf{S}^{(h)} - \mathbf{S}^{(P)}|_\infty \leq \rho_n$$

The graph of differences between two groups h and h' is estimated as $\hat{\boldsymbol{\Delta}}^{(h)} - \hat{\boldsymbol{\Delta}}^{(h')}$.

3.5 Comparison of methods

The methods described in Sections 3.2-3.4 are built with theoretically different purposes, which may result in different performances when applied to data.

First of all, the simplest approach to learning H graphs consists in fitting H separate Graphical Lasso models. The advantage in doing so resides primarily in the lower computational complexity of the Graphical Lasso algorithm, which treats every graph as a separate one. Another possible advantage is the ability to choose a different regularization parameter for each of the graphs, which would be beneficial to learn graphs considerably different from one another. However, in the case of highly similar structures, it may not be convenient to use this approach.

When it comes to the JGSL methods, the main distinction lies in the choice of reformulating the optimization problem either by using a single

tuning parameter ([Guo et al., 2011]) or a double penalty ([Danaher et al., 2014]). In the former, the reparametrization reduces the computational cost of tuning two parameters: while this is certainly advantageous, [Danaher et al., 2014] argue that it impacts negatively on the flexibility of the model. Furthermore, the reformulation (3.4) that the algorithm by [Guo et al., 2011] is built on uses a non-convex penalty function. This might make the computation slower and the iterative approximation is needed to avoid the optimization of a non-convex function. On the other hand, the penalties in (3.9) and (3.10) avoid this issue, even though the disadvantage of tuning two parameters is not negligible in terms of computational cost.

The main difference between joint graph structure learning methods and differential ones is the object of estimation. While JSGL methods estimate the concentration matrices associated to each of the graphs, the Direct estimation method only estimates the graph of differences. This may be a disadvantage if the aim of the analysis is to reconstruct the graphs for all the experimental conditions, but it may be advantageous if the focus is merely on the distinction between them. The benefit of learning the graph of differences resides in the need of sparsity only for the difference between the concentration matrices, rather than for the individual ones.

As a matter of fact, [Zhao et al., 2014] prove that their estimator can correctly identify the structure of the graph of differences if the matrix of differences have constant sparsity (which is a reasonable assumption when the individual graphs have a similar structure) and the covariances between the variables are not too high. They are weaker restrictions than assuming that the individual concentration matrices are sparse. In this regard, [Guo et al., 2011] prove that their estimators for the individual concentration matrices are consistent if each of the $\mathbf{\Omega}^{(h)}, h = 1, \ldots, H$, is well conditioned and their non-zero elements are bounded away from zero. Given additional bounds on the regularization parameter ρ, they also prove *sparsistency*, i.e. the ability to recover the set of zero elements of each of the concentration matrices.

Given the different features of each of these methods, it remains to understand which of these approaches is more suitable to a given MGSL problem. What we could expect is that, theoretically, the Separate Graphical Lasso would be a better fit when the graph structures differ more significantly. On the other hand, we would expect the JGSL and especially the DGSL methods to be better performing when the structures are more similar. In regard to the joint procedures, we wish to understand whether the choice of reparametrizing using one tuning parameter is effectively a limiting approach with respect to the Joint and Group JGL or if, on the other hand, the cost of two tuning parameters has a significant impact on the computation of the

estimates. As the similarity between the graphs increases, we could expect the method by [Zhao et al., 2014] to be outperforming the remaining ones, although its objective would be to reconstruct only the graph of differences, rather than the individual ones.

It is certainly of interest to understand the variation in performance between these methods for different scenarios with respect to the dimension of the data. Theoretically, we could expect that the larger the data, the worse the methods are at reconstructing the graphs. In addition to this, we would like to explore the influence that the sample size has on the quality of the estimates for the concentration matrices, in terms of how well the methods are able to adjust for the tuning parameters as n decreases with respect to p and viceversa.

Furthermore, it would be useful to understand how the graph structures themselves influence the performance of each method, so that we could outline a set of application strategies that may suit a variety of data sets.

For this reason, we construct a comprehensive simulation that includes a variety of scenarios for the dimension of the data, the sample size, the degree of dissimilarity between conditions, the sparsity and the graph structure. This process, which is the main body of this thesis, is thoroughly explained in Chapter 4.

Chapter 4

Simulation Study Analysis

Given the variety of multiple graph structure learning methods, as discussed in Chapter 3, the aim of this Chapter is to construct a simulation that includes a collection of scenarios under which the methods are fitted to datasets. For each scenario, the ability of each method to perform model selection for the underlying graph is evaluated. The objective of this collective analysis is to determine whether one method is systematically the best performing or if certain conditions favour specific learning methods. The key element of this work is the multiplicity of settings to be adjusted in order to cover the diversity of real-data applications, with the intent of understanding under which conditions one learning method would be preferred to the others.

For this reason, we discuss the simulation settings in Section 4.1. We give an outline of the steps of this simulation, for each of these settings. Each step is discussed in more detail in the indicated Sections.

1. Generating different graph structures (Section 4.2);
2. Generating a common covariance matrix (Section 4.3);
3. Distinguishing the covariance matrix according to each condition (Section 4.3.1);
4. Generating the data (Section 4.4);
5. Applying each of the learning methods (Section 4.5) and selecting the model with respect to the regularization parameters (Section 4.6);
6. Establishing the performance measures to be evaluated on the methods (Section 4.7);

7. Evaluating the performance results of each method under the given settings (Section 4.8).

We specify that this simulation is built in `R` [R Core Team, 2018]. The functions that we implemented to accomplish each step of it are reported in the Appendix (Section B).

4.1 Simulation Settings

The set of simulations that will be run varies according to a set of parameters determining different scenarios.

We recall that for multiple graph structure learning, we deal with a set of experimental conditions - which we also refer to as *groups*- that are indexed by h, $h = 1, \ldots, H$. In this simulation, we set the number of groups to $H = 2$, due to the fact that the vast majority of this type of analyses aims at comparing the features of healthy and diseased subjects.

Furthermore, we will simulate different graph structures as will be discussed in Section 4.2 and we index them by g, $g = 1, \ldots, G$. In this simulation, we have $G = 4$ types of graphs.

In addition to the graph structures, the simulations vary according to the following parameters:

- p: the number of variables,
- n: the number of observations,
- δ: the dissimilarity degree, i.e. the portion of edges changing between the two groups.

We will perform simulations for two different values for p: first a small number of variables, $p = 30$, which allows performing visual exploratory analyses on the graphs, than a larger one, $p = 100$. The interest of the latter is to understand how the methods are able to deal with higher dimensions.

The second varying element is the sample size n. We begin by stating that, while we could allow for different group sizes n_h, we choose to have equal group sample sizes, i.e. $n_h = n$, for $h = 1, 2$. This is due to the focus of this analysis not being the case of imbalanced classes, which may happen for instance when the data available for healthy patients is much easier to retrieve than for diseased ones. We consider this a particular case that is not the primary objective of this work.

The focus is rather put on the relationship between the sample size and the dimension of the data. It is interesting to understand how the learning methods react to the number of observations being smaller than the number of variables ($n < p$), in contrast with the cases where $n > p$, but $n \approx p$, and $n >> p$. Using the two choices for dimensionality, we construct five combinations for p and n, which we order from best-case (small p and $n >> p$) to worst-case (large p and $n < p$):

- $p = 30,\ n = 200$
- $p = 30,\ n = 40$
- $p = 30,\ n = 20$
- $p = 100,\ n = 400$
- $p = 100,\ n = 60$

Finally, we discuss the choices for the dissimilarity degree δ.

Given a pair of graphs $\mathcal{G} = (V, E)$ and $\mathcal{G}_h = (V, E_h)$, we define the dissimilarity degree δ_h as:

$$\delta_h = \frac{|\{(i,j) : (i \sim_{\mathcal{G}} j \wedge i \nsim_{\mathcal{G}_h} j) \vee (i \nsim_{\mathcal{G}} j \wedge i \sim_{\mathcal{G}_h} j);\ i,j = 1,\dots,V;\ i \neq j\}|}{|E|}$$

As will be discussed in Section 4.3.1, we start from a common graph $\mathcal{G}$, which is then distinguished into two graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ with a dissimilarity degree $\delta_h, h = 1, 2$. However, we choose to keep the degree constant, i.e. $\delta_1 = \delta_2 = \delta$.

This parameter then corresponds to the portion of edges changing from present to absent, or viceversa, from a common structure in order to construct a group distinction. The higher δ is, the larger the dissimilarity between the two groups will be. We explore two possibilities: $\delta = 0.05$ and $\delta = 0.20$. The reason for these two choices arises from the consideration that some of the learning methods may work better for highly similar graphs, namely the direct estimation method as discussed in Section 3.4.1, and others for less similar ones, as discussed for the Separate Graphical Lasso in Section 3.2.

4.2 Graph Structures

The first part of the simulation consists in choosing a variety of graph structures for the underlying conditional independence pattern. The decision was

to select four types of structures, in order to understand whether the performance of the methods was influenced by the structure of the graph. The four structures are the random graph, the Markov-chain graph, the tree, and the scale-free graph.

In this Section, we report also some visual examples of the graph structures for $p = 30$.

4.2.1 Random Graph

This graph is based on the assumption that each edge has a probability π of existence. This is also referred to as the *Erdős-Rényi Model* [Erdős and Rényi, 1959].

We indicate as $G(p, \pi)$ the probability distribution corresponding to graphs such that each of the p nodes has on average $(p-1)\pi$ connected nodes. For a node j, we refer to its degree as the number of nodes adjacent to it. Formally, $deg(j) = |bd(j)|$. It can be seen that the degree for a node j follows a binomial distribution:

$$P(deg(j) = k) = \binom{p-1}{k} \pi^k (1-\pi)^{p-1-k}$$

It is straightforward that a graph following $G(p, \pi)$ has on average $\binom{p}{2}\pi$ edges. Therefore, the parameter π implicitly controls the density of the graph.

The graph is created in `R` using the `sample_gnp(p,pi)` function, available in the `igraph` package.

The example in Figure 4.1 with 30 nodes shows how the density parameter influences the structure of the graph.

In this simulation, we choose $\pi = 0.1$ as the probability of edge existence that regulates the density of the random graph. The reason for this choice lies in the relevance of sparse settings for most real data applications.

4.2.2 Markov-chain graph

This structure can be thought of as the representation of an autoregressive process of order one for a p-variate distribution. Let $\mathbf{X}_V \sim \mathcal{N}_p(\mathbf{0}_p, \mathbf{C})$. A Markov-chain graph structure is generated when $\mathbf{C}$ is such that its elements associated to pairs of variables (X_i, X_j) are:

$$c_{ij} = \exp\left(-\frac{1}{2}\sum_{k=min(i,j)}^{max(i,j)-1} u_k\right) \qquad \text{for } i, j = 1, \ldots, p$$

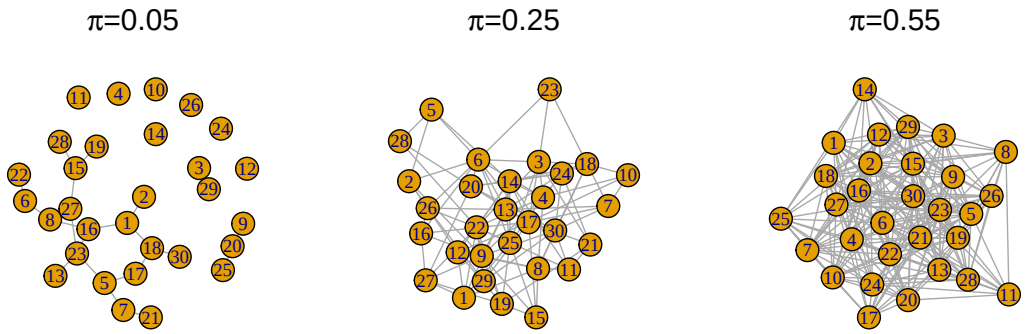


Figure 4.1: Random graphs with increasing density, $\pi = 0.05, 0.25, 0.55$ from left to right, over 30 vertices

where $\mathbf{u} = (u_1, \ldots, u_{p-1})$ is a $(p-1)$-dimensional vector with each element randomly generated by a uniform distribution: $u_k \overset{iid}{\sim} \text{Unif}(0.5, 1)$, $k = 1, \ldots, p-1$, following the approach pursued by [Fan et al., 2009] and [Guo et al., 2011]. This implies that the pairs of random variables (X_i, X_j) associated to entries c_{ij} such that $|j - i| = 1$ are dependent on one another conditioned on the remaining variables. On the other hand, pairs of variables (X_i, X_j) associated to entries c_{ij} such that $|j - i| > 1$ are retrieved as a combination of the other ones. The diagonal elements c_{ii} are set to 1.

The Markov-chain structure arises from the inverse of such $\mathbf{C}$ matrix, which is a tridiagonal matrix, with the only nonzero entries being in the main diagonal and the upper and lower ones.

Let us define $\mathbf{A}_{\mathcal{G}} = [a_{\mathcal{G},ij}]_{i,j=1,\ldots,p}$ as the adjacency matrix associated to a graph $\mathcal{G}$, such that $a_{\mathcal{G},ij} = 1 \iff i \sim_{\mathcal{G}} j \vee i = j$ and $a_{\mathcal{G},ij} = 0 \iff i \nsim_{\mathcal{G}} j$.

In order to generate the Markov-chain graph structure, then, it is sufficient to generate a tridiagonal adjacency matrix. We implemented this in `R` with the function `tridiag_1()`, which can be found in Appendix B.

As an example, creating a Markov-chain graph for 30 vertices will be done with `graph_from_adjacency_matrix(tridiag_1(30),mode="undir",diag=F)`, as shown in Figure 4.2.

4.2.3 Tree graph

The third structure is a tree graph. Even though we only deal with undirected graphs, the construction of a tree is more easily-explainable by referring to directed graphs.

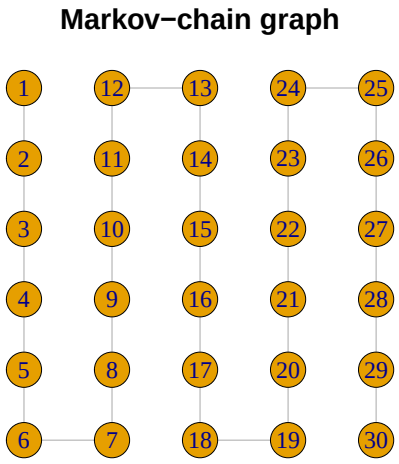


Figure 4.2: Markov-chain graph from tridiagonal matrix, 30 vertices.

The main characteristic of this object is that there are no cycles, i.e. no paths that connect a vertex to itself. The graph we construct in this step of the simulation study is a tree such that all vertices have the same number of children, except for the terminating ones (*leaves*). In this case, the degree distribution for the graph takes on three values: k for the root vertex of the tree (k being the number of children), $k+1$ for the intermediate nodes and 1 for the leaves.

The implementation is straightforward in `R` with the dedicated function in the `igraph` package.

For instance, running `make_tree(30, children=2, mode="undirected")` will produce the graph reported in Figure 4.3. We remark that if the parameter `children` were to be set to 1, this procedure would produce the a graph equal to the Markov-chain shown in Figure 4.2.

4.2.4 Scale-free graph

We choose to include this type of graph in this simulation due to its resemblance to the connectivity found in several types of biological networks [Jeong et al., 2000], [Babu et al., 2004], [Chen and Sharp, 2004], [Van Noort et al., 2004]. The characteristic of this graph is that the degrees of its nodes follow a power-law distribution [Barabási and Albert, 1999].

A vertex has degree k with probability $\frac{1}{k^\alpha}$, where the exponent α usually ranges between 2 and 4, regardless of the size of the network [Barabási and Albert, 1999]. For this reason, it is referred to as *scale-free network*.

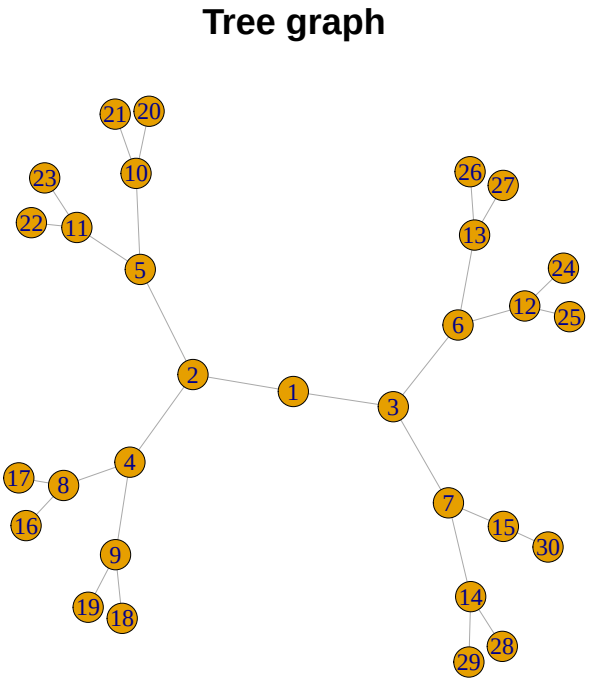


Figure 4.3: A tree graph, 30 vertices, with each non-leaf vertex having two children.

The power-law degree distribution generating the graph induces the presence of a few hub nodes, i.e. vertices with high degree, and a large amount of satellites, nodes with low degrees.

It should be noted that scale-free networks have two main properties which differentiate them from the general Erdős-Rényi model.

The first one is the *growing nature* of the graph. It is thought of as an expanding structure, starting with a set of nodes and progressively adding nodes and edges to the existing ones. An example would be the World Wide Web graph: new pages are created continuously and, as of their creation, they may influence the edges (in this case, hyper-links) of the overall network.

The second one is the *preferential attachment* that induces a new node to prevalently form interactions with vertices that already have a high connectivity. More precisely, for scale-free graphs, the probability Π that a new node connects to a node i depends on its degree k_i: $\Pi(k_i) = \frac{k_i}{\sum_{j=1}^{p-1} k_j}$.

In general, the preferential attachment follows a power-law distribution $\Pi(k) \sim k^{\phi}$. For scale-free graphs, the dependency is linear, i.e. $\phi = 1$. It has been observed that for some real networks, the preferential attachment

may depend on the degree of a node sub-linearly ($\phi < 1$) or super-linearly ($\phi > 1$) [Albert and Barabási, 2002]. However, the scaling property of the graph is only guaranteed for linear preferential attachment [Krapivsky et al., 2000].

The algorithm to construct a scale-free graph follows a sequential procedure. First, the graph is created with m_0 vertices. Then, at each step t a new vertex is added and m edges are formed with the existing graph. This implies that at each time point t there are $m \times t$ edges. The parameter m then implicitly regulates the density of the graph. Such a model is proved to converge to a stationary degree distribution following a power law $P(k) \sim k^{-3}$, i.e. $\alpha = 3$.

Let us now construct a scale-free graph with 30 vertices. The `R` function `sample_pa()` in the `igraph` package produces a Barabási-Albert game with the following specifications:

- `p`: the number of vertices for the graph
- `power`: the power for preferential attachment. The default is 1 for linear attachment
- `m`: the number of edges to add at each step of the algorithm
- `directed`: logical value to indicate whether the graph is directed or undirected (default is `TRUE`)

The graphs that may be built using this `R` function vary largely based on the choices for the number of edges m and the power of preferential attachment ϕ. We report an exploratory analysis on graphs with 30 vertices.

First, we analyse the impact of the power for preferential attachment. Figure 4.4 shows four graphs with 30 vertices, built with the `sample_pa()` function, with $m = 1$ (i.e. 29 edges) and varying values for the power ϕ.

For $\phi = 0.5$, the graph almost resembles a tree, but as the power increases, the structures rely more and more on hub nodes. In fact, when $\phi = 2.5$, the majority of the nodes are satellites to one node. We remark that only a preferential attachment power between $\phi = 1$ and $\phi = 1.5$ results in an accurate representation of a scale-free graph. We fix this value from now onwards at $\phi = 1.1$, to maintain linearity in preferential attachment while giving a slight incentive to hub nodes.

Secondly, we compare the structures resulting from different values for m. With 30 vertices and a fixed value $\phi = 1.1$, we construct four graphs, with

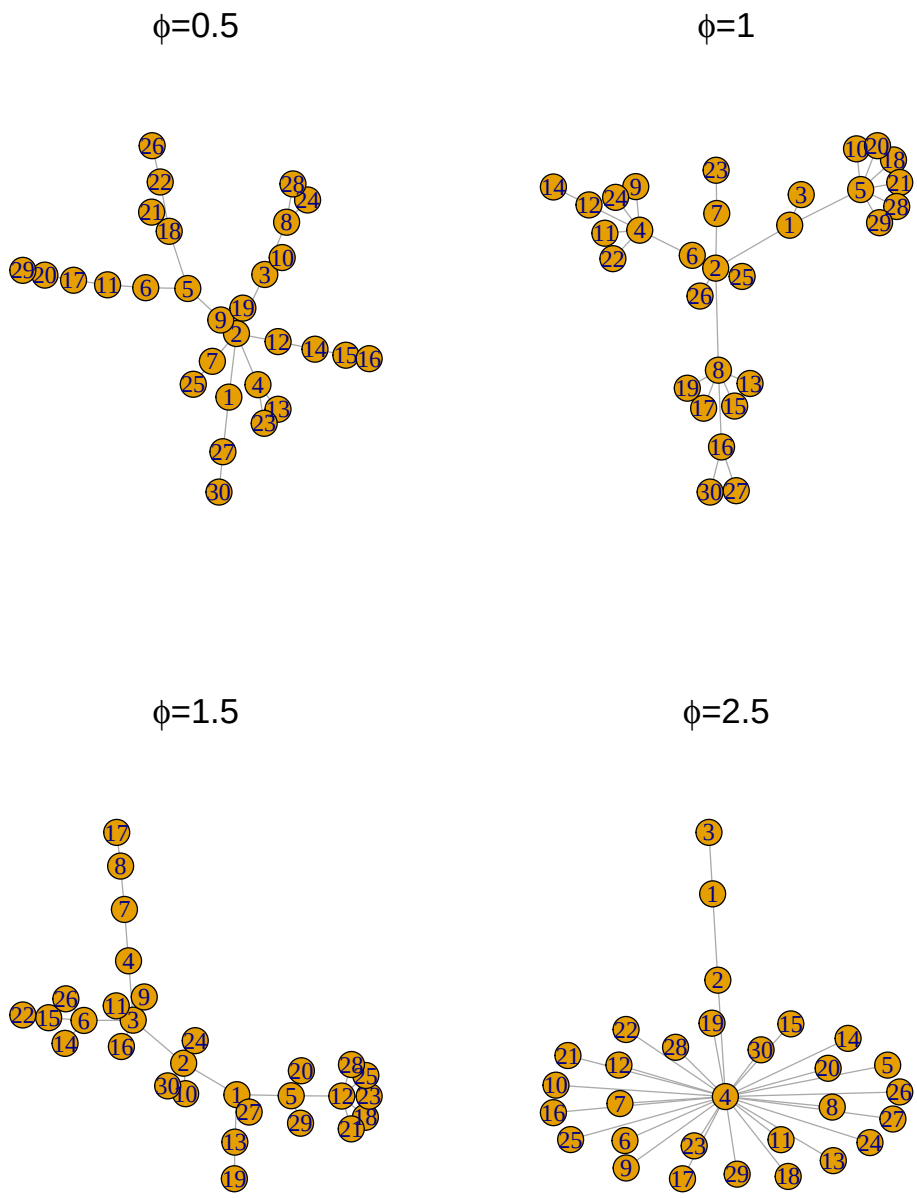


Figure 4.4: Scale-free graphs for 30 vertices, with varying power for preferential attachment ($\phi = \{0.5, 1, 1.5, 2.5\}$)

$m = \{1, 2, 3, 5\}$. From Figure 4.5, the hub structure emerges clearly when $m = 1$. At each step, single edges are attached to already highly-connected nodes, feeding the so-called *rich get richer* mechanism. This is progressively

less obvious when m increases, making the graph on the right ($m = 5$) similar to a random graph.

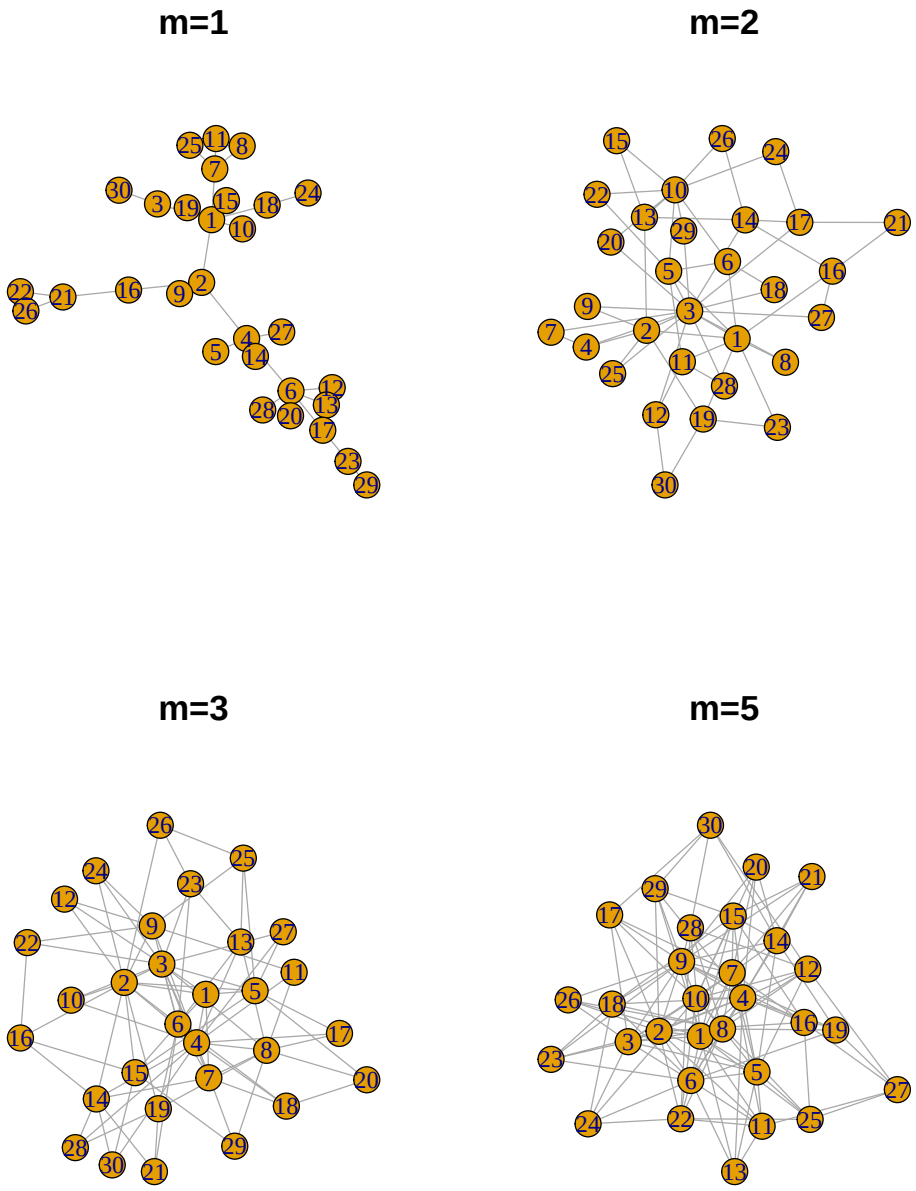


Figure 4.5: Scale-free graphs for 30 vertices, with $\phi = 1.1$ and varying number of edges added at each step of the BA algorithm ($m = \{1, 2, 3, 5\}$)

Evidently, the number of edges added at each step of the algorithm influ-

ences directly the density of the graph. More formally, we define the graph density as:

$$GD = \frac{|E|}{|V|(|V|-1)/2}$$

Since the `sample_pa()` function initializes the graph with $m_0 = 1$ nodes, we automatically have that for $m = 1$, $GD = \frac{29}{30\times 29/2} = 0.067$ and for $m = 5$, $GD = \frac{29\times 5}{30\times 29/2} = 0.333$. It is clear that as the dimension of the vertex set increases, a fixed value for m will result in a sparser graph. For instance, for the scenario $p = 100$, keeping $m = 5$ would result in $GD = 0.098$.

We keep in mind that for many application purposes, such as biological networks, the graphs have low density and for this reason we choose $m = 1$ for $p = 30$ and $m = 4$ for $p = 100$. Moreover, by choosing a parameter ϕ that promotes the hub structure emerging in sparser scale-free graphs, we conclude this exploratory analysis by opting for the following choices in our simulation:

- `sample_pa(30, power=1.1, m=1, directed=F)`
- `sample_pa(100, power=1.1, m=4, directed=F)`

In both cases, we produce graphs for which $GD \approxeq 0.07$.

To conclude on the graph structures, we make a comparison of the degree distribution between the Erdős-Rényi (ER) model and the Barabási-Albert (BA) model. For this analysis, we use a much larger graph, $p = 10000$.

In theory, what we expect is that the BA model gives larger opportunity to higher degrees, whereas the ER model would show a distribution concentrated on small degrees and with zero frequency of higher connectivities. To have the same density for the two models, we use:

```
er<- sample_gnp(p,1/p)
ba<- sample_pa(p,m=1,directed=F)
```

The graphs will have approximately the same number of edges: exactly $p - 1$ on the BA model and expectedly $p - 1$ in the ER one.

As we see from Figure 4.6, the random graph does not allow degrees greater than 7. On the other hand, the scale-free model has a large amount of small degrees, but also the presence of nodes that are highly connected, having up to circa 150 adjacencies. The shape of the degree distribution for the BA model clearly reconnects to the power-law distribution, whereas for the ER model, the distribution clearly follows a Poisson, more precisely a $Poi(1)$.

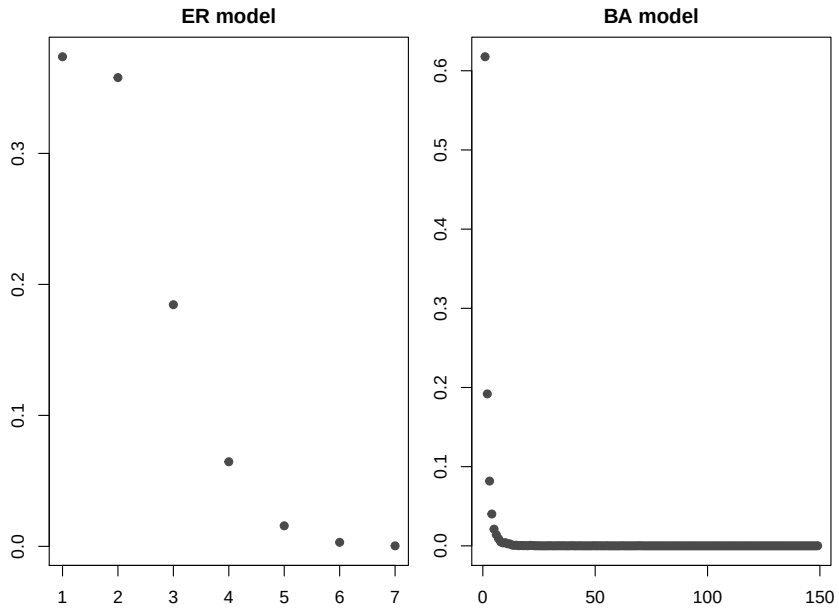


Figure 4.6: Degree distribution for Random Graph and Scale-free Graph, with 10000 vertices and 9999 edges.

4.3 Covariance Matrix Generation

The next step in the simulation is to generate a covariance matrix for each graph structure. The first step is to construct a starting base, a covariance matrix that can then be fitted to each graph structure, in terms of zero patterns.

Generation of $\mathbf{\Sigma}$

The data that we will generate for this simulation is Gaussian-distributed, as mentioned in Section 2.3. We assume that the data is centred around zero and thus we generate, for each simulation scenario, n observations from the p-variate distribution $\mathbf{X}_V \sim \mathcal{N}_p(\mathbf{0}_p, \mathbf{\Sigma})$. In this Section, we discuss how to construct the population covariance matrix $\mathbf{\Sigma}$ from which the data are generated.

Given that he main characteristics for generating a covariance matrix are that it needs to be a square $p \times p$, symmetric and positive-definite matrix we explore four generation strategies. The first starts from the equicorrelation matrix; the second generates entries of the correlation matrix from a Beta distribution; the third imposes bounds on the eigenvalues; the fourth relies on a Wishart distribution.

Equicorrelation Matrix

One way to generate such a matrix can be to start from an equicorrelation matrix $\mathbf{\Omega}$, i.e. a matrix of the form

$$\mathbf{\Omega} = \begin{bmatrix} 1 & \omega & \omega & \dots & \omega \\ \omega & 1 & \omega & \dots & \omega \\ \omega & \omega & 1 & \dots & \omega \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \omega & \omega & \omega & \dots & 1 \end{bmatrix}$$

Where $\omega \in \left(-\frac{1}{p-1}, 1\right]$, to ensure positive-definiteness.

The covariance matrix $\mathbf{\Sigma}$ is then obtained by inverting such $\mathbf{\Omega}$.

Beta Distribution

An alternative approach would be to construct a correlation matrix $\mathbf{R}$ by drawing its entries from a *Beta* distribution, as suggested in [Joe, 2006]. Each correlation has a $Beta(\alpha, \alpha)$ distribution on the (-1,1) interval, where $\alpha = a + \frac{p-2}{2}$. Setting $a = 1$ leads to a random matrix which is uniform over space of positive definite correlation matrices. The *Beta* distribution for the correlations complies with the distributional constraints given in [Joe, 2006] to ensure positive definiteness of the correlation matrix.

In order to obtain the covariance matrix $\mathbf{\Sigma}$ we generate p variances $(\sigma_1^2, \dots, \sigma_p^2)$. Then $\mathbf{\Sigma} = diag(\sigma_1^2, \dots, \sigma_p^2) \times \mathbf{R} \times diag(\sigma_1^2, \dots, \sigma_p^2)$.

This matrix can be constructed using the `genPositiveDefMat()` command from the `clusterGeneration` package. This function includes a variety of methods to generate positive definite covariance matrices. In this case, we use the following specification:

```
genPositiveDefMat(p,covMethod="unifcorr", rangeVar= c(1.5,10))
```

The method `covMethod="unifcorr"` generates the correlation matrix according to [Joe, 2006] and then generates p covariances randomly ranging within the `rangeVar` interval. The covariance matrix is obtained by multiplying the correlation matrix by the random covariances.

Bounded Eigenvalues

Another option consists in drawing p random eigenvalues $(\lambda_1, \dots, \lambda_p)$ and a random orthogonal matrix $\mathbf{Q}$. The covariance matrix is the constructed as $\mathbf{\Sigma} = \mathbf{Q} \times diag(\lambda_1, \dots, \lambda_p) \times \mathbf{Q}^T$.

This method corresponds to a different specification for the `genPositiveDefMat()` function in the `clusterGeneration` package. Therefore, the covariance matrix can be obtained by running the following:

```
genPositiveDefMat(p,covMethod="eigen", lambdaLow=2)
```

Where `lambdaLow=2` gives a lower bound for the smallest eigenvalue to be generated.

Wishart Distribution

Lastly, we explore the possibility of generating the covariance matrix according to a Wishart distribution [Mardia et al., 1979].

Let us have a $m \times p$ data matrix $\mathbf{Z}$, following $\mathbf{Z} \sim \mathcal{N}_p(\mathbf{0}_p, \mathbf{V})$.
Let $\mathbf{\Sigma} = \mathbf{Z}^T\mathbf{Z}$. Then, $\mathbf{\Sigma} \sim \mathcal{W}_p(m, \mathbf{V})$, i.e. $\mathbf{\Sigma}$ is a $p \times p$ Wishart-distributed matrix with m degrees of freedom and scale matrix $\mathbf{V}$. The degrees of freedom m are such that $m > p$, for positive definiteness. If $\mathbf{V} = \mathbf{I}_p$, the distribution is said to be in standard form.

We can then draw the positive definite Wishart-distributed covariance matrix using the `rWishart()` function. We draw one element from such distribution in standard form, by executing:

```
rWishart(1, df=500, diag(p))
```

The degrees of freedom `df=500` are set greater than the number of variables p and the scale matrix is the identity matrix $\mathbf{I}_p$.

Matrix Completion

Once the covariance matrix is generated, it needs to be adjusted in order to comply with each graph $\mathcal{G}$, whose structure in built as discussed in Section 4.2. For instance, given a graph $\mathcal{G} = (V, E)$ such that $\{(1,3),(2,3),(2,p)\} \notin E$, the covariance matrix will then have the form:

$$\mathbf{\Sigma}_{\mathcal{G}} = \begin{bmatrix} \sigma_{11} & \sigma_{12} & * & \dots & \sigma_{1p} \\ \sigma_{21} & \sigma_{22} & * & \dots & * \\ * & * & \sigma_{33} & \dots & \sigma_{3p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \sigma_{p1} & * & \sigma_{p3} & \dots & \sigma_{pp} \end{bmatrix}$$

This process is called *matrix completion*, as discussed in Section 2.4. Since the starting point for the completion is a positive-definite matrix by construction, we can perform the operation either via the IPS algorithm or via a Graphical Lasso with $\rho = 0$, which corresponds to a non-penalized maximum likelihood problem.

This is implemented in the `glasso` function of the corresponding package. When `rho=0`, the function requires specifying a list of missing edges, which we obtain from the four graphs created in Section 4.2 (see the `get_zero_structure()` function in Appendix B). In the implementation

of the `glasso` function, this corresponds to setting a $p \times p$ regularization matrix, where $\rho_{ij} = 0$ if the element corresponds to a present edge and $\rho_{ij} = 10^9$ if the element corresponds to a missing edge.

Therefore, using the Graphical Lasso algorithm, we obtain the completed covariance matrices for each of the graphs (see the `completed_matrices_H1()` function in the Appendix).

In order to choose which method is the most suitable to generate the completed covariance matrices, we perform an exploratory analysis on the distribution of the non-zero elements of the corresponding concentration matrices, based on the random graph with $\pi = 0.1$, for $p = 30$.

The reason for this analysis lies in the advantage of dealing with well-defined concentration matrices, having bounded eigenvalues and non-zero entries bounded away from zero, as discussed in [Guo et al., 2011].

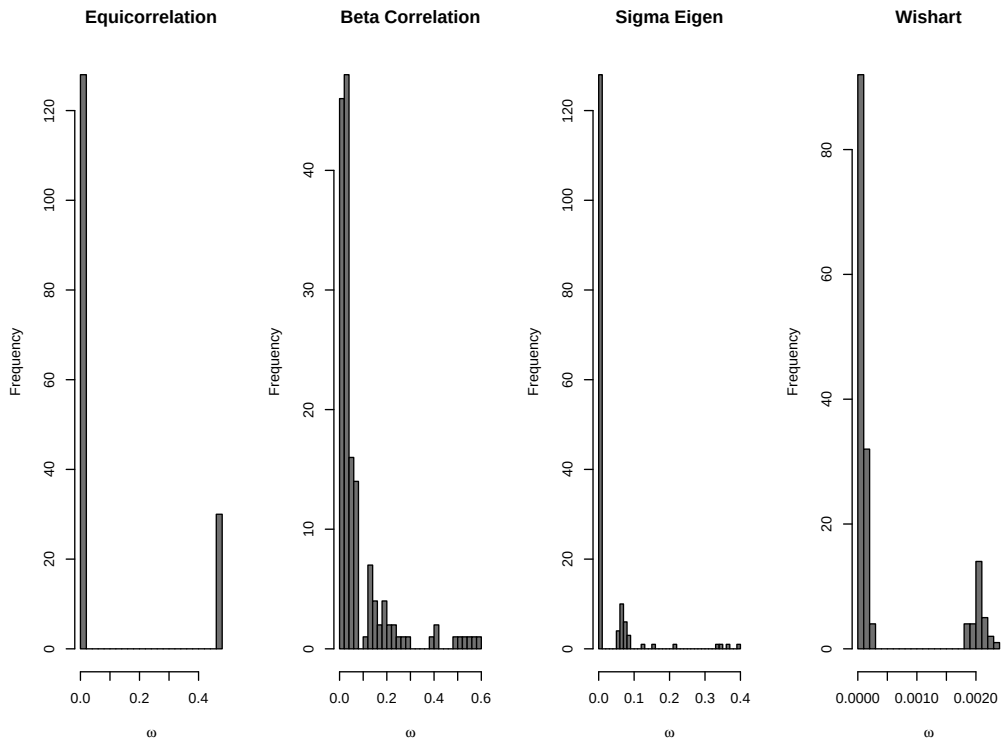


Figure 4.7: Distribution of non-zero entries of concentration matrices generated with four methods

Figure 4.7 shows the frequencies for values of the concentration matrices obtained, after completion, for each of the four generation strategies dis-

cussed. As we see, the main issue with the equicorrelation approach is that the non-zero elements are clearly concentrated only on two values. The Beta correlation approach seems the most viable, as the entries are sensibly differentiated and, moreover, they do not take on extremely low values. The eigenvalue approach does show a very large amount of values that are highly close to zero. As for the Wishart distribution, all entries of the corresponding concentration matrix are smaller than 0.0025.

Table 4.1: Minimum absolute non-zero values ($\min(|\mathbf{\Omega}|)$), minimum eigenvalues ($\phi_{\min}(\mathbf{\Omega})$) and maximum eigenvalues ($\phi_{\max}(\mathbf{\Omega})$) for the concentration matrices obtained via the equicorrelation, the Beta distribution, the bounded eigenvalues and the Wishart distribution generation strategies

	EQUICORR	BETA	EIGEN	WISHART
$\min(\lvert\mathbf{\Omega}\rvert)$	0.0111306	0.0000619	0.0000220	0.0000011
$\phi_{\min}(\mathbf{\Omega})$	0.2880298	0.0695605	0.0500301	0.0016816
$\phi_{\max}(\mathbf{\Omega})$	0.4058412	0.7310866	0.4045617	0.0025630

From Table 4.1, we remark that the Wishart distribution produces a concentration matrix with very small values and eigenvalues very close to each other and close to zero. The equicorrelation approach has larger eigenvalues, but as seen from Figure 4.7, the values concentrate only on two points. Both the processes of generating the covariance matrix from Beta-distributed entries and from random eigenvalues show large enough minimum eigenvalues and non-zero elements on the completed concentration matrices. We conclude by choosing the correlation matrix with Beta-distributed entries both for the wider variety of values in the corresponding concentration matrix (Figure 4.7) and for the minimum eigenvalue being farther away from zero.

4.3.1 Group Distinction

In Section 4.3, we generated one covariance matrix for each type of graph. However, we need to perform a distinction between the groups constituting the basis for the multiple graph structure learning. In practice, we focus on a two-group setting, indicated as $H = 2$, although the functions are generalized to any choice for H.

The procedure for the group distinction starts by choosing the portion of changed edges from common graphs, δ. Here we choose $\delta = \{0.05, 0.20\}$, i.e. each of the G graphs will be distinguished in H ways, with a portion δ of edges changed from present to absent or viceversa. A representation

of the scheme is shown in Figure 4.8. The graphs obtained after the group distinction are the object of this simulation analysis. Having $G = 4$ graph structures and $H = 2$ groups, each scenario of the simulation will involve the learning of eight graphs.

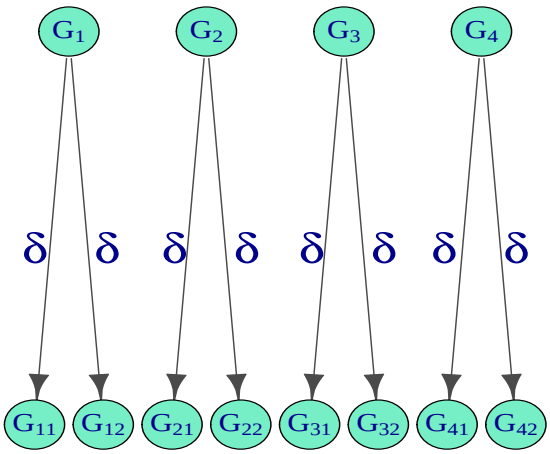


Figure 4.8: Scheme of the generation of graphs for each structure, with a portion δ of changed edges in each distinction.

The distinction process operates through the adjacency matrices associated to each graph, i.e. by changing the zero or non-zero elements of each adjacency matrix and thus resulting in adding or removing edges to their corresponding graph. For simplicity, we first transform the lower triangular part of each adjacency matrix, diagonal excluded, and we subsequently symmetrize.

We randomly generate edge identifiers which correspond to elements of the adjacency matrix to be changed, from missing to present edge and viceversa. For each graph structure and for each dissimilarity degree δ, we generate two vectors of length $\lceil \delta \frac{p(p-1)}{2} \rceil$, where $\lceil \cdot \rceil$ indicates the ceiling function. Each of the vectors contains elements in the $\left\{1, \ldots, \frac{p(p-1)}{2}\right\}$ integer interval, corresponding to the positions on the vectorized lower triangular part of the adjacency matrix to be changed. We indicate each of these vectors as $\mathbf{n}_h,\ h = 1, 2$.

In `R`, this is done by drawing elements from the $\left\{1, \ldots, \frac{p(p-1)}{2}\right\}$ interval using the `sample()` function, implemented in the Base package, with specification `replace=FALSE`, indicating that each element can only be drawn once.

While there is a different generation of $\mathbf{n}_1$ and $\mathbf{n}_2$ for each graph structure and each dissimilarity degree, we explain the procedure keeping g and δ fixed for notation purposes.

Since the adjacency matrix contains 0-1 elements, to obtain its opposite for the elements corresponding to changing edges it suffices to create its complement to 1. Let us denote the common adjacency matrix as $\mathbf{A}$. The new adjacency matrices $\mathbf{A}_h$ are obtained using the following:

$$[\text{vech}^d(\mathbf{A}_h)]_i = 1 - [\text{vech}^d(\mathbf{A})]_i \qquad \forall\ i \in \mathbf{n}_h,\ h = 1, 2$$

We indicate as $\text{vech}^d(\cdot)$ the operator vectorizing the lower triangular part of a matrix, main diagonal excluded. After changing the elements in the positions indicated by $\mathbf{n}_h$, the matrix is symmetrized. $\mathbf{A}_h$ is the new adjacency matrix in the h-th distinction. The `R` implementation for this procedure is in the `group_zeros()` function, which can be found in Appendix B.

At this point, we are able to distinguish two graphs $\mathcal{G}_1$ and $\mathcal{G}_2$, associated to the modified adjacency matrices $\mathbf{A}_1$ and $\mathbf{A}_2$.

After this process, the next step is to perform once again a completion procedure on the covariance matrix $\mathbf{\Sigma}$, using the new edge patterns for each $\mathcal{G}_h$. As previously discussed, we perform the matrix completion by applying a Graphical Lasso method with penalization $\rho = 0$ and edge patterns given by the now diversified groups with graphs $\mathcal{G}_h,\ h = 1, 2$. It is implemented in the `completed_matrices_H()` function, reported in Appendix B.

After performing the matrix completion, we obtain two covariance matrices for each graph structure and for each dissimilarity degree, which we indicate as $\mathbf{\Sigma}^{(1)}$ and $\mathbf{\Sigma}^{(2)}$.

To summarize and formalize the group distinction procedure, we report Algorithm 2.

Algorithm 2: Group Distinction

1) Starting point: a $p \times p$ covariance matrix $\mathbf{\Sigma}$ and a graph $\mathcal{G}$ associated to a $p \times p$ adjacency matrix $\mathbf{A}$
2) Set a dissimilarity degree $\delta \in [0, 1]$
3) Generate randomly vectors $\mathbf{n}_h$ of length $\lceil \delta \frac{p(p-1)}{2} \rceil$ of integers between 1 and $\frac{p(p-1)}{2}$
4) Obtain new adjacency matrices using $[\text{vech}^d(\mathbf{A}_h)]_i = 1 - [\text{vech}^d(\mathbf{A})]_i,\ \forall i \in \mathbf{n}_h,\ \forall h$
5) Associate each new adjacency matrix to a graph $\mathcal{G}_h$
6) Perform matrix completion on $\mathbf{\Sigma}$ based on $\mathcal{G}_h$

Minimum Spanning Tree

One issue with this group distinction method is that it changes edges regardless of the graph structure. This is not relevant for random graphs, and it is marginally of interest for scale-free ones, but it does change the nature of Markov-chain and tree graphs.

The Markov-chain graph can not easily be adapted to the group distinction, since the only way to maintain the chain structure is to swap node labels. We can then simply accept a variation of the graph, which has the same core, but a few alterations.

On the other hand, the tree graph can be slightly modified after the group distinction by recovering its Minimum Spanning Tree, i.e. a graph as close as possible that satisfies the property of the tree (no cycles). This is operated by scanning the nodes and removing any edges that form a cycle. It is easily applied in `R` via the `mst()` function in the `igraph` package. See also the `make_mst()` function in Appendix B.

However, it should be remarked that using this procedure, the dissimilarity degree δ for trees is very likely to be different from the other structures. From a set of 1000 simulations on $p = 30$ nodes, it appears that in this case, even though δ was set to 20%, the actual dissimilarity resulted in 10%. This is a consequence of the fact that regardless of the change, this type of graphs tends to be more similar after the group distinction in order to maintain the tree structure, e.g. the same edges have to be removed.

4.4 Data Generation

In this step of the simulation, we generate the data on which the MGSL procedures are evaluated. As previously discussed, we assume that the data follow a p-variate Gaussian distribution, centred around zero.

The covariance matrices for each group are the $\mathbf{\Sigma}^{(h)}$ matrices obtained in Section 4.3.1, keeping in mind that they also vary for each graph structure and each dissimilarity degree.

Therefore, we generate H samples of size n from the corresponding p-variate Gaussian distributions:

$$(X_1^{(i)}, \ldots, X_p^{(i)}) \overset{iid}{\sim} \mathcal{N}(\mathbf{0}_p, \mathbf{\Sigma}_p^{(h)}) \qquad i = 1, \ldots, n;\ h = 1, 2$$

Since we already have the population covariance matrices, in `R` it suffices to generate the multivariate distributions using the `rmvnorm()` function (`mvtnorm` package) to draw the n observations for each group from a p-variate

Gaussian. We assume n remains constant across the groups. We indicate each of the generated $n \times p$ data matrices as $\mathbf{x}^{(h)},\ h = 1, 2$.

4.5 Methods

This Section concerns the application of the methods described in Chapter 3 to the data generated in Section 4.4.

Since the learning methods are based on the sample covariance matrices, we calculate them based on the data generated for the h-th group as $\mathbf{S}^{(h)} = \frac{1}{n}\mathbf{x}^{(h)T}\mathbf{x}^{(h)}$.

We then apply the Graphical Lasso separately on $\mathbf{S}^{(1)}$ and $\mathbf{S}^{(2)}$ for each graph structure and each dissimilarity degree using the `glasso` function, with regularization parameters chosen through AIC (see Section 4.6).

For the Rarametrization Joint Graphical Lasso ([Guo et al., 2011]), we implemented an `R` function that takes in the sample covariance matrices $\mathbf{S}^{(1)}$ and $\mathbf{S}^{(2)}$ and computes the joint estimates for the concentration matrices following Algorithm 1 (Section 3.3.1). To estimate the coefficients of the penalized linear regression needed for each step (i.e. row-column) of the estimation of the covariance matrices in the Joint Graphical Lasso Equation (3.5), we use a coordinate descent method as indicated in [Friedman et al., 2007]. This implementation can be found in Appendix B as `guo_glasso()`.

We specify that the initial scalar ν added to the diagonal of sample covariance matrices for positive-definiteness does not coincide with the regularization parameter ρ. While we do focus on the tuning of the latter in Section 4.6, we choose to keep ν reasonably small (e.g. $\nu = 0.05$), so as to have negligible impact on the final estimates.

As for the joint learning methods using the Fused and Group Lasso penalties, we employ the dedicated `JGL()` function (`JGL` package) with specifications `penalty="fused"` and `penalty="group"` respectively. For the choice of penalization parameters, see Section 4.6.

The method based on the direct estimation of differential networks ([Zhao et al., 2014]) is implemented for a two-group application in the `dpm()` function (`https://github.com/sdzhao/dpm`). This function takes as input the data $\mathbf{x}^{(1)}$ and $\mathbf{x}^{(2)}$, for each of our graph structures and dissimilarity degrees, and the number `nlambda` of values to be considered for the constraint. It also allows different model selection criteria, such as AIC, BIC and cross validation.

All criteria are based on loss functions obtained with different norms such as:

$$L_\infty(\rho_n) = |\mathbf{S}^{(1)}\hat{\mathbf{\Delta}}_{\rho_n}\mathbf{S}^{(2)} - (\mathbf{S}^{(1)} - \mathbf{S}^{(2)})|_\infty,$$

where $\hat{\mathbf{\Delta}}_{\rho_n}$ is the estimate for the difference of concentration matrices associated the graph of differences and computed with penalty term equal to ρ_n.

Six options are given for the norm: sup-norm, element-wise l_1, matrix L_1, spectral, Frobenius and nuclear norm. The model selection criterion is then evaluated, for instance, as:

$$AIC_{\rho_n} = 2nL_*(\rho_n) + 2df_{\rho_n},$$

where $L_*(\cdot)$ is the chosen loss function, $df_{\rho_n} = |\{(i,j) : i < j, [\hat{\mathbf{\Delta}}_{\rho_n}]_{ij} \neq 0\}|$.

The `dpm()` function gives as output a list of `nlambda` matrices $\hat{\mathbf{\Delta}}$ estimating $\mathbf{\Delta}_0 = \mathbf{\Omega}^{(1)} - \mathbf{\Omega}^{(2)}$, which we apply to each graph structure for each dissimilarity degree. The optimal one can be chosen according to the preferred loss function in the `$opt` object of the output.

We report the result of learning the structure of the graph of differences for the random graph model, with $p = 30$, $n = 200$ and $\delta = 0.05$. We compare the optimal estimates $\hat{\mathbf{\Delta}}$ according to each of the loss functions. The loss function based on the sup-norm chooses a regularization parameter $\rho = 2.03$, whereas the remaining ones all choose $\rho = 0.14$. The selection is operated through AIC.

Figure 4.9 shows seven graphs: in each of them, an edge represents a conditional dependence between two variables that differs in terms of edge values among the two groups. The graph on the left is the true graph of differences, i.e. having an edge only for elements of $\mathbf{\Delta}_0$ different from zero. The remaining ones correspond to the estimated $\hat{\mathbf{\Delta}}$, based on each of the loss functions. As we see, only the sup-norm is able to reconstruct a graph that is similar to the true one, whereas the remaining ones produce graphs that are not comparable to the true one. For this reason, we choose to employ the sup-norm in our set of simulations, as the other options appear to be producing extremely dense graphs.

4.6 Model Selection criteria

The methods for learning the graphs rely heavily on the choice of the regularization parameters, which is discussed in this Section. Since these control

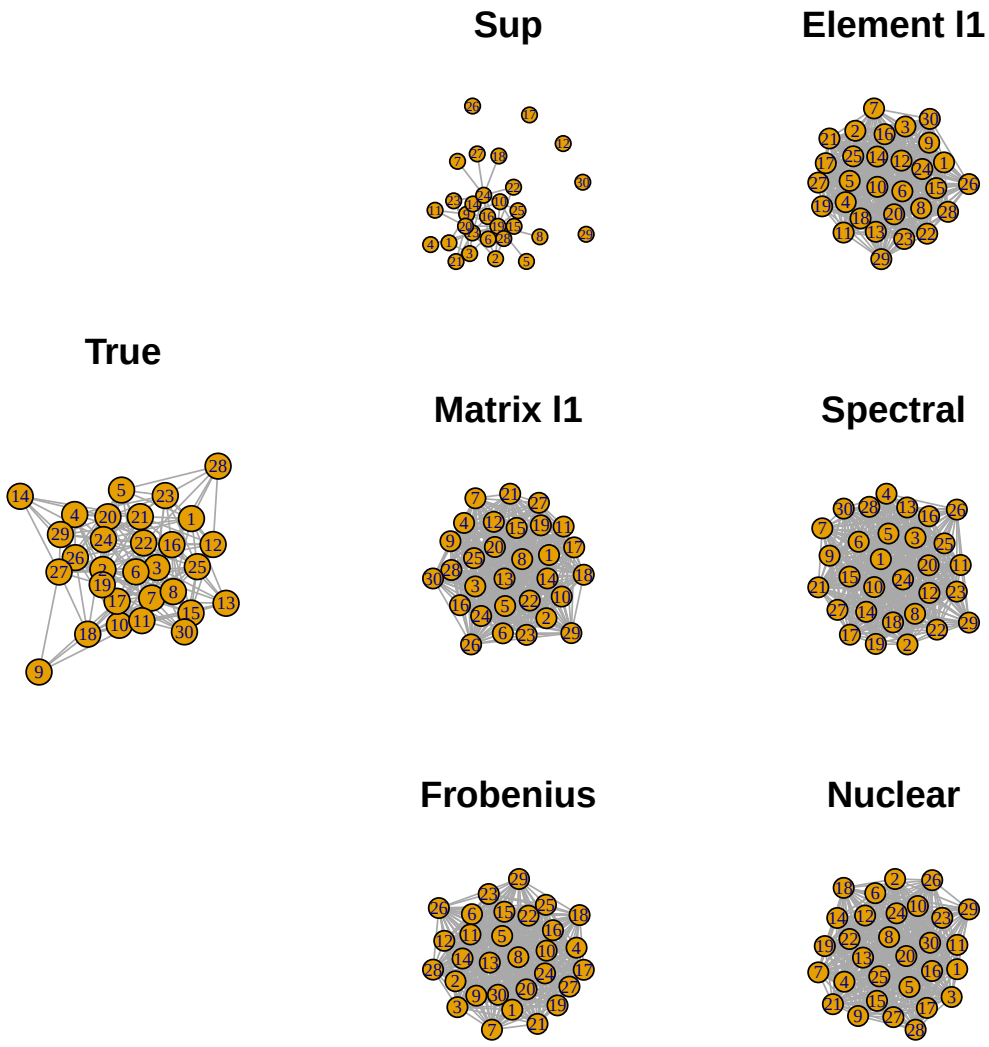


Figure 4.9: Graphs representing differential conditional dependence patterns for the Erdős-Rényi model, on the population (left) and using the direct estimation method with loss functions based on sup, element-wise $\ell 1$, matrix $\ell 1$, spectral, Frobenius and nuclear norms

the sparsity of the graphs, it is straightforward that two graphs obtained by imposing different penalty terms result in different edge patterns. For this reason, in regard to choosing the regularization parameter, we refer to *model selection*.

In the literature, the most common choices for model selection criteria for

MGSL problems are the cross-validation, the AIC and the BIC (e.g. [Danaher et al., 2014], [Guo et al., 2011]). All these methods are based on the log-likelihood evaluated for the estimated concentration matrices.

The cross-validation and the information criteria differ on more than one level. Since the cross-validation (see e.g. Equation (3.7) for Reparametrization JGL) requires evaluating the log-likelihood on segments of the data, it is clearly more time-consuming. The advantage would theoretically be that the estimates should produce better-fitting graphs. Furthermore, the AIC and the BIC are highly dependent on the degrees of freedom of the model, which are calculated as the number of present edges in the chosen graph. See, for instance, Equations (3.11) for AIC and (3.6) for BIC.

Given the theoretical differences between these model selection strategies, we explored the empirical performance of each of these, based on two aspects: the computing time and the ability to reproduce a graph density as close as possible to the true one.

We report a few results comparing these features for the Separate Graphical Lasso and the Reparametrization JGL for two scenarios: $p = 30,\ n = 20,\ \delta = 0.2$ and $p = 100,\ n = 400,\ \delta = 0.05$.

We specify the same grid of values to choose from for all methods. For the Separate Graphical Lasso, we notice that the regularization parameter usually lies between 0.2 and 1.6, therefore we generate in `R` a vector of possible penalization levels as:

```
rho<- seq(from=1e-2, to=1.6, length.out=30)
```

From which all the selection criteria evaluate their candidate for the optimal regularization parameter.

For the Reparametrization JGL, we find that the chosen optimal values for ρ usually lie between the 10^{-6} and 10^{-4} interval. For this reason, we specify the vector of candidates in `R` as:

```
rho<- seq(from=1e-6, to=1e-4, length.out=30)
```

For all the selection criteria.

We refer to Table 4.2 for the results on the first scenario for the Reparametrization Joint Graphical Lasso. As we see in Table 4.2a, the 5-fold cross validation requires a much larger time to compute the estimates for the concentration matrices. However, as reported in Tables 4.2b-4.2e, the estimated densities of the learnt graphs are extremely similar among the three criteria. More precisely, the AIC produces slightly denser graphs, which are to be preferred, since closer to the true densities.

Table 4.2: Comparison of model selection methods for Reparametrization Joint Graphical Lasso, $p = 30, n = 20, \delta = 0.2$. CV is 5-fold.

(a) Computing times (in seconds)

	CV	AIC	BIC
TIME	1825.46	332.36	325.91

(b) Density of random graph ($\pi = 0.05$)

	h=1	h=2
TRUE	0.22	0.23
CV	0.03	0.02
AIC	0.06	0.05
BIC	0.03	0.02

(c) Density of Markov chain graph

	h=1	h=2
TRUE	0.25	0.23
CV	0.06	0.06
AIC	0.03	0.03
BIC	0.03	0.03

(d) Density of tree graph

	h=1	h=2
TRUE	0.07	0.06
CV	0.03	0.03
AIC	0.04	0.05
BIC	0.03	0.04

(e) Density of scale-free graph

	h=1	h=2
TRUE	0.25	0.23
CV	0.04	0.06
AIC	0.04	0.06
BIC	0.04	0.06

For the second scenario, we show results for the Separate Graphical Lasso. Table 4.3a shows that the 10-fold cross-validation requires approximately 9 times as much computational time as the other selection methods. In spite of this, the regularization parameter chosen are not closer to reproducing the true graph densities than the AIC. From Tables 4.3b through 4.3e we gather that, as a matter of fact, the AIC is able to choose parameters that construct graphs with densities close to the true ones. The BIC, on the other hand, tends to lead to much sparser graphs. Other reports for these two scenarios can be found in the Appendix (Tables A.1 and A.2).

These results give us evidence that, while the cross-validation is undeniably more computationally-expensive than the AIC and the BIC, there is no remarkable improvement in the fit of the estimates, in terms of reproducing the density of the true graphs.

Given the tendency of the BIC to estimate larger regularization parameters than needed, we choose the AIC as the preferred model selection criterion from here onwards. We remark that the AIC is implemented in the

Table 4.3: Comparison of model selection methods for Separate Graphical Lasso, $p = 100,\ n = 400,\ \delta = 0.05$. CV is 10-fold.

(a) Computing times (in seconds)

	CV	AIC	BIC
TIME	17.84	2.09	2.01

(b) Density of random graph ($\pi = 0.05$)

	h=1	h=2
TRUE	0.09	0.09
CV	0.14	0.10
AIC	0.11	0.07
BIC	0.01	0.02

(c) Density of Markov chain graph

	h=1	h=2
TRUE	0.07	0.07
CV	0.11	0.10
BIC	0.08	0.07
AIC	0.01	0.01

(d) Density of tree graph

	h=1	h=2
TRUE	0.02	0.02
CV	0.10	0.10
AIC	0.06	0.10
BIC	0.01	0.01

(e) Density of scale-free graph

	h=1	h=2
TRUE	0.12	0.12
CV	0.10	0.10
AIC	0.06	0.10
BIC	0.01	0.01

`dpm()` function for the method of Direct Estimation of Differential Networks ([Zhao et al., 2014]). Furthermore, we remind that a bi-dimensional grid search would be needed to identify the optimal regularization parameters ρ_1 and ρ_2 for the Fused and Group JGL ([Danaher et al., 2014]), but that this is simplified with a double line search, as discussed in Section 3.3.2.

4.6.1 AIC selection algorithm

Given the choice of the AIC as model selection criterion for the regularization parameters, the issue when fixing the range of values among which to choose the optimal one lies in the lack of a-priori knowledge of the order of magnitude of the parameter itself. While an ideal solution would be to set a grid as large as possible, while keeping the intervals between values as small as possible, this could be very time-consuming.

Therefore, we think that it is more sensible to select a broad initial range and update its extrema based on the values for the parameters found as optimal based on the AIC. This allows a precise individuation of the optimal parameters, by progressively narrowing the intervals between candidates.

It should be reminded, though, that when $p > n$ there is a lower bound for the penalty term under which the starting points for the Graphical Lasso algorithms are not positive-definite matrices. The procedure for updating the range of penalization candidates then should not allow for these values to be employed in the computation of candidate estimates for the concentration matrices.

To take this element into consideration, we include a check on the positive definiteness of the starting point for the Graphical Lasso procedures. For instance, for the separate Graphical Lasso, we know that if $\mathbf{S} + \rho\mathbf{I}$ is not positive definite, the algorithm does not converge.

Algorithm 3: Select regularization parameters minimizing the AIC

Initialize extrema for the parameter range $\rho_L^{(0)}$ and $\rho_R^{(0)}$ and generate a vector of length l of equally-spaced values between $\rho_L^{(0)}$ and $\rho_R^{(0)}$
while $\mathbf{S} + \rho_L^{(0)}\mathbf{I}$ is not positive definite **do**
  Increase $\rho_L^{(0)}$
end while
for $i = 1, \ldots,$ `it.max` **do**
  Fit l models with varying parameters and calculate their AIC
  Find the minimum AIC and its corresponding parameter $\rho_{\min}^{(i)}$
  if $\rho_{\min}^{(i)} = \rho_L^{(i-1)}$ **then**
    Update the range with extrema $\frac{1}{c} \cdot \rho_L^{(i-1)}$ and $\frac{1}{c} \cdot \rho_R^{(i-1)}$, where $c > 0$ is some defined constant
  else if $\rho_{\min}^{(i)} = \rho_R^{(i-1)}$ **then**
    Update the range with extrema $c \cdot \rho_L^{(i-1)}$ and $c \cdot \rho_R^{(i-1)}$, where $c > 0$ is some defined constant
  else
    Update the extrema for the new range to be centred around $\rho_{\min}^{(i)}$
  end if
  while $\mathbf{S} + \rho_L^{(i)}\mathbf{I}$ is not positive definite **do**
    Increase $\rho_L^{(i)}$
  end while
end for

Algorithm 3 gives the outline of the model selection procedure and it is implemented slightly differently according to the learning method, each having different output structure. We remark that for the Reparametrization JGL the check on positive definiteness is made only once at the beginning of

the algorithm with $\mathbf{S}+\nu\mathbf{I}$, as ν is not subject to tuning. Moreover, we mention that even though the algorithm converges in a finite number of iterations, we fix a maximum number `it.max`, since fitting l models at each step can be very time-consuming.

The `R` functions can be found in Appendix B as `AIC_sep_lasso()`, `AIC_guo()` and `AIC_jgl()`. For the Direct learning method, we kept the implementation for the AIC selection as set in the `dpm()` function.

While the application of this procedure operates automatically regardless of the scenario, in terms of relationship between p and n and of dissimilarity degree given by δ, we do give a slight computational advantage to the AIC selection algorithm by setting the initial ranges higher when $p > n$. The comprehensive indication of the parameter settings employed in the model selection procedure described in Algorithm 3 for our set of simulations can be found in Table A.3.

4.7 Performance Evaluation

In this Section, we display the criteria for comparing the models fitted using each of the methods discussed in Section 4.5.

The evaluation of the performance of each method consists in sets of indexes aimed at understanding how accurately each graphical model is able to reconstruct the underlying true model, which is known in the simulation setting.

In order to do construct these indexes, we preface by defining what we will refer to as positive and negative values. We consider as *positive* an entry of the concentration matrix that is different from zero or, in other words, a present edge in the corresponding graph. The complementary definition for *negative* outcomes refers to entries of the concentration matrix that are equal to zero, i.e. corresponding to missing edges in the graph.

We can therefore construct a contingency table for the true underlying graph and the estimated one, regardless of the method, as shown in Table 4.4.

Estimated \ Population	Positive	Negative
Positive	TP	FP
Negative	FN	TN

Table 4.4: Contingency table for comparing two graphs

For instance, the true positives are the edges that are present both in the population graph and the in the estimated one. Due to the symmetry of the concentration matrices, we evaluate these measures only on their upper triangular portion, diagonal excluded. More formally,

$$TP = |\{(i,j) : \omega_{ij} \neq 0 \wedge \hat{\omega}_{ij} \neq 0; i,j \in V, i < j\}|$$

Traditionally, these measures are combined in the specificity (or True Negative Rate) and sensitivity (True Positive Rate) indexes:

$$TNR = \frac{TN}{TN + FP} \tag{4.1}$$

$$TPR = \frac{TP}{TP + FN} \tag{4.2}$$

However, in this context, the graphs can be very sparse, which results in a large number of true negatives. Therefore, the specificity of the estimates may not be an informative measure, since the positive values may be a very small proportion of the entries, hence inflating the index regardless of the learning method. In light of this issue, it would be appropriate to consider an alternative measure that is not as influenced by the amount of true negative results.

For this reason, we refer to the precision (or Positive Predictive Value) and recall (TPR) indexes. The former is defined as:

$$PPV = \frac{TP}{TP + FP} \tag{4.3}$$

This quantifies the proportion of estimated positive values that are truly positive. On the other hand, the recall measure is equivalent to the sensitivity (TPR) and it measures the proportion of population positive values that is correctly estimated as positive by the learning method.

In addition to the precision and recall, we consider as an auxiliary indicator for the performance of each method the comparison between the density of the population graph and the estimated graph. We define as *Graph Density* (GD) the proportion of present edges in a graph. More formally:

$$GD = \frac{|\{(i,j) : \omega^*_{ij} \neq 0; i,j \in V, i < j\}|}{p(p-1)/2} \tag{4.4}$$

Where ω^*_{ij} corresponds either to the population concentration matrix or to the estimated one. In the first case, when $\omega^*_{ij} = \omega_{ij}$, we label this index as

Population Graph Density (PGD). On the other hand, when $\omega_{ij}^* = \hat{\omega}_{ij}$, we refer to this index as Estimated Graph Density (EGD).

We now discuss the purpose of these indexes. As presented in Chapter 3, there are different strategies to learning multiple graphs. We believe that the indexes evaluating the performance of the methods should take into consideration the variety of learning strategies. Therefore, we distinguish three categories of performance evaluation: separate, joint and differential indexes.

Indexes for Separate Graph Structure Learning

The first set of indexes that we consider is aimed at evaluating the ability to correctly learn each of the graphs as if they were considered separately. This can be suited to methods that do not borrow any strength from the common structure of the graphs, such as the Separate Graphical Lasso method. However, we also evaluate these indexes for the joint learning methods (Reparametrization, Fused and Group Joint Graphical Lasso).

The way that these indexes are calculated consists in disregarding any commonality between the two groups and evaluating the overall precision and recall. For instance, we compute the precision as:

$$PPV = \frac{TP_1 + TP_2}{(TP_1 + FP_1) + (TP_2 + FP_2)}$$

Using the same approach, we evaluate the recall.

We remark that these indexes cannot be computed for the output produced by the Direct Estimation method, as we do not have any information on the estimates of the single concentration matrices.

As for the GD indexes, we consider as an aggregate measure the average GD over the H conditions, both for the population and estimated graph. For instance, the Population Average Graph Density will be computed as:

$$PAGD = \frac{1}{H} \sum_{h=1}^{H} \frac{|\{(i,j) : \omega_{ij}^{(h)} \neq 0; i,j \in V, i < j\}|}{p(p-1)/2} \tag{4.5}$$

The EAGD is computed accordingly based on the estimated concentration matrices.

Indexes for Joint Graph Structure Learning

As opposed to evaluating the indexes separately, we consider the performance of the methods by calculating them on the graphs that consider the common and different structure between the two groups.

In order to analyse the common structure of the graphs, we consider the evaluation of the indexes on the set of edges that are present in both conditions, which we refer to as *matching present*. Formally, the graph of matching present edges can be defined as $\mathcal{G}^*_{mp} = (V, E^*_{mp})$, where $E^*_{mp} = \{(i,j) : \omega^{*(1)}_{ij} \neq 0 \wedge \omega^{*(2)}_{ij} \neq 0; i,j \in V; i < j\}$. For $\omega^{*(h)}_{ij} = \omega^{(h)}_{ij}\ \forall h$ we have the corresponding population graph and $\omega^{*(h)}_{ij} = \hat{\omega}^{(h)}_{ij}\ \forall h$ corresponds to the estimated one.

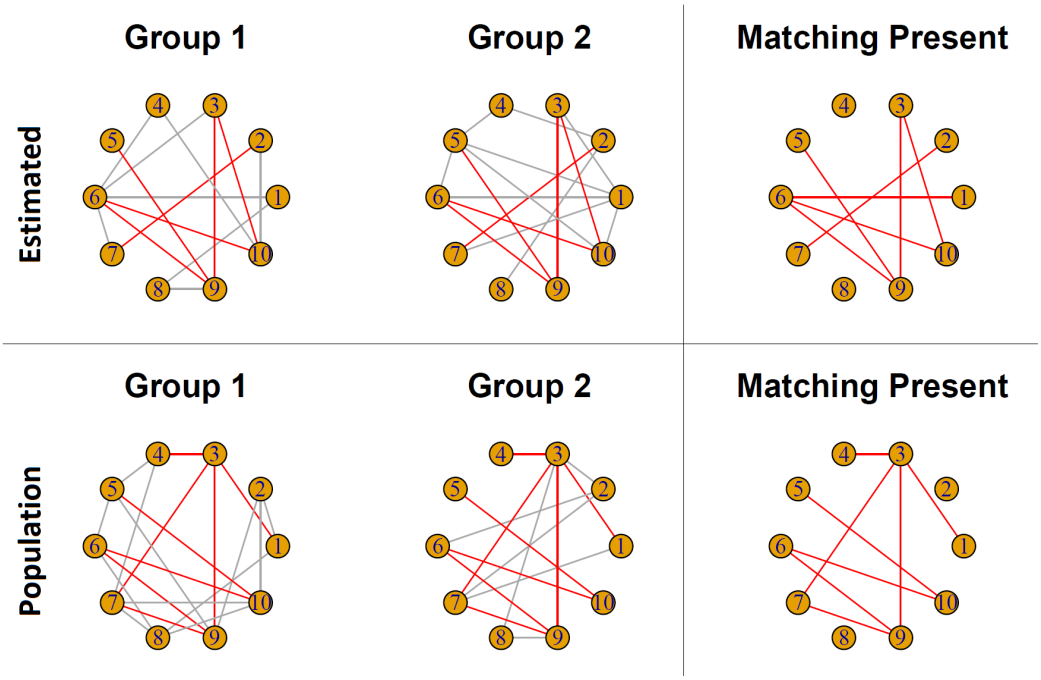


Figure 4.10: Estimated and population graphs of matching present edges (in red)

Figure 4.10 shows the graphs of matching present edges between the two estimated graphs (above) and the two population ones (below). Once the graph of matching present edges is built the PPV and TPR indexes are evaluated.

In this case, a positive value is an edge that is present in both conditions. As a consequence, the TPR is to be interpreted as the portion of the edges present in both population groups that are estimated as present in both

groups by the learning method under analysis. The same reasoning applies to the PPV index.

The second part of this evaluation consists in considering the edges that differ in presence or absence between the two conditions, which we refer to as *mismatching edges*. We define this as $\mathcal{G}_m^* = (V, E_m^*)$, where $E_m^* = \{(i,j) : (\omega_{ij}^{*(1)} = 0 \wedge \omega_{ij}^{*(2)} \neq 0) \vee (\omega_{ij}^{*(1)} \neq 0 \wedge \omega_{ij}^{*(2)} = 0); i, j \in V; i < j\}$. The population and estimated variants are retrieved with $\omega_{ij}^{*(h)} = \omega_{ij}^{(h)}\ \forall h$ and $\omega_{ij}^{*(h)} = \hat{\omega}_{ij}^{(h)}\ \forall h$ respectively. Figure 4.11 shows the construction of graphs of mismatching edges.

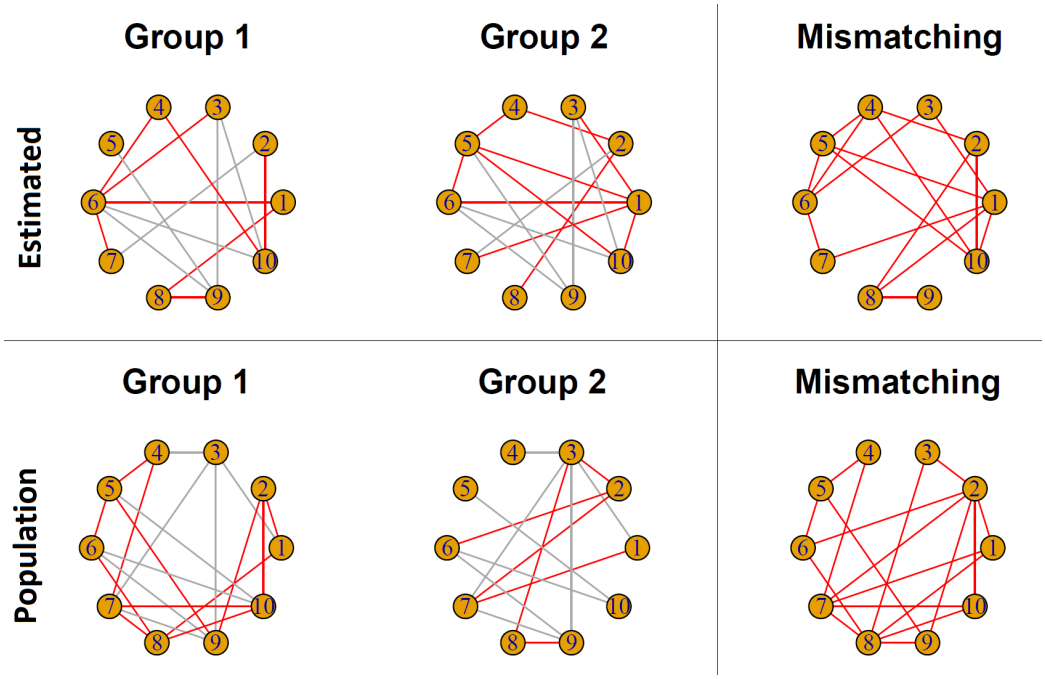


Figure 4.11: Estimated and Population graphs of mismatching edges (in red)

The PPV and TPR are then evaluated on the mismatching graph. For instance, then, the PPV on mismatching edges corresponds to the proportion of estimated mismatching edges that are truly mismatching in the population graphs.

It should be kept in mind that these indexes cannot be computed for the Direct Estimation method, as we do not have any information on each of the graphs. We compute these for the JGSL methods as well as the Separate Graphical Lasso. In theory, we would expect the separate learning method to show worse performance than the joint ones for these indexes, as it is not

built with the consideration of accounting for the common structure of the graphs.

We remark that the evaluation of the Graph Density is equivalent to that of Separate Graph Structure Learning. Therefore, we retrieve the Population Average Graph Density as in Equation (4.5) and the Estimated one accordingly.

Indexes for Differential Graph Structure Learning

This last set of indexes refers to the ability of learning the graphs of differences. It is suited to evaluating the performance of the Direct estimation method by [Zhao et al., 2014], where the graphs are learnt based on $\mathbf{S}^{(1)} - \mathbf{S}^{(2)}$.

We perform this evaluation also on the other learning methods, with the awareness that this conveys a slightly different information than considering the matching and mismatching edges as done for the JGSL indexes. We use the term *different edges* to indicate a non-zero difference in edge values. Using the definition in Section 3.4, the graphs of differences are $\mathcal{G}_d^* = (V, E_d^*)$, with $E_d^* = \{(i,j) : \omega_{ij}^{*(1)} - \omega_{ij}^{*(2)} \neq 0; i, j \in V; i < j\}$. The population and estimated ones correspond again to $\omega_{ij}^{*(h)} = \omega_{ij}^{(h)}\ \forall h$ and $\omega_{ij}^{*(h)} = \hat{\omega}_{ij}^{(h)}\ \forall h$ respectively.

As we see from Figure 4.12, the graph constructed from the non-zero entries of the difference between concentration matrices is denser than that produced by the discrepancy in absence/presence of edges that was shown in Figure 4.11. This is due to the fact that while the edges may be present in both conditions, the corresponding element in the concentration matrices may be different, thus resulting in an edge in the right-hand side of Figure 4.12.

We make the distinction between these two types of graphs due to the fact that for the Direct Estimation method we only have an estimate $\hat{\mathbf{\Delta}}$ for $\mathbf{\Delta}_0 = \mathbf{\Omega}^{(1)} - \mathbf{\Omega}^{(2)}$. This is not directly comparable to the other methods, which produce the single estimates $\hat{\mathbf{\Omega}}^{(1)}$ and $\hat{\mathbf{\Omega}}^{(2)}$. However, for the other learning methods, we can obtain a comparison by using the difference of the estimates: $\hat{\mathbf{\Delta}} = \hat{\mathbf{\Omega}}^{(1)} - \hat{\mathbf{\Omega}}^{(2)}$.

In regard to the interpretation of the PPV and TPR for edges in graphs of differences, we explain as an example that $PPV = 0.25$ would mean that among the elements of the concentration matrices that are estimated to have different value between the two conditions, 25% of them have truly different value in the population.

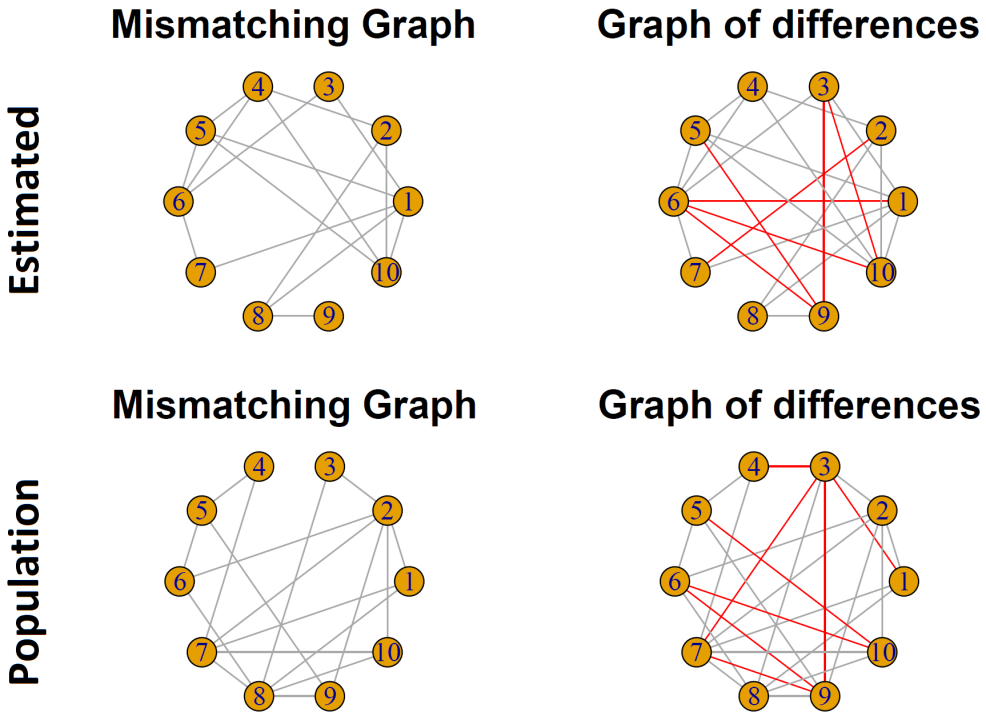


Figure 4.12: Comparison between mismatching graphs (left) and graphs of differences (right). The additional edges resulting from different edge values are shown in red.

It should be noted that in the graphs obtained by differences in edge values, the index for the Graph Density does not need averaging, since there is directly one graph of differences. Therefore, the PGD and EGD are calculated as in Equation (4.4), with the notion that the concentration matrices of reference are in this case $\mathbf{\Delta}_0$ for the PGD and $\hat{\mathbf{\Delta}}$ for the EGD.

The interpretation of these measures varies consequently: in this case, an edge means a different partial correlation and therefore $PGD = 0.1$ would mean that 10% of the entries of the upper triangular concentration matrices have different values.

Table 4.5 shows a summary of the indexes used to evaluate the performance of the methods that produce as output each of the concentration matrices. Table 4.6, on the other hand, refers to the indexes computed on the difference of concentration matrices.

Table 4.5: Summary of performance measures for separate and joint learning methods, [Danaher et al., 2014])

SEPARATE INDEXES		
	PPV	Proportion of correctly present edges among the estimated ones
	TPR	Proportion of correctly present edges among the truly present
JOINT INDEXES		
MATCHING PRESENCE:	PPV	Proportion of correctly matching present edges among the estimated ones
MATCHING PRESENCE:	TPR	Proportion of correctly matching present edges among the truly matching present
MISMATCHING:	PPV	Proportion of correctly mismatching edges among the estimated ones
MISMATCHING:	TPR	Proportion of correctly mismatching edges among the truly mismatching
DENSITY		
	EAGD	Average proportion of estimated present edges
	PAGD	Average proportion of truly present edges

Table 4.6: Summary of performance measures for graph of differences

DIFFERENTIAL INDEXES		
DIFFERENCE:	PPV	Proportion of correctly different edges among the truly different ones
DIFFERENCE:	TPR	Proportion of correctly different edges among the truly different ones
DENSITY		
	EGD	Proportion of non-zero edges in the estimated graph of differences
	PGD	Proportion of non-zero edges in the true graph of differences

4.8 Results

In this section, we show the results for the set of simulations run as described in Sections 4.2-4.7. The analysis of the results for each scenario includes a

large amount of tables whose features are here summarized by reporting some exemplary ones. The totality of the performance results can be found in the Appendix (Tables A.4-A.108).

For each scenario, the simulations were run 20 times. Therefore, the results that we show are the averages of the 20 repetitions. For each index, we also report its standard deviation in brackets, calculated with respect to the 20 repetitions.

In the comparison of the results, the references to each method will be done using the following abbreviations:

- Sep-GL: Separate Graphical Lasso (Section 3.2),
- R-JGL: Reparametrization Joint Graphical Lasso (Section 3.3.1),
- Fused JGL: Fused Joint Graphical Lasso (Section 3.3.2),
- Group JGL: Group Joint Graphical Lasso (Section 3.3.2),
- Direct: Direct Estimation method (Section 3.4.1).

The first simulation scenario we report refers to the case $p = 30,\ n = 200$, with dissimilarity degree $\delta = 0.20$. Table 4.7 shows the performance of the learning methods that produce estimates of each concentration matrix constructed on the random graph model, with $\pi = 0.1$ probability of edge presence in the common structure. The first piece of information we retrieve from Table 4.7 is that the four methods produce graphs with varying average density, from 0.351 (Fused JGL) to 0.608 (Sep-GL). None of them, however, is close to the true one. The JGSL methods have a higher precision than the separate one, but the Sep-GL shows higher recall. This is influenced by the estimated average densities, since identifying denser graphs results in estimating more positive values. Overall, it seems that the Fused JGL and the R-JGL are able to reconstruct the concentration matrices based on the random graphs most closely to the population ones. The former shows a higher precision on mismatching edges, but the latter is more precise in determining the matching present ones and in the graphs overall when considered separately.

To follow this evaluation, we report the results for learning tree graphs in the same simulation scenario ($p = 30,\ n = 200$), but with a smaller portion of changed edges, $\delta = 0.05$. Table 4.8 shows that the average density of the true graphs is much lower, which some of the learning methods are not able to correctly identify. For instance, the Sep-GL estimates around 28% of present

Table 4.7: Performance on random graphs ($\pi = 0.1$), p=30, n=200, $\delta = 0.20$

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.351 (0.025)	0.473 (0.049)	0.457 (0.041)	0.432 (0.040)
TPR	0.807 (0.042)	0.642 (0.061)	0.600 (0.090)	0.651 (0.062)
MATCHING PRESENCE: PPV	0.184 (0.040)	0.249 (0.058)	0.245 (0.055)	0.219 (0.051)
MATCHING PRESENCE: TPR	0.677 (0.088)	0.594 (0.103)	0.646 (0.102)	0.606 (0.090)
MISMATCHING: PPV	0.330 (0.029)	0.449 (0.049)	0.575 (0.084)	0.466 (0.043)
MISMATCHING: TPR	0.456 (0.049)	0.303 (0.030)	0.238 (0.077)	0.307 (0.048)
EAGD	0.608 (0.057)	0.363 (0.058)	0.351 (0.074)	0.400 (0.056)
PAGD	0.264 (0.014)	0.264 (0.014)	0.264 (0.014)	0.264 (0.014)

edges, as opposed to the true 6.6%. Once again, the Fused JGL shows higher precision than the other methods, but clearly the recall is lower, especially on mismatching edges; by identifying sparser graphs than the other methods, it results in estimating fewer positive values.

Furthermore, it can be seen how overall all the methods have worse performance with respect to Table 4.7, which may be due mainly to the lower average density of the underlying population graphs.

Table 4.8: Performance on tree graphs, p=30, n=200, $\delta = 0.05$

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.151 (0.022)	0.245 (0.063)	0.507 (0.178)	0.349 (0.099)
TPR	0.630 (0.096)	0.583 (0.083)	0.331 (0.131)	0.414 (0.115)
MATCHING PRESENCE: PPV	0.147 (0.048)	0.186 (0.082)	0.362 (0.221)	0.267 (0.127)
MATCHING PRESENCE: TPR	0.503 (0.164)	0.549 (0.146)	0.428 (0.186)	0.414 (0.172)
MISMATCHING: PPV	0.114 (0.030)	0.177 (0.073)	0.652 (0.250)	0.308 (0.112)
MISMATCHING: TPR	0.525 (0.071)	0.325 (0.055)	0.097 (0.044)	0.264 (0.078)
EAGD	0.280 (0.057)	0.167 (0.048)	0.051 (0.030)	0.086 (0.037)
PAGD	0.066 (0.001)	0.066 (0.001)	0.066 (0.001)	0.066 (0.001)

When it comes to learning graphs with dimension larger than the samples size, we notice a common pattern for $p = 30,\ n = 20$, which we report, for instance, in Tables 4.9 and 4.10. In both the Markov-chain graphs and the scale-free ones, the Fused JGL and Group JGL methods estimate lower average densities than the true ones, whereas the R-JGL estimates greater ones. Furthermore, it can be seen that the regularization parameters estimated by the R-JGL are more variable than the other methods, presenting larger standard deviation for the average densities.

These examples from the $p = 30,\ n = 20$ scenario show that the Sep-GL is the most suitable candidate to reproduce graphs with average densities

Table 4.9: Performance on Markov-chain graphs, p=30, n=20, δ=0.05

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.171 (0.050)	0.168 (0.046)	0.215 (0.329)	0.195 (0.350)
TPR	0.174 (0.076)	0.285 (0.222)	0.064 (0.141)	0.009 (0.018)
MATCHING PRESENCE: PPV	0.151 (0.176)	0.123 (0.082)	0.234 (0.365)	0.017 (0.073)
MATCHING PRESENCE: TPR	0.053 (0.054)	0.225 (0.241)	0.075 (0.155)	0.002 (0.009)
MISMATCHING: PPV	0.127 (0.046)	0.125 (0.033)	0.029 (0.080)	0.029 (0.073)
MISMATCHING: TPR	0.240 (0.111)	0.159 (0.058)	0.004 (0.009)	0.008 (0.023)
EAGD	0.116 (0.060)	0.215 (0.221)	0.043 (0.107)	0.004 (0.008)
PAGD	0.108 (0.005)	0.108 (0.005)	0.108 (0.005)	0.108 (0.005)

comparable to the true ones. However, it should be noted that the R-JGL shows a precision comparable to the former and with higher recall. This may indicate that if the R-JGL were able to identify a regularization parameter that reproduces the correct graph densities, it could be more precise in identifying the true graphs.

Table 4.10: Performance on scale-free graphs, p=30, n=20, δ=0.05

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.153 (0.043)	0.158 (0.043)	0.182 (0.275)	0.319 (0.348)
TPR	0.188 (0.081)	0.337 (0.260)	0.023 (0.042)	0.016 (0.018)
MATCHING PRESENCE: PPV	0.180 (0.126)	0.117 (0.066)	0.167 (0.266)	0.192 (0.347)
MATCHING PRESENCE: TPR	0.076 (0.054)	0.282 (0.264)	0.034 (0.058)	0.014 (0.025)
MISMATCHING: PPV	0.120 (0.037)	0.126 (0.062)	0.025 (0.109)	0.162 (0.307)
MISMATCHING: TPR	0.255 (0.102)	0.151 (0.070)	0.001 (0.005)	0.013 (0.022)
EAGD	0.131 (0.051)	0.274 (0.269)	0.009 (0.025)	0.005 (0.006)
PAGD	0.108 (0.004)	0.108 (0.004)	0.108 (0.004)	0.108 (0.004)

Since the graphs analysed and reported in Tables 4.9 and 4.10 have equal average densities, it is interesting to investigate whether there is some remarkable difference in estimation only depending on the graph structure. For instance, we gather that the Sep-GL and the R-JGL have similar performance among the two graph types, with larger TPR in the scale-free graphs due to larger estimated densities. In this occurrence, the Fused JGL is more suited to the Markov-chain graph, whereas the Group JGL is preferable for the scale-free ones. These differences, however, are very subtle and we cannot conclude that the fitting is remarkably different between these two structures. Throughout the set of simulations, we do not witness a significant distinction in the performance of the learning methods, which can be assessed by inspecting the Tables in Appendix A.2.1.

When the dimension of the problem increases, such as the $p = 100$ scenarios, the Sep-GL loses its ability to closely estimate the regularization parameters reproducing the true graph densities. Since the densities reproduced by this method are either much larger or much smaller than the true ones, this indicates that the model selection procedure is not able to identify a correct value for ρ, either shrinking too much or too little the model parameters. By analysing the Tables in Appendix A.2.1 for large p and $n >> p$, it is not straightforward to conclude whether one method is evidently preferable to the other ones since there is no remarkable peak in performance from any of them. The main difference lies in the better ability of the R-JGL and Group JGL methods to retrieve graph densities comparable to the population ones, followed closely by the Fused JGL.

We show for instance the results on the scale-free graphs, for $p = 100$, $n = 400$, $\delta = 0.05$. As we see from Table 4.11, the Group JGL is able to best identify graphs with EAGD close to the true ones; the Sep-GL and the R-JGL produce denser ones and the Fused JGL sparser ones. The Fused JGL has better ability in identifying matching present edges, but it is not ideal for mismatching edges, for which it presents lowest recall, as seen in the previous occasions.

In what can be considered the worst-case scenario, i.e. when the dimension of the data is large and the sample size is smaller ($p = 100$, $n = 60$), the learning methods appear to have weaker ability to reconstruct the true graphs. Only the R-JGL is able to recover an EAGD that is close to the true one, whereas the other methods identify extremely sparse graphs, which compromises their whole performance. As previously mentioned, when the dimension of the problem is greater than the sample size, the Fused JGL and Group JGL estimate larger regularization parameters than needed and

Table 4.11: Performance on scale-free graphs, p=100, n=400, $\delta = 0.05$

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.254 (0.015)	0.320 (0.014)	0.503 (0.043)	0.378 (0.035)
TPR	0.555 (0.031)	0.533 (0.017)	0.324 (0.049)	0.394 (0.044)
MATCHING PRESENCE: PPV	0.304 (0.035)	0.297 (0.019)	0.409 (0.044)	0.338 (0.035)
MATCHING PRESENCE: TPR	0.399 (0.033)	0.475 (0.024)	0.397 (0.055)	0.373 (0.051)
MISMATCHING: PPV	0.143 (0.008)	0.176 (0.010)	0.538 (0.099)	0.248 (0.030)
MISMATCHING: TPR	0.501 (0.030)	0.312 (0.020)	0.074 (0.019)	0.233 (0.028)
EAGD	0.265 (0.026)	0.201 (0.008)	0.079 (0.016)	0.128 (0.024)
PAGD	0.121 (0.001)	0.121 (0.001)	0.121 (0.001)	0.121 (0.001)

when the dimension is large, this happens for the Sep-GL as well. This can be assessed thoroughly from the Tables in Appendix A.2.1.

We show, for instance, the attempt at learning tree graphs, with dissimilarity degree $\delta = 0.20$ (Table 4.12). In this instance, it is evident that none of the methods are able to give plausible estimates of the population graphs, identifying at best 10% of the positive values.

Table 4.12: Performance on tree graphs, p=100, n=60, $\delta = 0.20$

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.046 (0.017)	0.042 (0.010)	0.025 (0.109)	0.078 (0.127)
TPR	0.075 (0.033)	0.129 (0.035)	0.000 (0.001)	0.007 (0.012)
MATCHING PRESENCE: PPV	0.006 (0.018)	0.009 (0.007)	0.000 (0.000)	0.000 (0.000)
MATCHING PRESENCE: TPR	0.009 (0.027)	0.098 (0.064)	0.000 (0.000)	0.000 (0.000)
MISMATCHING: PPV	0.061 (0.019)	0.056 (0.015)	0.000 (0.000)	0.094 (0.149)
MISMATCHING: TPR	0.102 (0.052)	0.112 (0.028)	0.000 (0.000)	0.008 (0.013)
EAGD	0.034 (0.016)	0.062 (0.018)	0.000 (0.001)	0.001 (0.002)
PAGD	0.020 (0.000)	0.020 (0.000)	0.020 (0.000)	0.020 (0.000)

Graphs of differences.

Since the Direct method learns the graphs based on differences between edge values in the two conditions, its results cannot be comparable to the estimates produced by the other methods for multiple reasons. First of all, there can be no distinction between edges that are both missing or both present with the same value under the the two conditions. Moreover, the edges in the graphs of differences are present even when they are present in both conditions, but with different value. As discussed in Section 4.7, we compare the estimates $\hat{\mathbf{\Delta}}$ with the estimates $\hat{\mathbf{\Omega}}^{(1)} - \hat{\mathbf{\Omega}}^{(2)}$ produced by non-differential learning methods.

We begin by showing the scenario $p = 30,\ n = 200$. Table 4.13 displays as an example scale-free graphs, with dissimilarity degree $\delta = 0.20$. As we see, the Sep-GL and the Direct methods estimate regularization parameters producing denser graphs in terms of differences, with values being double the true densities. On the other hand, the Fused JGL identifies sparser graphs. The two methods that are able to produce adequate graphs are the R-JGL and the Group JGL. In the scenario example considered in Table 4.13, the R-JGL is best performing, not only reproducing graphs of differences close in density to the true ones, but also with a good precision and recall on different edges. From the Tables in Appendix A.2.2, the same conclusions can be drawn for all graph structures.

Table 4.13: Performance on differences of scale-free graphs, p=30, n=200, δ=0.20

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.443 (0.018)	0.601 (0.041)	0.689 (0.056)	0.541 (0.039)	0.413 (0.031)
DIFFERENCE: TPR	0.891 (0.047)	0.665 (0.061)	0.296 (0.067)	0.677 (0.051)	0.830 (0.260)
EGD	0.797 (0.052)	0.440 (0.055)	0.173 (0.047)	0.498 (0.056)	0.811 (0.270)
PGD	0.396 (0.011)	0.396 (0.011)	0.396 (0.011)	0.396 (0.011)	0.396 (0.011)

The behaviour of the Direct method changes when the sample size decreases, for instance with $p = 30$ and $n = 40$. It is able to estimate the graph densities more closely, though with large variability. Moreover, it is worth noting that when $\delta = 0.05$, the Direct method tends to learn graphs of differences that are closer in density to the true ones than when $\delta = 0.20$.

Table 4.14 shows an example for the random graphs in this simulation scenario, with probability $\pi = 0.1$ of edge presence in the graph generation and probability $\delta = 0.05$ of edges changing between the two conditions. As we see from Table 4.14, the Direct method has the closest EGD to the population one. However, it still appears that the R-JGL produces better estimates, having higher precision and recall on different edges. The other methods either present estimates than are too sparse (Fused and Group JGL) or too dense (Sep-GL).

For large dimensions with larger sample size ($p = 100,\ n = 400$), we observe similar behaviours as the case for low dimensions ($p = 30,\ n = 200$). The R-JGL and the Group JGL are preferable, with similar performance both in terms of estimated densities and in terms of precision and recall evaluated on different edges. The Sep-GL produces denser graphs of differences than the population ones, whereas the Fused JGL and the Direct estimate sparser ones.

Table 4.14: Performance on differences of random graphs ($\pi = 0.1$), p=30, n=40, δ=0.05

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.251 (0.037)	0.289 (0.049)	0.312 (0.323)	0.356 (0.289)	0.108 (0.104)
DIFFERENCE: TPR	0.495 (0.123)	0.400 (0.086)	0.019 (0.021)	0.104 (0.105)	0.176 (0.256)
EGD	0.377 (0.113)	0.261 (0.063)	0.009 (0.011)	0.050 (0.059)	0.166 (0.235)
PGD	0.186 (0.014)	0.186 (0.014)	0.186 (0.014)	0.186 (0.014)	0.186 (0.014)

We consider as an example the differences of random graphs, with $\delta = 0.05$ (Table 4.15). In this instance, the Direct method is not able to identify graphs of differences similar to the population ones, showing and average density of 0.2%. It would appear, then, the Direct method is not favoured by sample sizes larger than the dimension of the graph.

Table 4.15: Performance on differences of random graphs ($\pi = 0.1$), p=100, n=400, δ=0.05

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.280 (0.017)	0.370 (0.015)	0.573 (0.086)	0.404 (0.028)	0.076 (0.171)
DIFFERENCE: TPR	0.673 (0.048)	0.578 (0.019)	0.091 (0.021)	0.457 (0.052)	0.003 (0.010)
EGD	0.451 (0.053)	0.291 (0.011)	0.031 (0.009)	0.213 (0.037)	0.002 (0.008)
PGD	0.187 (0.005)	0.187 (0.005)	0.187 (0.005)	0.187 (0.005)	0.187 (0.005)

To conclude, we explore the scenario for $p = 100$ and $n = 60$. It is confirmed that the Fused JGL and Group JGL do not perform adequately when the dimension exceeds the sample size. In this case as well, they produce estimates that are too sparse with respect to the true graphs. The R-JGL is to be preferred throughout this simulation scenario, as confirmed by the corresponding Tables in Appendix A.2.2.

For instance, we show the results for the difference of Markov-chain graphs, with $\delta = 0.05$, as displayed in Table 4.16. The Direct method again shows better performance than in the case with $n > p$. However, we remark that this method has the largest variability in the estimates and shows inconsistent performance as the graph structures vary.

4.8.1 Fixing the density

The main observation that can be drawn from the results of our set of simulations is that in most cases the methods are not able to reproduce the

Table 4.16: Performance on differences of Markov-chain Graph, p=100, n=60, δ=0.05

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.178 (0.021)	0.180 (0.020)	0.000 (0.000)	0.181 (0.226)	0.063 (0.070)
DIFFERENCE:TPR	0.110 (0.030)	0.168 (0.034)	0.000 (0.000)	0.003 (0.006)	0.162 (0.249)
EGD	0.070 (0.019)	0.106 (0.026)	0.000 (0.000)	0.001 (0.002)	0.164 (0.251)
PGD	0.112 (0.001)	0.112 (0.001)	0.112 (0.001)	0.112 (0.001)	0.112 (0.001)

correct densities of the graphs. This clearly impacts on their performance, as both the precision and the recall measures depend on the density itself. Clearly we cannot have a high precision for graphs that are denser than the population ones and viceversa we cannot obtain a large recall for sparser graphs. If we wish to compare the performance of each method without the impact of the density, we can impose the regularization parameters to be as close as possible to those that reproduce the true density.

In order to do so, we use a modified model selection procedure with respect to Algorithm 3, discussed in Section 4.6.1. Instead of choosing the parameter that minimizes the AIC, we choose the parameter that minimizes $|EAGD - PAGD|$:

Algorithm 4: Select regularization parameters that reproduce the true graph densities

Initialize extrema for the parameter range $\rho_L^{(0)}$ and$\rho_R^{(0)}$ and generate a vector of length l of equally-spaced values between $\rho_L^{(0)}$ and $\rho_R^{(0)}$
for $i = 1, \ldots,$ `it.max` **do**
  Fit l models and calculate their $EAGD$
  Find the minimum $|EAGD - PAGD|$ and its corresponding parameter $\rho_{\min}^{(i)}$
  if $\rho_{\min}^{(i)} = \rho_L^{(i-1)}$ **then**
    Update the range with extrema $\frac{1}{c} \cdot \rho_L^{(i-1)}$ and $\frac{1}{c} \cdot \rho_R^{(i-1)}$, where $c > 0$ is some defined constant
  else if $\rho_{\min}^{(i)} = \rho_R^{(i-1)}$ **then**
    Update the range with extrema $c \cdot \rho_L^{(i-1)}$ and $c \cdot \rho_R^{(i-1)}$, where $c > 0$ is some defined constant
  else
    Update the extrema for the new range to be centred around $\rho_{\min}^{(i)}$
  end if
end for

Algorithm 4 is implemented for each of the learning methods. The corresponding functions `rho_finder_sep_lasso()`, `rho_finder_guo()` and `rho_finder_jgl()` can be found in Appendix B. As opposed to the results reported in Section 4.8, the performance of the methods in this Section does not include information on the standard deviation of the indexes, since the regularization parameters reproduce the population densities with very little fluctuation by construction.

We report, for instance, the results on Markov-chain graphs, for $p = 30,\ n = 40$ and with $\delta = 0.20$, in Table 4.17. The Fused JGL produces graphs for which the true positive rate for mismatching edges is lower than the other methods, but higher for matching present edges. The Group JGL shows similar characteristics, but less emphasized. The opposite behaviour goes for the Sep-GL: highest TPR on mismatching, but lowest on matching present. Though these traits were already detected in Section 4.8, it is confirmed that it is not only a consequence of the different estimated densities, but of the learning methods themselves.

As a common feature that can be observed from the Tables in Appendix A.3.1, the R-JGL shows the best performance on the graphs considered separately, which is also confirmed in Table 4.17.

Table 4.17: Performance on Markov-chain graphs with fixed densities, p=30, n=40, δ=0.20

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.361	0.377	0.362	0.360
TPR	0.354	0.392	0.363	0.363
MATCHING PRESENCE: PPV	0.276	0.209	0.180	0.149
MATCHING PRESENCE: TPR	0.242	0.424	0.485	0.333
MISMATCHING: PPV	0.393	0.465	0.400	0.394
MISMATCHING: TPR	0.404	0.274	0.096	0.178
EAGD	0.239	0.253	0.245	0.246
PAGD	0.244	0.244	0.244	0.244

Another case we consider is the scenario where $p = 100$ and $n = 400$. Table 4.18 shows the results for random graphs with $\delta = 0.05$. It is interesting to compare these results to the same scenario with AIC selection, as in Table 4.11. From this comparison, it is evident that the performance indexes highly depend on the estimated density. For the Sep-GL and the R-JGL, in Table 4.18 the TPR is lower, but the PPV is higher. The opposite goes for the Fused JGL.

Table 4.18: Performance on scale-free graphs with fixed densities, p=100, n=400, δ=0.05

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.387	0.418	0.403	0.402
TPR	0.389	0.435	0.406	0.400
MATCHING PRESENCE: PPV	0.547	0.392	0.309	0.353
MATCHING PRESENCE: TPR	0.244	0.361	0.457	0.370
MISMATCHING: PPV	0.202	0.250	0.366	0.280
MISMATCHING: TPR	0.373	0.305	0.110	0.256
EAGD	0.120	0.125	0.121	0.119
PAGD	0.120	0.120	0.120	0.120

In the same simulation scenario, we explore the ability of the learning methods to identify the correct edge patterns for graphs with lower densities, which is the case for tree graphs. Table 4.19 shows this as an example for $p = 100,\ n = 400$ and dissimilarity degree $\delta = 0.05$. Although the EAGDs are equal to the population ones, by comparing Table 4.19 with Table 4.18, it is clear that the performance is overall weaker for all the learning methods. The R-JGL still shows better overall precision and recall, especially on the graphs considered separately.

From this observation, it is confirmed that not only a low density is hard to identify, but it also negatively impacts on the overall graph structure learning.

Table 4.19: Performance on tree graphs with fixed densities, p=100, n=400, δ=0.05

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.216	0.257	0.151	0.214
TPR	0.212	0.268	0.141	0.217
MATCHING PRESENCE: PPV	0.200	0.138	0.057	0.093
MATCHING PRESENCE: TPR	0.091	0.364	0.455	0.364
MISMATCHING: PPV	0.196	0.236	0.556	0.200
MISMATCHING: TPR	0.205	0.199	0.028	0.131
EAGD	0.020	0.021	0.019	0.020
PAGD	0.020	0.020	0.020	0.020

Graphs of Differences.

In regard to the comparison of learning methods for graphs of differences, we use a modification of Algorithm 4 to choose the regularization parameters,

by minimizing $|EGD - PGD|$ on the graphs of differences instead. The corresponding `R` functions can be found in Appendix B as `rho_finder_diff()` and `rho_finder_jgl_diff()`.

Figure 4.13 shows a comparison of the estimated differences of tree graphs for the $p = 30$, $n = 200$ simulation scenario, with $\delta = 0.05$. Although the regularization parameters are set to be reproducing the true density of the graphs of differences, we see that the graphs are not that similar. In particular, the graph identified with the Direct method tends to find a few nodes with large connectivity (e.g. nodes 3 and 26), leaving many nodes unconnected. The Sep-GL and Group JGL seem to select very similar models, which is also reflected in highly similar performance indexes in Table 4.20. The Fused JGL shows many pairwise connections between nodes 14 to 24 that are not in the true graph of differences, which confirms the lower TPR and PPV in Table 4.20. Overall, the R-JGL is undoubtedly the best performing on the difference of tree graphs in this instance, showing larger TPR and PPV for different edges.

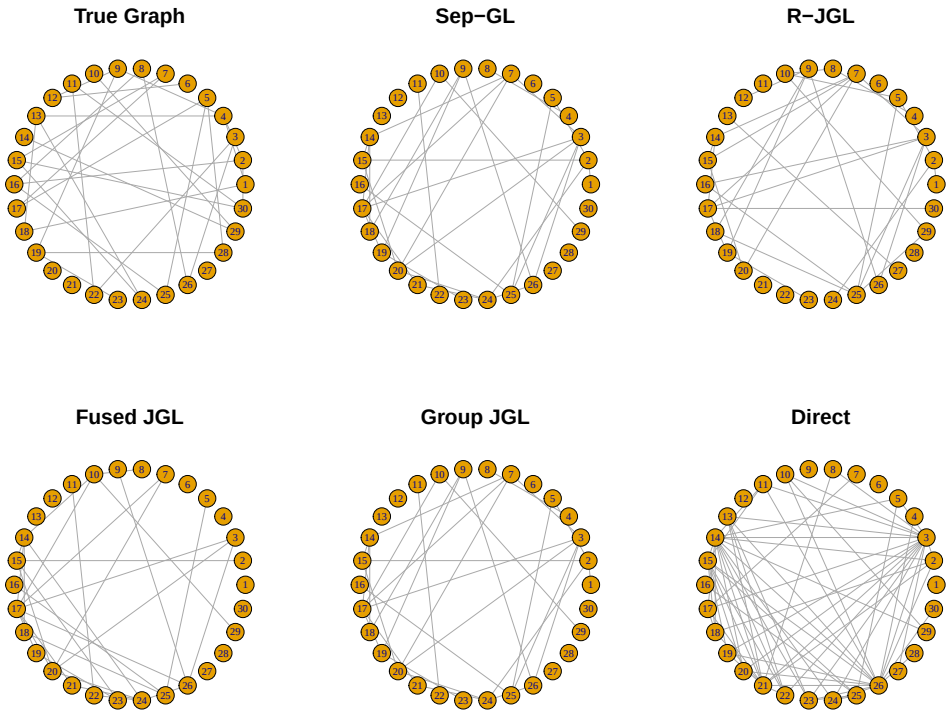


Figure 4.13: Comparison of graphs of differences for tree graphs, $p = 30$, $n = 200, \delta = 0.05$. Regularization parameters are imposed to reproduce true graph density.

Table 4.20: Performance on differences of tree graphs with fixed densities, p=30, n=200, δ=0.05

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.258	0.290	0.233	0.267	0.036
DIFFERENCE: TPR	0.267	0.300	0.233	0.267	0.033
EGD	0.071	0.071	0.069	0.069	0.064
PGD	0.069	0.069	0.069	0.069	0.069

To conclude the assessment on the graphs of differences with fixed densities, we present the opposite simulation scenario, with $p = 100,\ n = 60,\ \delta = 0.20$. When the graphs are more different, the performance of the learning methods improves greatly. We remark that the improvement in performance happens in spite of the less-favourable relationship between the dimension and the sample size and regardless of the graph structure. From this observation, we conclude that the ability in learning the graphs, when the regularization parameters are set to reproduce the PGD, depends primarily on the densities of the graphs of differences.

The Table 4.21 shows the results on the differences of scale-free graphs: the Direct method still shows lower precision and recall; the R-JGL and Group JGL appear to be the best methods in identifying the different edges, which is confirmed in all the simulation scenarios reported in Appendix A.3.2.

Table 4.21: Performance on differences of scale-free graphs with fixed densities, p=100, n=60, δ=0.20

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.445	0.451	0.429	0.451	0.400
DIFFERENCE: TPR	0.447	0.450	0.427	0.452	0.401
EGD	0.405	0.403	0.402	0.405	0.404
PGD	0.404	0.404	0.404	0.404	0.404

It is interesting to observe that, with the estimated densities out of consideration, the Fused JGL shows worse performance than the Group one in identifying the different edges. Theoretically, the Fused penalty imposes a constraint on the similarity of the entries of the concentration matrices, which would lead to a closer approximation of the matrix of differences. In spite of this, we see that it is the Group penalty that is better able to identify the differences that are non-zero between the two groups.

4.9 Final Remarks

After investigating the results in Section 4.8, we draw some final remarks related to our comparative analysis.

Overall performance.

The first observation we raise is that there is a commonly low ability to correctly identify the population graphs among all the considered learning methods. In the majority of cases, none of them is even able to select half of the true conditional dependencies between couples of random variables. As the density of the population graphs decreases, the performance decays as well, both because the model selection tends to lead the method towards models that do not match the true density and because even with the right regularization parameter, the methods do not identify correctly most of the edges.

Performance assessment.

Evaluating the performance of these Multiple Graph Structure Learning methods can be done by comparing the precision and recall indexes. Nevertheless, we remark that this comparison is not straightforward when the methods also vary in terms of average density of the learnt graphs, since these measures highly depend on the density itself. If a graph is estimated as denser than the true one, it will tend to have larger TPR. Viceversa, if a graph is learnt as sparser, it will tend to show larger PPV. The graphs should therefore be compared either by also portraying the information on the density (EGD and PGD) or by allowing the methods to reproduce the true density, so that the precision and recall can be compared directly without any direct responsibility of the density. This gives further information on how differently the graphs are estimated in terms of *which* edges are identified as present rather than *how many*.

Graph structures.

From our set of simulations, there does not appear to be a significant and systematic difference in the ability to learn the graphs with respect to the graph structures.

When comparing graphs that experienced a low changing proportion of edges between the two conditions ($\delta = 0.05$), two different graph structures such as the Markov-chain graph and the scale-free graph do not show any preference among learning methods. Even so, it would be difficult to assess whether the ability or inability to correctly estimate the graphs is due to the graph structure or to the density in the first place.

We remark the shared limited ability to correctly estimate the tree graph, which is mainly due to its lower density with respect to the other graph structures. In this regard, we observe that if the right regularization parameter is chosen, the R-JGL shows slightly better performance than the other methods.

Method comparison.

Across this set of simulations, it appeared that the R-JGL is preferable in order to reproduce a density that is not excessively far from the true one, even though it tends to identify denser graphs than needed when the dimension of the data is larger than the sample size. When p increases, this method, as well as the Group JGL, is the closest to estimating the true graph densities.

Furthermore, when the regularization parameter is set so that the resulting graphs have the same densities as the true ones, the R-JGL shows evidence of being the most suitable method in identifying the graph of differences in edge values.

In terms of performance indexes, this method shows largest precision and recall systematically when computed on the graphs as if they were separate.

It should be noted that the R-JGL can lead to estimates showing more variability in performance, especially when $p > n$.

The Sep-GL has the main advantage of low computational effort compared to the other methods. It is the only method that is able to learn the graphs in few seconds even with $p = 100$. Secondly, it is the method that is able to best identify the graph density when $p > n$, especially in the scenario $p = 30, n = 20$. On the other hand, it is not recommended when learning graphs with higher dimension, such as the $p = 100$ scenarios.

We remark that when the regularization parameters are imposed to be reproducing the true graph densities, the Sep-GL is similar to the other methods, if not preferable. On many scenarios, it shows the best precision in identifying matching present edges and highest recall for mismatching edges.

As for the Fused JGL, we notice that this is the method that is the least able to learn graphs without any knowledge of their densities. In particular, it tends to identify graphs that are sparser than the true ones, especially when $p > n$, which also results is low recall. On the other hand, when the regularization parameters are fixed, this method shows the best precision in identifying mismatching edges and the best sensitivity for matching present ones.

The Group JGL shares this inability to correctly estimate the regularization parameters when the dimension of the data is larger than its sample size, by identifying graphs that are sparser than the population ones. The Group

JGL, as opposed to the Fused, is the best at reproducing the true densities when $n >> p$, especially for the graphs of differences. Furthermore, it does not suffer from the low TPR on mismatching edges that can be frequently seen for the Fused JGL.

Both these methods rely on a joint evaluation of the estimates and thus require a larger application time than the Sep-GL, but lower than the R-JGL.

When it comes to estimating the graphs of differences, we do not consider the Direct method to be competitive with respect to the Sep-GL and the JGL methods evaluated on the differences of their estimates. This conclusion comes from the fact that in most cases, the Direct Estimation, as proposed by [Zhao et al., 2014], is not able to learn graphs of differences that are close in density to the true ones. Furthermore, we remark that when the portion δ of changed edges is large, this method is less able to learn the true graphs of differences. Even when the regularization parameter is set so as to be compared with the other methods with equal density, this method shows systematically worse performance than its competitors.

In addition to this, it is the method that requires more computing time when the dimension increases.

Computational load.

Even though the primary focus of this comparative analysis is the ability to learn the graphs correctly and not the amount of time required to do so, it is worth considering briefly a comparison of the computational load involved in each of their applications. Table 4.22 shows the computation times (in seconds) needed by each method in order to complete an AIC selection procedure, as described in Section 4.6.1 (Algorithm 3).

Table 4.22: Comparison of computing times (in seconds) for model selection with AIC using five different methods, under five simulation scenarios. Operated on i7-7700K CPU

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
$p = 30,\ n = 200$	0.45	94.33	123.33	107.50	4.47
$p = 30,\ n = 40$	0.48	117.56	175.47	144.61	14.54
$p = 30,\ n = 20$	9.42	674.50	210.56	194.09	35.44
$p = 100,\ n = 400$	19.40	1676.35	812.57	844.00	4655.89
$p = 100,\ n = 60$	6.61	2215.77	404.34	263.77	10133.18

Among these methods, the Sep-GL is the least demanding in terms of computational time required to fit the models. This aspect should be taken into consideration if the analysis were to be expanded to larger dimensions. The JGL methods all required much longer than the Sep-GL. The R-JGL

was more demanding than the other JGL methods, especially with a larger dimension ($p = 100$).

The Direct method is considerably limited when it comes to its implementation. It is extremely slow with larger dimension, e.g. almost three hours for $p = 100,\ n = 60$. Furthermore, this method could not be applied in an average Personal Computer - with 8GB of RAM - for any number of variables greater than 150 due to insufficient memory. While this limitation could be overcome with a more powerful machine, we do consider its limited scalability to problems with much larger dimension.

Separate Graph Structure Learning.

One of the questions that motivated this set of simulations concerned the possible improvements in estimating graphs jointly rather than separately, by exploiting the common structure of the graphs. We analysed two different degrees of similarity: a portion $\delta = 0.05$ of edges changing between the two conditions and a portion $\delta = 0.20$. Overall, we can not conclude that the Joint Graph Structure Learning methods are substantially better than the Separate Graphical Lasso in learning the graphs. In most cases, the Sep-GL is not inferior to the R-JGL or the Fused or Group JGL. It was interesting to discover that when the regularization parameters were set in order to reproduce the true graph densities, the Sep-GL showed higher precision in identifying the matching present edges, while the R-JGL showed higher precision and recall on the graphs when evaluated separately.

Model selection.

From this set of simulations, it is evident that the main limitation of these methods is the unpredictability of the density of the learnt graphs. While we do conclude that the R-JGL appears to oscillate more closely to the true graph densities than the other methods, it does not provide with any certainty of high similarity to the true graphs.

It was our goal to give the best possibility in finding the correct regularization parameters to all the methods, by implementing an automatic AIC selection algorithm that progressively gets closer to the minimum AIC in the most accurate way, while not compromising its possible application to data with larger dimensions. Among the common model selection criteria, such as the cross validation and the BIC, the AIC resulted as preferable as a good trade-off between the proximity to the true regularization parameters and the rapidity in application.

Even so, in most cases, the multiple graph structure learning methods were not able to learn graphs with densities -and thus edge patterns- similar to the population ones.

Chapter 5

Conclusions

The aim of this thesis was to give an understanding of different graph structure learning strategies applied to research problems that include a distinction between multiple experimental conditions for Gaussian-distributed data. In particular, the research interest lies in the possibility to borrow strength from the similarity of the graph structures by operating a joint estimation or a differential one.

In order to perform our comparison, we built a simulation study analysis that we designed in order to control a variety of scenarios, giving an overview of the strengths and weaknesses of each learning method. The scenarios diversify in terms of dimensionality, relationship between number of variables and sample sizes, level of differentiation between experimental conditions and underlying graph structure.

We compared graph structure learning methods that are based on estimating the edge patterns for each condition separately, jointly or differentially by using an ℓ_1-norm penalty on the maximum likelihood estimate for the concentration matrices. In order to compare the proximity of each learnt graph to its population equivalent, we calculated the precision and recall indexes. Therefore, we were able to understand whether a strategy was able to effectively reconstruct the population graphs or not.

The most notable differences arising from the results of our simulation analysis consist in the reduced ability of the joint learning methods proposed by [Danaher et al., 2014] to learn graphs with no knowledge of their density when the dimension of the problem exceeds the sample size. In addition to this, we found the Direct Estimation method proposed by [Zhao et al., 2014] to be the least suitable to learning graphs of differences in most scenarios, in terms of retrieving the true structures, having higher variability and requiring greater computational load.

Overall, the Reparametrization Joint Graphical Lasso ([Guo et al., 2011])

showed slightly better performance in many scenarios, especially in terms of estimating regularization parameters producing graphs with density closer to the true one.

Moreover, a surprising result was that the Separate Graphical Lasso was highly competitive with respect to the other learning methods, even with a large level of similarity between the graphs. It can be then considered as a valid alternative for MGSL problems, also due to its lower computational complexity.

As far as the graph structures were concerned, there did not appear to be a systematic and significant difference in performance with respect to different graph types. The only observation we reported was a subtle advantage of the Reparametrization JGL in learning tree-structured graphs.

As the main common feature of the methods under analysis, we observed a shared low performance in terms of precision and recall. This phenomenon was emphasized for population graphs with few edges, as the methods estimated regularization parameters that produced graphs with density not comparable to the true one.

The emphasis that we convey on the graph densities is motivated by the fact that the ability or inability of a method to learn a graph can be decomposed in two tasks. The first is the ability to choose the correct regularization parameters. The second is the effectiveness in identifying the correct edges with respect to the population graphs. The second task relies heavily on the first one, since a graph that is too dense may contaminate the results with many false positives and viceversa a sparser graph may not identify the true positives.

We believe that the main limitation of these graph structure learning procedures is the identification of the regularization parameters, which we chose to select through AIC, as commonly suggested in the literature. We believe that relying on the evaluation of the degrees of freedom of the model, estimated as number of edges in the learnt graphs, often does not lead to choosing the right parameter. However, we did not observe any improvement with cross validation, which on the other hand is more computationally-demanding.

Given all our considerations, we suggest that an alternative strategy could be to set a desired level of graph density. For instance, for studies that involve the comparison of two experimental conditions, it can be interesting to perform model selection for graphs that identify the differences between the elements of the concentration matrices, imposing as much dissimilarity

as can be assumed in a specific application domain. In this case, we suggest learning the graphs of differences with the Reparametrization JGL method.

For studies involving more conditions or for which each graph needs to be recovered, the choice of learning method can be operated with respect to the objective of the analysis. If the focus is on the mismatching edges, the Separate Graphical Lasso tends to identify more of them. If the researcher wants to select the matching present ones, the Fused JGL is preferred. Finally, if there is no regard for the commonality of the structures, we believe that the Reparametrization JGL is the most suitable one.

Bibliography

[Albert and Barabási, 2002] Albert, R. and Barabási, A.-L. (2002). Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47.

[Anders and Huber, 2010] Anders, S. and Huber, W. (2010). Differential expression analysis for sequence count data. *Genome biology*, 11(10):R106.

[Babu, 2004] Babu, M. M. (2004). Introduction to microarray data analysis. *Computational genomics: Theory and application*, 225:249.

[Babu et al., 2004] Babu, M. M., Luscombe, N. M., Aravind, L., Gerstein, M., and Teichmann, S. A. (2004). Structure and evolution of transcriptional regulatory networks. *Current opinion in structural biology*, 14(3):283–291.

[Banerjee et al., 2008] Banerjee, O., Ghaoui, L. E., and d'Aspremont, A. (2008). Model selection through sparse maximum likelihood estimation for multivariate gaussian or binary data. *Journal of Machine learning research*, 9(Mar):485–516.

[Barabási and Albert, 1999] Barabási, A.-L. and Albert, R. (1999). Emergence of scaling in random networks. *science*, 286(5439):509–512.

[Bogdan et al., 2015] Bogdan, M., Van Den Berg, E., Sabatti, C., Su, W., and Candès, E. J. (2015). Slope—adaptive variable selection via convex optimization. *The annals of applied statistics*, 9(3):1103.

[Chen and Sharp, 2004] Chen, H. and Sharp, B. M. (2004). Content-rich biological network constructed by mining pubmed abstracts. *BMC bioinformatics*, 5(1):147.

[Choi et al., 2005] Choi, J. K., Yu, U., Yoo, O. J., and Kim, S. (2005). Differential coexpression analysis using microarray data and its application to human cancer. *Bioinformatics*, 21(24):4348–4355.

[Cox and Wermuth, 1996] Cox, D. and Wermuth, N. (1996). *Multivariate Dependencies: Models, Analysis and Interpretation*, volume 67. CRC Press.

[Cui and Churchill, 2003] Cui, X. and Churchill, G. A. (2003). Statistical tests for differential expression in cdna microarray experiments. *Genome biology*, 4(4):210.

[Danaher et al., 2014] Danaher, P., Wang, P., and Witten, D. M. (2014). The joint graphical lasso for inverse covariance estimation across multiple classes. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 76(2):373–397.

[Dempster, 1972] Dempster, A. P. (1972). Covariance selection. *Biometrics*, pages 157–175.

[Erdős and Rényi, 1959] Erdős, P. and Rényi, A. (1959). On random graphs i. *Publ. Math. Debrecen*, 6:290–297.

[Fan et al., 2009] Fan, J., Feng, Y., and Wu, Y. (2009). Network exploration via the adaptive lasso and scad penalties. *The annals of applied statistics*, 3(2):521.

[Friedman et al., 2007] Friedman, J., Hastie, T., Höfling, H., Tibshirani, R., et al. (2007). Pathwise coordinate optimization. *The Annals of Applied Statistics*, 1(2):302–332.

[Friedman et al., 2008] Friedman, J., Hastie, T., and Tibshirani, R. (2008). Sparse inverse covariance estimation with the graphical lasso. *Biostatistics*, 9(3):432–441.

[Guo et al., 2011] Guo, J., Levina, E., Michailidis, G., and Zhu, J. (2011). Joint estimation of multiple graphical models. *Biometrika*, 98(1):1–15.

[Hannart and Naveau, 2014] Hannart, A. and Naveau, P. (2014). Estimating high dimensional covariance matrices: A new look at the gaussian conjugate framework. *Journal of Multivariate Analysis*, 131:149–162.

[Hu et al., 2009] Hu, R., Qiu, X., Glazko, G., Klebanov, L., and Yakovlev, A. (2009). Detecting intergene correlation changes in microarray analysis: a new approach to gene selection. *BMC bioinformatics*, 10(1):20.

[Jeanmougin et al., 2010] Jeanmougin, M., De Reynies, A., Marisa, L., Paccard, C., Nuel, G., and Guedj, M. (2010). Should we abandon the t-test in

the analysis of gene expression microarray data: a comparison of variance modeling strategies. *PloS one*, 5(9):e12336.

[Jeong et al., 2000] Jeong, H., Tombor, B., Albert, R., Oltvai, Z. N., and Barabási, A.-L. (2000). The large-scale organization of metabolic networks. *Nature*, 407(6804):651.

[Joe, 2006] Joe, H. (2006). Generating random correlation matrices based on partial correlations. *Journal of Multivariate Analysis*, 97(10):2177–2189.

[Kiiveri, 2011] Kiiveri, H. T. (2011). Multivariate analysis of microarray data: differential expression and differential connection. *BMC bioinformatics*, 12(1):42.

[Krapivsky et al., 2000] Krapivsky, P. L., Redner, S., and Leyvraz, F. (2000). Connectivity of growing random networks. *Physical review letters*, 85(21):4629.

[Lauritzen, 1996] Lauritzen, S. L. (1996). *Graphical models*, volume 17. Clarendon Press.

[Liu et al., 2014] Liu, S., Quinn, J. A., Gutmann, M. U., Suzuki, T., and Sugiyama, M. (2014). Direct learning of sparse changes in markov networks by density ratio estimation. *Neural computation*, 26(6):1169–1197.

[Mardia et al., 1979] Mardia, K., Kent, J., and Bibby, J. (1979). Multivariate analysis. *Probability and mathematical statistics. Academic Press Inc.*

[Massa et al., 2010] Massa, M. S., Chiogna, M., and Romualdi, C. (2010). Gene set analysis exploiting the topology of a pathway. *BMC systems biology*, 4(1):121.

[Meinshausen and Bühlmann, 2006] Meinshausen, N. and Bühlmann, P. (2006). High-dimensional graphs and variable selection with the lasso. *The annals of statistics*, pages 1436–1462.

[R Core Team, 2018] R Core Team (2018). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

[Reverter et al., 2006] Reverter, A., Ingham, A., Lehnert, S. A., Tan, S.-H., Wang, Y., Ratnakumar, A., and Dalrymple, B. P. (2006). Simultaneous identification of differential gene expression and connectivity in inflammation, adipogenesis and cancer. *Bioinformatics*, 22(19):2396–2404.

[Schäfer and Strimmer, 2004] Schäfer, J. and Strimmer, K. (2004). An empirical bayes approach to inferring large-scale gene association networks. *Bioinformatics*, 21(6):754–764.

[Speed and Kiiveri, 1986] Speed, T. P. and Kiiveri, H. T. (1986). Gaussian markov distributions over finite graphs. *The Annals of Statistics*, pages 138–150.

[Tibshirani, 1996] Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288.

[Tibshirani et al., 2005] Tibshirani, R., Saunders, M., Rosset, S., Zhu, J., and Knight, K. (2005). Sparsity and smoothness via the fused lasso. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(1):91–108.

[Uhler, 2017] Uhler, C. (2017). Gaussian graphical models: An algebraic and geometric perspective. *arXiv preprint arXiv:1707.04345*.

[Van Noort et al., 2004] Van Noort, V., Snel, B., and Huynen, M. A. (2004). The yeast coexpression network has a small-world, scale-free architecture and can be explained by a simple model. *EMBO reports*, 5(3):280–284.

[Yuan and Lin, 2006] Yuan, M. and Lin, Y. (2006). Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(1):49–67.

[Zhang et al., 2009] Zhang, B., Li, H., Riggins, R. B., Zhan, M., Xuan, J., Zhang, Z., Hoffman, E. P., Clarke, R., and Wang, Y. (2009). Differential dependency network analysis to identify condition-specific topological changes in biological networks. *Bioinformatics*, 25(4):526–532.

[Zhao et al., 2014] Zhao, S. D., Cai, T. T., and Li, H. (2014). Direct estimation of differential networks. *Biometrika*, 101(2):253–268.

Appendix A

Tables

A.1 Model selection: CV, AIC, BIC

Table A.1: Comparison of model selection methods for Separate Graphical Lasso, $p = 30, n = 20, \delta = 0.2$. CV is 5-fold.

(a) Computing times

	CV	AIC	BIC
TIME(s)	1.58	0.42	0.42

(b) Density of Random Graph ($\pi = 0.05$)

	h=1	h=2
TRUE	0.22	0.23
CV	0.21	0.17
AIC	0.16	0.12
BIC	0.16	0.12

(c) Density of Markov Chain Graph

	h=1	h=2
TRUE	0.25	0.23
CV	0.22	0.28
AIC	0.14	0.16
BIC	0.14	0.16

(d) Density of Tree Graph

	h=1	h=2
TRUE	0.07	0.06
CV	0.17	0.21
AIC	0.13	0.15
BIC	0.13	0.15

(e) Density of Scale-free Graph

	h=1	h=2
TRUE	0.25	0.23
CV	0.19	0.24
AIC	0.14	0.23
BIC	0.14	0.20

Table A.2: Comparison of model selection methods for Reparametrization Joint Graphical Lasso, $p = 100, n = 400, \delta = 0.05$. CV is 10-fold.

(a) Computing times

	CV	AIC	BIC
TIME(s)	10252.48	1867.55	1880.36

(b) Density of Random Graph ($\pi = 0.05$)

	h=1	h=2
TRUE	0.09	0.09
CV	0.00	0.00
AIC	0.10	0.09
BIC	0.00	0.00

(c) Density of Markov Chain Graph

	h=1	h=2
TRUE	0.07	0.07
CV	0.00	0.00
AIC	0.10	0.09
BIC	0.00	0.00

(d) Density of Tree Graph

	h=1	h=2
TRUE	0.02	0.02
CV	0.00	0.00
AIC	0.09	0.08
BIC	0.00	0.00

(e) Density of Scale-free Graph

	h=1	h=2
TRUE	0.12	0.12
CV	0.00	0.00
AIC	0.09	0.10
BIC	0.00	0.00

Table A.3: Parameters for AIC selection algorithm on each simulation setting, for each method. s indicates length of range of values to choose from, c is the multiplying/dividing factor to update the range, L is the left endpoint of the range, R is the right endpoint, `it.max` is maximum number of iterations for updating the range, ν is the initial penalization parameter for the R-JGL by [Guo et al., 2011], `nlambda` is the number of penalization values in the AIC implemented in the `dpm` function

p	n	method	s	c	L	R	`it.max`	ν	`nlambda`
30	200								
		Sep-GL	15	2	0.05	2	15		
		R-JGL	5	2	9×10^{-5}	5×10^{-4}	7	0.01	
		Fused and Group JGL	10	5	0.05	2	5		
		Direct							10
30	40								
		Sep-GL	15	2	0.5	2	15		
		R-JGL	5	2	9×10^{-5}	5×10^{-4}	7	0.01	
		Fused and Group JGL	10	2	0.5	1.2	5		
		Direct							15
30	20								
		Sep-GL	10	2	0.7	2	15		
		R-JGL	5	2	5×10^{-4}	5×10^{-3}	5	0.05	
		Fused and Group JGL	10	2	0.2	0.7	5		
		Direct							15
100	400								
		Sep. G-lasso	10	2	0.05	2	15		
		R-JGL	5	2	9×10^{-5}	5×10^{-4}	5	0.1	
		Fused and Group JGL	10	2	0.5	1.5	5		
		Direct							10
100	60								
		Sep-GL	10	2	0.7	2	15		
		R-JGL	5	2	1×10^{-4}	1×10^{-3}	5	0.5	
		Fused and Group JGL	10	2	0.5	2	5		
		Direct							10

A.2 Performance results: AIC selection

A.2.1 Separate and Joint GSL

Table A.4: Performance on random graphs ($\pi = 0.1$), p=30, n=200, $\delta = 0.05$

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.234 (0.030)	0.377 (0.072)	0.465 (0.100)	0.409 (0.151)
TPR	0.703 (0.074)	0.589 (0.073)	0.493 (0.113)	0.517 (0.135)
MATCHING PRESENCE: PPV	0.256 (0.069)	0.369 (0.092)	0.376 (0.095)	0.373 (0.161)
MATCHING PRESENCE: TPR	0.563 (0.120)	0.545 (0.105)	0.542 (0.122)	0.496 (0.143)
MISMATCHING: PPV	0.114 (0.018)	0.183 (0.048)	0.440 (0.184)	0.254 (0.185)
MISMATCHING: TPR	0.503 (0.071)	0.309 (0.059)	0.137 (0.057)	0.254 (0.101)
EAGD	0.418 (0.079)	0.222 (0.053)	0.160 (0.074)	0.202 (0.094)
PAGD	0.137 (0.014)	0.137 (0.014)	0.137 (0.014)	0.137 (0.014)

Table A.5: Performance on Markov-chain graphs, p=30, n=200, $\delta = 0.05$

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.214 (0.019)	0.338 (0.057)	0.455 (0.096)	0.351 (0.050)
TPR	0.675 (0.081)	0.580 (0.083)	0.426 (0.126)	0.494 (0.094)
MATCHING PRESENCE: PPV	0.230 (0.048)	0.303 (0.085)	0.338 (0.099)	0.300 (0.075)
MATCHING PRESENCE: TPR	0.539 (0.124)	0.536 (0.114)	0.494 (0.150)	0.469 (0.118)
MISMATCHING: PPV	0.129 (0.021)	0.216 (0.048)	0.504 (0.226)	0.253 (0.055)
MISMATCHING: TPR	0.529 (0.065)	0.354 (0.066)	0.130 (0.070)	0.306 (0.091)
EAGD	0.348 (0.060)	0.196 (0.056)	0.111 (0.055)	0.160 (0.054)
PAGD	0.109 (0.003)	0.109 (0.003)	0.109 (0.003)	0.109 (0.003)

Table A.6: Performance on Markov-chain graphs, p=30, n=200, $\delta = 0.20$

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.336 (0.015)	0.460 (0.047)	0.436 (0.036)	0.410 (0.033)
TPR	0.774 (0.040)	0.620 (0.042)	0.576 (0.073)	0.635 (0.062)
MATCHING PRESENCE: PPV	0.159 (0.032)	0.216 (0.045)	0.201 (0.042)	0.175 (0.044)
MATCHING PRESENCE: TPR	0.635 (0.083)	0.583 (0.074)	0.642 (0.077)	0.574 (0.093)
MISMATCHING: PPV	0.342 (0.027)	0.452 (0.061)	0.584 (0.095)	0.458 (0.057)
MISMATCHING: TPR	0.487 (0.057)	0.297 (0.035)	0.204 (0.044)	0.298 (0.035)
EAGD	0.553 (0.043)	0.328 (0.046)	0.320 (0.058)	0.376 (0.061)
PAGD	0.240 (0.006)	0.240 (0.006)	0.240 (0.006)	0.240 (0.006)

Table A.7: Performance on tree graphs, p=30, n=200, $\delta = 0.20$

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.159 (0.024)	0.233 (0.044)	0.401 (0.119)	0.355 (0.138)
TPR	0.626 (0.079)	0.556 (0.088)	0.264 (0.123)	0.391 (0.109)
MATCHING PRESENCE: PPV	0.067 (0.059)	0.078 (0.055)	0.130 (0.117)	0.177 (0.234)
MATCHING PRESENCE: TPR	0.498 (0.250)	0.521 (0.201)	0.386 (0.205)	0.413 (0.218)
MISMATCHING: PPV	0.173 (0.030)	0.249 (0.061)	0.682 (0.271)	0.431 (0.143)
MISMATCHING: TPR	0.555 (0.080)	0.357 (0.067)	0.089 (0.076)	0.275 (0.091)
EAGD	0.268 (0.055)	0.167 (0.050)	0.051 (0.033)	0.087 (0.042)
PAGD	0.067 (0.000)	0.067 (0.000)	0.067 (0.000)	0.067 (0.000)

Table A.8: Performance on scale-free graphs, p=30, n=200, $\delta = 0.05$

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.221 (0.039)	0.328 (0.069)	0.448 (0.107)	0.376 (0.066)
TPR	0.692 (0.062)	0.614 (0.061)	0.474 (0.112)	0.515 (0.099)
MATCHING PRESENCE: PPV	0.251 (0.081)	0.311 (0.097)	0.354 (0.113)	0.334 (0.080)
MATCHING PRESENCE: TPR	0.578 (0.086)	0.588 (0.075)	0.561 (0.125)	0.516 (0.104)
MISMATCHING: PPV	0.133 (0.030)	0.191 (0.052)	0.464 (0.162)	0.266 (0.076)
MISMATCHING: TPR	0.542 (0.058)	0.337 (0.050)	0.146 (0.057)	0.295 (0.091)
EAGD	0.355 (0.071)	0.218 (0.060)	0.127 (0.050)	0.160 (0.061)
PAGD	0.110 (0.004)	0.110 (0.004)	0.110 (0.004)	0.110 (0.004)

Table A.9: Performance on scale-free graphs, p=30, n=200, $\delta = 0.20$

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.331 (0.020)	0.457 (0.042)	0.447 (0.031)	0.396 (0.039)
TPR	0.79 (0.050)	0.629 (0.055)	0.535 (0.066)	0.644 (0.049)
MATCHING PRESENCE: PPV	0.154 (0.027)	0.221 (0.053)	0.208 (0.029)	0.175 (0.036)
MATCHING PRESENCE: TPR	0.656 (0.084)	0.601 (0.066)	0.636 (0.105)	0.616 (0.079)
MISMATCHING: PPV	0.350 (0.021)	0.466 (0.058)	0.592 (0.077)	0.458 (0.062)
MISMATCHING: TPR	0.505 (0.063)	0.325 (0.049)	0.177 (0.038)	0.315 (0.050)
EAGD	0.569 (0.060)	0.330 (0.051)	0.287 (0.053)	0.390 (0.053)
PAGD	0.237 (0.009)	0.237 (0.009)	0.237 (0.009)	0.237 (0.009)

Table A.10: Performance on random graphs ($\pi = 0.1$), p=30, n=40, δ=0.05

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.245 (0.040)	0.281 (0.051)	0.482 (0.244)	0.358 (0.290)
TPR	0.362 (0.088)	0.349 (0.065)	0.105 (0.086)	0.094 (0.096)
MATCHING PRESENCE: PPV	0.315 (0.118)	0.272 (0.073)	0.403 (0.233)	0.382 (0.348)
MATCHING PRESENCE: TPR	0.181 (0.073)	0.282 (0.064)	0.126 (0.106)	0.078 (0.089)
MISMATCHING: PPV	0.118 (0.027)	0.128 (0.034)	0.298 (0.318)	0.188 (0.160)
MISMATCHING: TPR	0.377 (0.106)	0.207 (0.071)	0.025 (0.026)	0.062 (0.064)
EAGD	0.220 (0.071)	0.184 (0.049)	0.032 (0.033)	0.036 (0.046)
PAGD	0.143 (0.012)	0.143 (0.012)	0.143 (0.012)	0.143 (0.012)

Table A.11: Performance on random graphs ($\pi = 0.1$), p=30, n=40, δ=0.20

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.349 (0.030)	0.395 (0.046)	0.539 (0.244)	0.541 (0.142)
TPR	0.415 (0.122)	0.358 (0.067)	0.114 (0.128)	0.144 (0.104)
MATCHING PRESENCE: PPV	0.228 (0.078)	0.199 (0.042)	0.342 (0.246)	0.337 (0.184)
MATCHING PRESENCE: TPR	0.226 (0.116)	0.289 (0.091)	0.152 (0.142)	0.129 (0.091)
MISMATCHING: PPV	0.353 (0.038)	0.396 (0.069)	0.382 (0.392)	0.412 (0.175)
MISMATCHING: TPR	0.436 (0.077)	0.216 (0.038)	0.018 (0.025)	0.085 (0.064)
EAGD	0.319 (0.117)	0.243 (0.065)	0.066 (0.089)	0.082 (0.073)
PAGD	0.262 (0.009)	0.262 (0.009)	0.262 (0.009)	0.262 (0.009)

Table A.12: Performance on Markov-chain graphs, p=30, n=40, δ=0.05

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.221 (0.051)	0.221 (0.041)	0.576 (0.247)	0.424 (0.228)
TPR	0.309 (0.087)	0.337 (0.076)	0.092 (0.078)	0.101 (0.088)
MATCHING PRESENCE: PPV	0.281 (0.161)	0.188 (0.062)	0.500 (0.287)	0.347 (0.330)
MATCHING PRESENCE: TPR	0.168 (0.097)	0.273 (0.080)	0.126 (0.097)	0.088 (0.103)
MISMATCHING: PPV	0.133 (0.038)	0.137 (0.030)	0.144 (0.304)	0.257 (0.216)
MISMATCHING: TPR	0.322 (0.106)	0.220 (0.070)	0.009 (0.017)	0.065 (0.050)
EAGD	0.168 (0.070)	0.174 (0.047)	0.027 (0.033)	0.035 (0.044)
PAGD	0.111 (0.003)	0.111 (0.003)	0.111 (0.003)	0.111 (0.003)

Table A.13: Performance on Markov-chain graphs, p=30, n=40, δ=0.20

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.334 (0.037)	0.378 (0.031)	0.476 (0.239)	0.492 (0.144)
TPR	0.448 (0.155)	0.363 (0.075)	0.082 (0.074)	0.109 (0.057)
MATCHING PRESENCE: PPV	0.181 (0.070)	0.169 (0.037)	0.335 (0.238)	0.236 (0.126)
MATCHING PRESENCE: TPR	0.270 (0.179)	0.314 (0.115)	0.135 (0.118)	0.096 (0.071)
MISMATCHING: PPV	0.350 (0.042)	0.385 (0.059)	0.310 (0.387)	0.552 (0.156)
MISMATCHING: TPR	0.430 (0.075)	0.205 (0.035)	0.009 (0.012)	0.071 (0.039)
EAGD	0.331 (0.142)	0.231 (0.057)	0.043 (0.045)	0.053 (0.033)
PAGD	0.238 (0.008)	0.238 (0.008)	0.238 (0.008)	0.238 (0.008)

Table A.14: Performance on tree graphs, p=30, n=40, δ=0.05

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.144 (0.034)	0.163 (0.034)	0.391 (0.374)	0.323 (0.334)
TPR	0.268 (0.076)	0.308 (0.105)	0.056 (0.072)	0.055 (0.049)
MATCHING PRESENCE: PPV	0.190 (0.286)	0.106 (0.062)	0.254 (0.398)	0.190 (0.348)
MATCHING PRESENCE: TPR	0.096 (0.103)	0.229 (0.152)	0.075 (0.106)	0.045 (0.060)
MISMATCHING: PPV	0.108 (0.018)	0.128 (0.028)	0.131 (0.310)	0.181 (0.293)
MISMATCHING: TPR	0.320 (0.117)	0.219 (0.088)	0.006 (0.013)	0.033 (0.048)
EAGD	0.131 (0.049)	0.133 (0.056)	0.016 (0.028)	0.014 (0.016)
PAGD	0.066 (0.000)	0.066 (0.000)	0.066 (0.000)	0.066 (0.000)

Table A.15: Performance on tree graphs, p=30, n=40, δ=0.20

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.145 (0.036)	0.163 (0.045)	0.160 (0.218)	0.365 (0.262)
TPR	0.269 (0.092)	0.286 (0.098)	0.024 (0.036)	0.043 (0.053)
MATCHING PRESENCE: PPV	0.085 (0.119)	0.041 (0.040)	0.114 (0.176)	0.113 (0.254)
MATCHING PRESENCE: TPR	0.151 (0.176)	0.239 (0.223)	0.099 (0.171)	0.068 (0.166)
MISMATCHING: PPV	0.162 (0.033)	0.182 (0.059)	0.075 (0.238)	0.324 (0.296)
MISMATCHING: TPR	0.295 (0.091)	0.204 (0.064)	0.002 (0.006)	0.033 (0.039)
EAGD	0.127 (0.040)	0.125 (0.052)	0.005 (0.011)	0.010 (0.013)
PAGD	0.067 (0.000)	0.067 (0.000)	0.067 (0.000)	0.067 (0.000)

Table A.16: Performance on scale-free graphs, p=30, n=40, δ=0.05

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.201 (0.033)	0.215 (0.034)	0.422 (0.286)	0.413 (0.205)
TPR	0.331 (0.114)	0.342 (0.093)	0.075 (0.084)	0.086 (0.067)
MATCHING PRESENCE: PPV	0.206 (0.097)	0.162 (0.048)	0.323 (0.283)	0.188 (0.179)
MATCHING PRESENCE: TPR	0.158 (0.122)	0.253 (0.114)	0.105 (0.126)	0.068 (0.077)
MISMATCHING: PPV	0.128 (0.029)	0.132 (0.042)	0.05 (0.130)	0.316 (0.280)
MISMATCHING: TPR	0.364 (0.112)	0.211 (0.064)	0.006 (0.018)	0.056 (0.042)
EAGD	0.190 (0.080)	0.184 (0.065)	0.023 (0.032)	0.026 (0.025)
PAGD	0.111 (0.003)	0.111 (0.003)	0.111 (0.003)	0.111 (0.003)

Table A.17: Performance on scale-free graphs, p=30, n=40, δ=0.20

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.343 (0.039)	0.383 (0.048)	0.558 (0.287)	0.546 (0.140)
TPR	0.379 (0.107)	0.335 (0.065)	0.073 (0.081)	0.114 (0.065)
MATCHING PRESENCE: PPV	0.208 (0.070)	0.180 (0.062)	0.386 (0.272)	0.316 (0.231)
MATCHING PRESENCE: TPR	0.191 (0.090)	0.274 (0.102)	0.124 (0.124)	0.104 (0.072)
MISMATCHING: PPV	0.350 (0.036)	0.391 (0.064)	0.365 (0.422)	0.513 (0.118)
MISMATCHING: TPR	0.408 (0.104)	0.204 (0.025)	0.011 (0.014)	0.076 (0.042)
EAGD	0.274 (0.096)	0.215 (0.056)	0.033 (0.044)	0.058 (0.042)
PAGD	0.241 (0.007)	0.241 (0.007)	0.241 (0.007)	0.241 (0.007)

Table A.18: Performance on random graphs ($\pi = 0.1$), p=30, n=20, δ=0.05

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.218 (0.031)	0.235 (0.056)	0.237 (0.280)	0.221 (0.189)
TPR	0.192 (0.077)	0.280 (0.165)	0.040 (0.073)	0.067 (0.080)
MATCHING PRESENCE: PPV	0.238 (0.128)	0.198 (0.075)	0.184 (0.265)	0.254 (0.320)
MATCHING PRESENCE: TPR	0.059 (0.035)	0.208 (0.173)	0.047 (0.086)	0.047 (0.068)
MISMATCHING: PPV	0.112 (0.028)	0.118 (0.048)	0.045 (0.136)	0.105 (0.115)
MISMATCHING: TPR	0.243 (0.128)	0.150 (0.057)	0.004 (0.011)	0.043 (0.053)
EAGD	0.128 (0.061)	0.202 (0.187)	0.019 (0.040)	0.034 (0.045)
PAGD	0.143 (0.012)	0.143 (0.012)	0.143 (0.012)	0.143 (0.012)

Table A.19: Performance on random graphs ($\pi = 0.1$), p=30, n=20, δ=0.20

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.341 (0.048)	0.307 (0.060)	0.259 (0.255)	0.354 (0.283)
TPR	0.275 (0.102)	0.499 (0.256)	0.061 (0.098)	0.051 (0.058)
MATCHING PRESENCE: PPV	0.169 (0.128)	0.130 (0.067)	0.135 (0.183)	0.166 (0.293)
MATCHING PRESENCE: TPR	0.068 (0.047)	0.421 (0.252)	0.071 (0.099)	0.027 (0.039)
MISMATCHING: PPV	0.351 (0.030)	0.347 (0.081)	0.055 (0.146)	0.286 (0.237)
MISMATCHING: TPR	0.392 (0.187)	0.149 (0.031)	0.002 (0.005)	0.037 (0.039)
EAGD	0.219 (0.105)	0.463 (0.276)	0.044 (0.081)	0.032 (0.037)
PAGD	0.260 (0.012)	0.260 (0.012)	0.260 (0.012)	0.260 (0.012)

Table A.20: Performance on Markov-chain graphs, p=30, n=20, δ=0.20

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.337 (0.040)	0.289 (0.041)	0.252 (0.200)	0.410 (0.307)
TPR	0.250 (0.111)	0.446 (0.262)	0.055 (0.079)	0.043 (0.086)
MATCHING PRESENCE: PPV	0.191 (0.227)	0.101 (0.042)	0.112 (0.145)	0.080 (0.226)
MATCHING PRESENCE: TPR	0.076 (0.064)	0.385 (0.291)	0.062 (0.091)	0.025 (0.066)
MISMATCHING: PPV	0.358 (0.043)	0.364 (0.063)	0.157 (0.316)	0.390 (0.284)
MISMATCHING: TPR	0.316 (0.136)	0.152 (0.030)	0.004 (0.009)	0.030 (0.049)
EAGD	0.188 (0.099)	0.405 (0.273)	0.038 (0.058)	0.028 (0.063)
PAGD	0.242 (0.007)	0.242 (0.007)	0.242 (0.007)	0.242 (0.007)

Table A.21: Performance on tree graphs, p=30, n=20, δ=0.05

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.105 (0.034)	0.123 (0.051)	0.177 (0.199)	0.131 (0.241)
TPR	0.174 (0.087)	0.282 (0.184)	0.064 (0.133)	0.019 (0.036)
MATCHING PRESENCE: PPV	0.086 (0.128)	0.082 (0.077)	0.115 (0.188)	0.065 (0.155)
MATCHING PRESENCE: TPR	0.064 (0.072)	0.222 (0.190)	0.075 (0.138)	0.017 (0.034)
MISMATCHING: PPV	0.094 (0.039)	0.091 (0.045)	0.112 (0.298)	0.134 (0.262)
MISMATCHING: TPR	0.233 (0.138)	0.138 (0.062)	0.011 (0.026)	0.027 (0.059)
EAGD	0.114 (0.053)	0.186 (0.191)	0.039 (0.117)	0.008 (0.021)
PAGD	0.066 (0.001)	0.066 (0.001)	0.066 (0.001)	0.066 (0.001)

Table A.22: Performance on tree graphs, p=30, n=20, δ=0.20

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.107 (0.027)	0.105 (0.036)	0.125 (0.250)	0.176 (0.355)
TPR	0.180 (0.076)	0.332 (0.251)	0.022 (0.051)	0.006 (0.013)
MATCHING PRESENCE: PPV	0.021 (0.044)	0.019 (0.023)	0.002 (0.009)	0.000 (0.000)
MATCHING PRESENCE: TPR	0.043 (0.092)	0.232 (0.304)	0.020 (0.087)	0.000 (0.000)
MISMATCHING: PPV	0.128 (0.035)	0.123 (0.055)	0.142 (0.313)	0.174 (0.354)
MISMATCHING: TPR	0.226 (0.073)	0.137 (0.053)	0.004 (0.009)	0.007 (0.012)
EAGD	0.109 (0.032)	0.260 (0.255)	0.013 (0.035)	0.002 (0.003)
PAGD	0.066 (0.000)	0.066 (0.000)	0.066 (0.000)	0.066 (0.000)

Table A.23: Performance on scale-free graphs, p=30, n=20, δ=0.20

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.328 (0.048)	0.287 (0.047)	0.411 (0.37)	0.366 (0.294)
TPR	0.251 (0.123)	0.532 (0.269)	0.022 (0.031)	0.021 (0.020)
MATCHING PRESENCE: PPV	0.213 (0.163)	0.126 (0.063)	0.319 (0.389)	0.204 (0.259)
MATCHING PRESENCE: TPR	0.099 (0.076)	0.479 (0.265)	0.039 (0.060)	0.02 (0.024)
MISMATCHING: PPV	0.355 (0.046)	0.319 (0.062)	0.078 (0.191)	0.294 (0.332)
MISMATCHING: TPR	0.317 (0.163)	0.142 (0.048)	0.007 (0.020)	0.012 (0.017)
EAGD	0.195 (0.108)	0.487 (0.280)	0.013 (0.019)	0.012 (0.014)
PAGD	0.241 (0.007)	0.241 (0.007)	0.241 (0.007)	0.241 (0.007)

Table A.24: Performance on random graphs (π = 0.1), p=100, n=400, δ = 0.05

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.281 (0.021)	0.358 (0.016)	0.492 (0.038)	0.383 (0.030)
TPR	0.553 (0.044)	0.526 (0.019)	0.372 (0.052)	0.432 (0.051)
MATCHING PRESENCE: PPV	0.344 (0.050)	0.345 (0.025)	0.412 (0.036)	0.349 (0.038)
MATCHING PRESENCE: TPR	0.393 (0.046)	0.457 (0.029)	0.427 (0.054)	0.395 (0.056)
MISMATCHING: PPV	0.140 (0.010)	0.176 (0.015)	0.448 (0.103)	0.225 (0.024)
MISMATCHING: TPR	0.500 (0.040)	0.310 (0.031)	0.090 (0.022)	0.249 (0.033)
EAGD	0.280 (0.037)	0.208 (0.008)	0.109 (0.023)	0.161 (0.031)
PAGD	0.141 (0.005)	0.141 (0.005)	0.141 (0.005)	0.141 (0.005)

Table A.25: Performance on random graphs ($\pi = 0.1$), p=100, n=400, $\delta = 0.20$

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.376 (0.014)	0.482 (0.047)	0.494 (0.022)	0.446 (0.020)
TPR	0.679 (0.029)	0.533 (0.057)	0.412 (0.045)	0.516 (0.045)
MATCHING PRESENCE: PPV	0.212 (0.018)	0.260 (0.037)	0.271 (0.019)	0.230 (0.018)
MATCHING PRESENCE: TPR	0.519 (0.035)	0.485 (0.061)	0.509 (0.043)	0.487 (0.048)
MISMATCHING: PPV	0.360 (0.016)	0.450 (0.050)	0.639 (0.039)	0.472 (0.025)
MISMATCHING: TPR	0.503 (0.018)	0.281 (0.010)	0.120 (0.020)	0.263 (0.021)
EAGD	0.468 (0.037)	0.292 (0.063)	0.217 (0.033)	0.301 (0.040)
PAGD	0.258 (0.003)	0.258 (0.003)	0.258 (0.003)	0.258 (0.003)

Table A.26: Performance on Markov-chain graphs, p=100, n=400, $\delta = 0.05$

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.200 (0.018)	0.197 (0.009)	0.454 (0.045)	0.334 (0.037)
TPR	0.471 (0.032)	0.529 (0.019)	0.169 (0.041)	0.292 (0.033)
MATCHING PRESENCE: PPV	0.166 (0.036)	0.098 (0.014)	0.254 (0.052)	0.187 (0.038)
MATCHING PRESENCE: TPR	0.317 (0.058)	0.467 (0.053)	0.283 (0.070)	0.294 (0.051)
MISMATCHING: PPV	0.177 (0.017)	0.178 (0.009)	0.771 (0.087)	0.344 (0.039)
MISMATCHING: TPR	0.448 (0.026)	0.320 (0.017)	0.038 (0.015)	0.194 (0.025)
EAGD	0.162 (0.022)	0.183 (0.007)	0.026 (0.008)	0.060 (0.012)
PAGD	0.068 (0.001)	0.068 (0.001)	0.068 (0.001)	0.068 (0.001)

Table A.27: Performance on Markov-chain graphs, p=100, n=400, $\delta = 0.20$

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.342 (0.016)	0.452 (0.012)	0.482 (0.029)	0.419 (0.019)
TPR	0.637 (0.036)	0.487 (0.017)	0.291 (0.049)	0.456 (0.031)
MATCHING PRESENCE: PPV	0.140 (0.016)	0.176 (0.015)	0.203 (0.029)	0.153 (0.017)
MATCHING PRESENCE: TPR	0.465 (0.056)	0.445 (0.032)	0.425 (0.063)	0.435 (0.039)
MISMATCHING: PPV	0.374 (0.017)	0.491 (0.012)	0.746 (0.045)	0.503 (0.027)
MISMATCHING: TPR	0.514 (0.010)	0.297 (0.010)	0.084 (0.018)	0.257 (0.016)
EAGD	0.395 (0.042)	0.228 (0.011)	0.129 (0.026)	0.231 (0.025)
PAGD	0.211 (0.001)	0.211 (0.001)	0.211 (0.001)	0.211 (0.001)

Table A.28: Performance on tree graphs, p=100, n=400, $\delta = 0.05$

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.080 (0.013)	0.076 (0.021)	0.406 (0.225)	0.293 (0.071)
TPR	0.368 (0.038)	0.453 (0.044)	0.044 (0.041)	0.131 (0.043)
MATCHING PRESENCE: PPV	0.038 (0.020)	0.019 (0.009)	0.122 (0.112)	0.099 (0.075)
MATCHING PRESENCE: TPR	0.229 (0.126)	0.404 (0.110)	0.117 (0.116)	0.154 (0.112)
MISMATCHING: PPV	0.085 (0.012)	0.083 (0.022)	0.616 (0.460)	0.326 (0.079)
MISMATCHING: TPR	0.371 (0.034)	0.305 (0.027)	0.010 (0.013)	0.101 (0.027)
EAGD	0.094 (0.014)	0.129 (0.039)	0.002 (0.003)	0.010 (0.004)
PAGD	0.020 (0.000)	0.020 (0.000)	0.020 (0.000)	0.020 (0.000)

Table A.29: Performance on tree graphs, p=100, n=400, $\delta = 0.20$

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.082 (0.013)	0.082 (0.019)	0.434 (0.294)	0.275 (0.067)
TPR	0.387 (0.045)	0.447 (0.065)	0.050 (0.045)	0.151 (0.060)
MATCHING PRESENCE: PPV	0.041 (0.027)	0.023 (0.015)	0.237 (0.333)	0.093 (0.066)
MATCHING PRESENCE: TPR	0.253 (0.103)	0.464 (0.160)	0.143 (0.127)	0.204 (0.109)
MISMATCHING: PPV	0.087 (0.014)	0.090 (0.019)	0.560 (0.473)	0.322 (0.063)
MISMATCHING: TPR	0.388 (0.055)	0.304 (0.047)	0.014 (0.016)	0.116 (0.045)
EAGD	0.097 (0.023)	0.118 (0.041)	0.003 (0.003)	0.012 (0.006)
PAGD	0.020 (0.000)	0.020 (0.000)	0.020 (0.000)	0.020 (0.000)

Table A.30: Performance on scale-free graphs, p=100, n=400, $\delta = 0.20$

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.372 (0.015)	0.492 (0.038)	0.489 (0.021)	0.447 (0.018)
TPR	0.667 (0.029)	0.516 (0.045)	0.398 (0.050)	0.497 (0.040)
MATCHING PRESENCE: PPV	0.203 (0.021)	0.260 (0.035)	0.253 (0.019)	0.223 (0.021)
MATCHING PRESENCE: TPR	0.509 (0.036)	0.478 (0.039)	0.501 (0.051)	0.474 (0.040)
MISMATCHING: PPV	0.369 (0.015)	0.481 (0.041)	0.657 (0.042)	0.490 (0.022)
MISMATCHING: TPR	0.517 (0.015)	0.291 (0.011)	0.117 (0.023)	0.271 (0.022)
EAGD	0.443 (0.035)	0.261 (0.048)	0.202 (0.034)	0.274 (0.031)
PAGD	0.246 (0.002)	0.246 (0.002)	0.246 (0.002)	0.246 (0.002)

Table A.31: Performance on random graphs ($\pi = 0.1$), p=100, n=60, $\delta = 0.05$

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.241 (0.028)	0.235 (0.017)	0.28 (0.307)	0.332 (0.348)
TPR	0.089 (0.044)	0.170 (0.033)	0.005 (0.007)	0.005 (0.007)
MATCHING PRESENCE: PPV	0.284 (0.095)	0.196 (0.021)	0.257 (0.288)	0.240 (0.356)
MATCHING PRESENCE: TPR	0.018 (0.009)	0.107 (0.029)	0.006 (0.009)	0.003 (0.005)
MISMATCHING: PPV	0.125 (0.014)	0.124 (0.013)	0.025 (0.109)	0.135 (0.182)
MISMATCHING: TPR	0.122 (0.066)	0.131 (0.020)	0.000 (0.000)	0.004 (0.006)
EAGD	0.054 (0.034)	0.101 (0.022)	0.001 (0.002)	0.002 (0.003)
PAGD	0.139 (0.003)	0.139 (0.003)	0.139 (0.003)	0.139 (0.003)

Table A.32: Performance on random graphs ($\pi = 0.1$), p=100, n=60, $\delta = 0.20$

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.411 (0.036)	0.394 (0.025)	0.325 (0.314)	0.653 (0.303)
TPR	0.096 (0.029)	0.169 (0.020)	0.005 (0.007)	0.008 (0.017)
MATCHING PRESENCE: PPV	0.275 (0.106)	0.204 (0.026)	0.215 (0.259)	0.288 (0.349)
MATCHING PRESENCE: TPR	0.019 (0.009)	0.116 (0.016)	0.008 (0.012)	0.005 (0.010)
MISMATCHING: PPV	0.385 (0.020)	0.386 (0.016)	0.050 (0.218)	0.442 (0.28)
MISMATCHING: TPR	0.129 (0.045)	0.132 (0.016)	0.000 (0.000)	0.005 (0.012)
EAGD	0.062 (0.025)	0.112 (0.016)	0.002 (0.003)	0.004 (0.011)
PAGD	0.259 (0.004)	0.259 (0.004)	0.259 (0.004)	0.259 (0.004)

Table A.33: Performance on Markov-chain graphs, p=100, n=60, $\delta = 0.05$

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.140 (0.018)	0.136 (0.018)	0.130 (0.247)	0.174 (0.247)
TPR	0.075 (0.021)	0.133 (0.027)	0.003 (0.006)	0.003 (0.005)
MATCHING PRESENCE: PPV	0.097 (0.086)	0.059 (0.023)	0.117 (0.247)	0.040 (0.102)
MATCHING PRESENCE: TPR	0.015 (0.011)	0.082 (0.031)	0.007 (0.018)	0.002 (0.005)
MISMATCHING: PPV	0.147 (0.022)	0.146 (0.018)	0.000 (0.000)	0.166 (0.193)
MISMATCHING: TPR	0.103 (0.029)	0.116 (0.019)	0.000 (0.000)	0.003 (0.005)
EAGD	0.036 (0.010)	0.067 (0.018)	0.001 (0.001)	0.001 (0.001)
PAGD	0.068 (0.001)	0.068 (0.001)	0.068 (0.001)	0.068 (0.001)

Table A.34: Performance on Markov-chain graphs, p=100, n=60, $\delta = 0.20$

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.362 (0.032)	0.322 (0.015)	0.265 (0.297)	0.611 (0.28)
TPR	0.086 (0.017)	0.163 (0.023)	0.004 (0.005)	0.005 (0.005)
MATCHING PRESENCE: PPV	0.192 (0.091)	0.113 (0.021)	0.171 (0.246)	0.273 (0.368)
MATCHING PRESENCE: TPR	0.019 (0.007)	0.113 (0.025)	0.009 (0.012)	0.005 (0.007)
MISMATCHING: PPV	0.399 (0.027)	0.387 (0.018)	0.125 (0.311)	0.548 (0.296)
MISMATCHING: TPR	0.114 (0.022)	0.133 (0.019)	0.000 (0.000)	0.004 (0.003)
EAGD	0.051 (0.011)	0.107 (0.017)	0.002 (0.002)	0.002 (0.002)
PAGD	0.211 (0.001)	0.211 (0.001)	0.211 (0.001)	0.211 (0.001)

Table A.35: Performance on tree graphs, p=100, n=60, $\delta = 0.05$

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.046 (0.014)	0.040 (0.008)	0.104 (0.299)	0.058 (0.158)
TPR	0.077 (0.032)	0.124 (0.031)	0.002 (0.003)	0.002 (0.004)
MATCHING PRESENCE: PPV	0.015 (0.033)	0.012 (0.011)	0.100 (0.300)	0.050 (0.218)
MATCHING PRESENCE: TPR	0.019 (0.041)	0.106 (0.082)	0.007 (0.022)	0.004 (0.017)
MISMATCHING: PPV	0.057 (0.013)	0.054 (0.013)	0.050 (0.218)	0.073 (0.226)
MISMATCHING: TPR	0.101 (0.037)	0.113 (0.027)	0.000 (0.001)	0.001 (0.003)
EAGD	0.033 (0.008)	0.063 (0.016)	0.000 (0.001)	0.000 (0.001)
PAGD	0.020 (0.000)	0.020 (0.000)	0.020 (0.000)	0.020 (0.000)

Table A.36: Performance on scale-free graphs, p=100, n=60, $\delta = 0.05$

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.234 (0.024)	0.218 (0.021)	0.340 (0.398)	0.335 (0.328)
TPR	0.077 (0.018)	0.162 (0.035)	0.004 (0.007)	0.005 (0.006)
MATCHING PRESENCE: PPV	0.327 (0.136)	0.181 (0.034)	0.307 (0.376)	0.354 (0.417)
MATCHING PRESENCE: TPR	0.017 (0.008)	0.105 (0.034)	0.006 (0.009)	0.003 (0.004)
MISMATCHING: PPV	0.131 (0.017)	0.125 (0.015)	0.025 (0.109)	0.126 (0.146)
MISMATCHING: TPR	0.097 (0.021)	0.124 (0.021)	0.000 (0.000)	0.004 (0.005)
EAGD	0.039 (0.007)	0.091 (0.024)	0.001 (0.002)	0.001 (0.002)
PAGD	0.121 (0.001)	0.121 (0.001)	0.121 (0.001)	0.121 (0.001)

Table A.37: Performance on scale-free graphs, p=100, n=60, $\delta = 0.20$

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.401 (0.024)	0.368 (0.014)	0.224 (0.319)	0.490 (0.317)
TPR	0.094 (0.030)	0.174 (0.015)	0.003 (0.006)	0.005 (0.007)
MATCHING PRESENCE: PPV	0.317 (0.179)	0.179 (0.023)	0.189 (0.282)	0.105 (0.172)
MATCHING PRESENCE: TPR	0.020 (0.011)	0.121 (0.016)	0.006 (0.012)	0.003 (0.006)
MISMATCHING: PPV	0.383 (0.022)	0.379 (0.016)	0.025 (0.109)	0.471 (0.292)
MISMATCHING: TPR	0.125 (0.043)	0.135 (0.007)	0.000 (0.000)	0.004 (0.005)
EAGD	0.059 (0.022)	0.117 (0.010)	0.001 (0.002)	0.002 (0.003)
PAGD	0.247 (0.002)	0.247 (0.002)	0.247 (0.002)	0.247 (0.002)

A.2.2 Differential GSL

Table A.38: Performance on differences of random graphs ($\pi = 0.1$), p=30, n=200, δ=0.05

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.233 (0.024)	0.373 (0.062)	0.572 (0.176)	0.414 (0.148)	0.212 (0.027)
DIFFERENCE: TPR	0.822 (0.064)	0.629 (0.074)	0.169 (0.067)	0.533 (0.142)	0.586 (0.279)
EGD	0.629 (0.092)	0.305 (0.065)	0.063 (0.046)	0.259 (0.110)	0.521 (0.304)
PGD	0.177 (0.018)	0.177 (0.018)	0.177 (0.018)	0.177 (0.018)	0.177 (0.018)

Table A.39: Performance on differences of random graphs ($\pi = 0.1$), p=30, n=200, δ=0.20

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.464 (0.020)	0.617 (0.040)	0.666 (0.061)	0.578 (0.039)	0.438 (0.031)
DIFFERENCE: TPR	0.907 (0.041)	0.682 (0.059)	0.375 (0.103)	0.686 (0.065)	0.832 (0.224)
EGD	0.828 (0.053)	0.470 (0.062)	0.243 (0.078)	0.504 (0.063)	0.813 (0.238)
PGD	0.422 (0.014)	0.422 (0.014)	0.422 (0.014)	0.422 (0.014)	0.422 (0.014)

Table A.40: Performance on differences of Markov-Chain graphs, p=30, n=200, δ=0.05

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.203 (0.018)	0.325 (0.043)	0.600 (0.173)	0.342 (0.052)	0.173 (0.034)
DIFFERENCE: TPR	0.780 (0.063)	0.620 (0.094)	0.168 (0.090)	0.512 (0.106)	0.621 (0.260)
EGD	0.545 (0.074)	0.277 (0.072)	0.044 (0.034)	0.218 (0.070)	0.547 (0.290)
PGD	0.141 (0.008)	0.141 (0.008)	0.141 (0.008)	0.141 (0.008)	0.141 (0.008)

Table A.41: Performance on differences of Markov-Chain graphs, p=30, n=200, δ=0.20

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.449 (0.016)	0.611 (0.049)	0.683 (0.067)	0.562 (0.034)	0.418 (0.025)
DIFFERENCE: TPR	0.878 (0.031)	0.659 (0.043)	0.346 (0.088)	0.670 (0.064)	0.854 (0.207)
EGD	0.779 (0.040)	0.433 (0.051)	0.206 (0.065)	0.479 (0.066)	0.827 (0.221)
PGD	0.398 (0.008)	0.398 (0.008)	0.398 (0.008)	0.398 (0.008)	0.398 (0.008)

Table A.42: Performance on differences of tree graphs, p=30, n=200, δ=0.05

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.119 (0.031)	0.191 (0.073)	0.654 (0.250)	0.267 (0.103)	0.098 (0.018)
DIFFERENCE: TPR	0.711 (0.116)	0.575 (0.111)	0.163 (0.076)	0.393 (0.126)	0.552 (0.210)
EGD	0.456 (0.078)	0.240 (0.063)	0.019 (0.010)	0.121 (0.049)	0.440 (0.237)
PGD	0.074 (0.010)	0.074 (0.010)	0.074 (0.010)	0.074 (0.010)	0.074 (0.010)

Table A.43: Performance on differences of tree graphs, p=30, n=200, δ=0.20

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.178 (0.027)	0.272 (0.052)	0.765 (0.173)	0.388 (0.116)	0.145 (0.025)
DIFFERENCE: TPR	0.707 (0.095)	0.586 (0.100)	0.157 (0.104)	0.393 (0.126)	0.534 (0.19)
EGD	0.449 (0.075)	0.250 (0.066)	0.026 (0.020)	0.126 (0.057)	0.428 (0.203)
PGD	0.112 (0.010)	0.112 (0.010)	0.112 (0.010)	0.112 (0.010)	0.112 (0.010)

Table A.44: Performance on differences of scale-free graphs, p=30, n=200, δ=0.05

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.21 (0.034)	0.318 (0.059)	0.575 (0.185)	0.371 (0.067)	0.188 (0.026)
DIFFERENCE: TPR	0.781 (0.064)	0.638 (0.064)	0.188 (0.071)	0.518 (0.103)	0.557 (0.149)
EGD	0.553 (0.095)	0.305 (0.076)	0.053 (0.024)	0.214 (0.076)	0.444 (0.158)
PGD	0.146 (0.006)	0.146 (0.006)	0.146 (0.006)	0.146 (0.006)	0.146 (0.006)

Table A.45: Performance on differences of random graphs ($\pi = 0.1$), p=30, n=40, δ=0.20

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.480 (0.030)	0.546 (0.047)	0.457 (0.402)	0.645 (0.124)	0.200 (0.225)
DIFFERENCE: TPR	0.583 (0.139)	0.425 (0.075)	0.025 (0.035)	0.160 (0.118)	0.152 (0.297)
EGD	0.515 (0.142)	0.330 (0.069)	0.015 (0.022)	0.114 (0.093)	0.151 (0.298)
PGD	0.420 (0.012)	0.420 (0.012)	0.420 (0.012)	0.420 (0.012)	0.420 (0.012)

Table A.46: Performance on differences of Markov-chain graphs, p=30, n=40, δ=0.05

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.218 (0.042)	0.233 (0.041)	0.297 (0.421)	0.419 (0.245)	0.098 (0.071)
DIFFERENCE: TPR	0.414 (0.124)	0.396 (0.094)	0.011 (0.019)	0.110 (0.085)	0.138 (0.161)
EGD	0.291 (0.115)	0.251 (0.066)	0.004 (0.007)	0.049 (0.054)	0.141 (0.155)
PGD	0.145 (0.008)	0.145 (0.008)	0.145 (0.008)	0.145 (0.008)	0.145 (0.008)

Table A.47: Performance on differences of Markov-chain graphs, p=30, n=40, δ=0.20

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.457 (0.041)	0.524 (0.038)	0.335 (0.359)	0.653 (0.117)	0.137 (0.189)
DIFFERENCE: TPR	0.596 (0.170)	0.415 (0.078)	0.012 (0.015)	0.120 (0.061)	0.091 (0.146)
EGD	0.528 (0.180)	0.316 (0.067)	0.008 (0.010)	0.076 (0.047)	0.090 (0.145)
PGD	0.396 (0.009)	0.396 (0.009)	0.396 (0.009)	0.396 (0.009)	0.396 (0.009)

Table A.48: Performance on differences of tree graphs, p=30, n=40, δ=0.05

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.112 (0.018)	0.139 (0.037)	0.181 (0.361)	0.184 (0.292)	0.032 (0.043)
DIFFERENCE: TPR	0.366 (0.132)	0.356 (0.132)	0.008 (0.014)	0.048 (0.059)	0.084 (0.139)
EGD	0.239 (0.086)	0.196 (0.076)	0.002 (0.005)	0.022 (0.022)	0.086 (0.126)
PGD	0.074 (0.009)	0.074 (0.009)	0.074 (0.009)	0.074 (0.009)	0.074 (0.009)

Table A.49: Performance on differences of tree graphs, p=30, n=40, δ=0.20

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.162 (0.033)	0.195 (0.044)	0.075 (0.238)	0.318 (0.266)	0.046 (0.079)
DIFFERENCE: TPR	0.328 (0.101)	0.320 (0.104)	0.003 (0.010)	0.040 (0.044)	0.040 (0.073)
EGD	0.231 (0.070)	0.190 (0.071)	0.000 (0.001)	0.016 (0.018)	0.043 (0.067)
PGD	0.112 (0.008)	0.112 (0.008)	0.112 (0.008)	0.112 (0.008)	0.112 (0.008)

Table A.50: Performance on differences of scale-free graphs, p=30, n=40, δ=0.05

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.217 (0.033)	0.244 (0.037)	0.118 (0.198)	0.425 (0.213)	0.038 (0.061)
DIFFERENCE: TPR	0.464 (0.137)	0.416 (0.100)	0.010 (0.017)	0.093 (0.066)	0.117 (0.262)
EGD	0.329 (0.121)	0.262 (0.079)	0.004 (0.007)	0.038 (0.033)	0.123 (0.271)
PGD	0.150 (0.006)	0.150 (0.006)	0.150 (0.006)	0.150 (0.006)	0.150 (0.006)

Table A.51: Performance on differences of scale-free graphs, p=30, n=40, δ=0.20

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.406 (0.035)	0.524 (0.050)	0.509 (0.439)	0.664 (0.114)	0.157 (0.268)
DIFFERENCE: TPR	0.524 (0.143)	0.388 (0.066)	0.014 (0.017)	0.128 (0.076)	0.149 (0.317)
EGD	0.461 (0.148)	0.300 (0.066)	0.008 (0.011)	0.083 (0.056)	0.146 (0.317)
PGD	0.399 (0.009)	0.399 (0.009)	0.399 (0.009)	0.399 (0.009)	0.399 (0.009)

Table A.52: Performance on differences of random graphs ($\pi = 0.1$), p=30, n=20, δ=0.05

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.241 (0.033)	0.269 (0.065)	0.123 (0.269)	0.222 (0.179)	0.068 (0.094)
DIFFERENCE: TPR	0.300 (0.130)	0.345 (0.175)	0.006 (0.015)	0.082 (0.093)	0.189 (0.338)
EGD	0.231 (0.106)	0.266 (0.195)	0.002 (0.005)	0.049 (0.058)	0.186 (0.336)
PGD	0.184 (0.012)	0.184 (0.012)	0.184 (0.012)	0.184 (0.012)	0.184 (0.012)

Table A.53: Performance on differences of random graphs ($\pi = 0.1$), p=30, n=20, δ=0.20

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.481 (0.042)	0.467 (0.062)	0.170 (0.330)	0.463 (0.344)	0.259 (0.215)
DIFFERENCE: TPR	0.446 (0.191)	0.563 (0.255)	0.003 (0.007)	0.065 (0.070)	0.218 (0.322)
EGD	0.400 (0.200)	0.533 (0.274)	0.003 (0.005)	0.046 (0.052)	0.217 (0.324)
PGD	0.419 (0.013)	0.419 (0.013)	0.419 (0.013)	0.419 (0.013)	0.419 (0.013)

Table A.54: Performance on differences of Markov-chain graphs, p=30, n=20, δ=0.05

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.189 (0.053)	0.181 (0.051)	0.034 (0.091)	0.125 (0.246)	0.045 (0.062)
DIFFERENCE: TPR	0.272 (0.114)	0.331 (0.219)	0.005 (0.012)	0.013 (0.030)	0.180 (0.280)
EGD	0.210 (0.101)	0.276 (0.225)	0.003 (0.008)	0.006 (0.012)	0.183 (0.276)
PGD	0.138 (0.009)	0.138 (0.009)	0.138 (0.009)	0.138 (0.009)	0.138 (0.009)

Table A.55: Performance on differences of Markov-chain graphs, p=30, n=20, δ=0.20

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.471 (0.040)	0.454 (0.046)	0.199 (0.331)	0.483 (0.322)	0.165 (0.187)
DIFFERENCE: TPR	0.382 (0.173)	0.510 (0.265)	0.006 (0.014)	0.054 (0.107)	0.249 (0.365)
EGD	0.331 (0.162)	0.472 (0.274)	0.005 (0.012)	0.039 (0.083)	0.254 (0.369)
PGD	0.401 (0.009)	0.401 (0.009)	0.401 (0.009)	0.401 (0.009)	0.401 (0.009)

Table A.56: Performance on differences of tree graphs, p=30, n=20, δ=0.05

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.094 (0.035)	0.102 (0.038)	0.161 (0.354)	0.126 (0.239)	0.053 (0.040)
DIFFERENCE: TPR	0.262 (0.149)	0.309 (0.191)	0.022 (0.063)	0.032 (0.074)	0.295 (0.346)
EGD	0.205 (0.090)	0.244 (0.197)	0.015 (0.052)	0.012 (0.032)	0.302 (0.343)
PGD	0.076 (0.009)	0.076 (0.009)	0.076 (0.009)	0.076 (0.009)	0.076 (0.009)

Table A.57: Performance on differences of tree graphs, p=30, n=20, δ=0.20

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.134 (0.035)	0.138 (0.038)	0.192 (0.362)	0.178 (0.355)	0.047 (0.092)
DIFFERENCE: TPR	0.255 (0.087)	0.376 (0.255)	0.005 (0.009)	0.008 (0.016)	0.197 (0.335)
EGD	0.204 (0.058)	0.322 (0.262)	0.001 (0.002)	0.003 (0.005)	0.194 (0.333)
PGD	0.106 (0.007)	0.106 (0.007)	0.106 (0.007)	0.106 (0.007)	0.106 (0.007)

Table A.58: Performance on differences of scale-free graphs, p=30, n=20, δ=0.05

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.180 (0.042)	0.188 (0.044)	0.050 (0.218)	0.252 (0.354)	0.081 (0.077)
DIFFERENCE: TPR	0.290 (0.114)	0.391 (0.263)	0.002 (0.007)	0.016 (0.024)	0.218 (0.303)
EGD	0.234 (0.088)	0.332 (0.271)	0.000 (0.001)	0.008 (0.010)	0.224 (0.311)
PGD	0.146 (0.007)	0.146 (0.007)	0.146 (0.007)	0.146 (0.007)	0.146 (0.007)

Table A.59: Performance on differences of scale-free graphs, p=30, n=20, δ=0.20

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.457 (0.048)	0.442 (0.049)	0.130 (0.279)	0.483 (0.359)	0.116 (0.178)
DIFFERENCE: TPR	0.377 (0.188)	0.592 (0.271)	0.006 (0.017)	0.023 (0.023)	0.221 (0.386)
EGD	0.342 (0.188)	0.557 (0.279)	0.005 (0.016)	0.016 (0.018)	0.224 (0.387)
PGD	0.400 (0.009)	0.400 (0.009)	0.400 (0.009)	0.400 (0.009)	0.400 (0.009)

Table A.60: Performance on differences of random graphs ($\pi = 0.1$), p=100, n=400, δ=0.20

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.484 (0.012)	0.620 (0.042)	0.726 (0.033)	0.590 (0.019)	0.000 (0.000)
DIFFERENCE: TPR	0.799 (0.029)	0.575 (0.063)	0.172 (0.030)	0.546 (0.048)	0.000 (0.000)
EGD	0.690 (0.041)	0.391 (0.072)	0.100 (0.021)	0.388 (0.046)	0.000 (0.000)
PGD	0.417 (0.003)	0.417 (0.003)	0.417 (0.003)	0.417 (0.003)	0.417 (0.003)

Table A.61: Performance on differences of Markov-chain graphs, p=100, n=400, δ=0.05

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.217 (0.018)	0.238 (0.009)	0.827 (0.067)	0.386 (0.038)	0.092 (0.219)
DIFFERENCE: TPR	0.542 (0.036)	0.568 (0.018)	0.052 (0.018)	0.295 (0.037)	0.012 (0.019)
EGD	0.284 (0.035)	0.269 (0.010)	0.007 (0.003)	0.087 (0.016)	0.011 (0.018)
PGD	0.112 (0.001)	0.112 (0.001)	0.112 (0.001)	0.112 (0.001)	0.112 (0.001)

Table A.62: Performance on differences of Markov-chain graphs, p=100, n=400, δ=0.20

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.452 (0.014)	0.600 (0.010)	0.799 (0.037)	0.573 (0.018)	0.000 (0.000)
DIFFERENCE: TPR	0.748 (0.038)	0.524 (0.019)	0.123 (0.029)	0.480 (0.035)	0.000 (0.000)
EGD	0.614 (0.052)	0.324 (0.014)	0.058 (0.016)	0.311 (0.031)	0.000 (0.000)
PGD	0.370 (0.002)	0.370 (0.002)	0.370 (0.002)	0.370 (0.002)	0.370 (0.002)

Table A.63: Performance on differences of tree graphs, p=100, n=400, δ=0.05

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.086 (0.011)	0.092 (0.021)	0.729 (0.424)	0.315 (0.075)	0.028 (0.048)
DIFFERENCE: TPR	0.414 (0.034)	0.484 (0.051)	0.016 (0.020)	0.129 (0.038)	0.009 (0.018)
EGD	0.171 (0.024)	0.197 (0.055)	0.001 (0.001)	0.015 (0.006)	0.007 (0.013)
PGD	0.035 (0.001)	0.035 (0.001)	0.035 (0.001)	0.035 (0.001)	0.035 (0.001)

Table A.64: Performance on differences of tree graphs, p=100, n=400, δ=0.20

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.088 (0.013)	0.097 (0.018)	0.672 (0.448)	0.299 (0.069)	0.016 (0.032)
DIFFERENCE: TPR	0.433 (0.064)	0.466 (0.078)	0.020 (0.020)	0.145 (0.057)	0.020 (0.041)
EGD	0.178 (0.042)	0.182 (0.058)	0.001 (0.001)	0.019 (0.010)	0.011 (0.018)
PGD	0.036 (0.002)	0.036 (0.002)	0.036 (0.002)	0.036 (0.002)	0.036 (0.002)

Table A.65: Performance on differences of scale-free graphs, p=100, n=400, δ=0.05

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.258 (0.011)	0.339 (0.013)	0.637 (0.093)	0.403 (0.035)	0.042 (0.138)
DIFFERENCE: TPR	0.667 (0.038)	0.579 (0.002)	0.075 (0.021)	0.410 (0.044)	0.001 (0.003)
EGD	0.432 (0.040)	0.285 (0.010)	0.020 (0.006)	0.172 (0.031)	0.000 (0.001)
PGD	0.167 (0.002)	0.167 (0.002)	0.167 (0.002)	0.167 (0.002)	0.167 (0.002)

Table A.66: Performance on differences of scale-free graphs, p=100, n=400, δ=0.20

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.478 (0.011)	0.631 (0.034)	0.734 (0.034)	0.592 (0.014)	0.000 (0.000)
DIFFERENCE: TPR	0.785 (0.033)	0.554 (0.051)	0.168 (0.032)	0.527 (0.044)	0.000 (0.000)
EGD	0.665 (0.040)	0.358 (0.055)	0.093 (0.022)	0.361 (0.038)	0.000 (0.000)
PGD	0.405 (0.002)	0.405 (0.002)	0.405 (0.002)	0.405 (0.002)	0.405 (0.002)

Table A.67: Performance on differences of random graphs ($\pi = 0.1$), p=100, n=60, δ=0.05

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.267 (0.028)	0.27 (0.019)	0.025 (0.109)	0.354 (0.336)	0.105 (0.117)
DIFFERENCE:TPR	0.143 (0.078)	0.221 (0.039)	0.000 (0.000)	0.006 (0.010)	0.198 (0.331)
EGD	0.101 (0.065)	0.152 (0.030)	0.000 (0.000)	0.003 (0.005)	0.198 (0.331)
PGD	0.185 (0.003)	0.185 (0.003)	0.185 (0.003)	0.185 (0.003)	0.185 (0.003)

Table A.68: Performance on differences of random graphs ($\pi = 0.1$), p=100, n=60, δ=0.20

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.532 (0.030)	0.533 (0.023)	0.050 (0.218)	0.678 (0.299)	0.102 (0.178)
DIFFERENCE: TPR	0.145 (0.049)	0.211 (0.026)	0.000 (0.000)	0.009 (0.021)	0.072 (0.203)
EGD	0.116 (0.047)	0.166 (0.023)	0.000 (0.000)	0.006 (0.017)	0.073 (0.204)
PGD	0.418 (0.005)	0.418 (0.005)	0.418 (0.005)	0.418 (0.005)	0.418 (0.005)

Table A.69: Performance on differences of Markov-chain graphs, p=100, n=60, δ=0.20

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.481 (0.029)	0.469 (0.013)	0.133 (0.323)	0.688 (0.286)	0.184 (0.214)
DIFFERENCE: TPR	0.125 (0.025)	0.205 (0.030)	0.000 (0.000)	0.005 (0.005)	0.046 (0.089)
EGD	0.096 (0.021)	0.162 (0.024)	0.000 (0.000)	0.003 (0.003)	0.045 (0.087)
PGD	0.370 (0.002)	0.370 (0.002)	0.370 (0.002)	0.370 (0.002)	0.370 (0.002)

Table A.70: Performance on differences of tree graphs, p=100, n=60, δ=0.05

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.057 (0.012)	0.053 (0.010)	0.050 (0.218)	0.071 (0.226)	0.017 (0.019)
DIFFERENCE: TPR	0.105 (0.038)	0.151 (0.037)	0.000 (0.001)	0.001 (0.003)	0.095 (0.183)
EGD	0.063 (0.015)	0.100 (0.022)	0.000 (0.000)	0.000 (0.001)	0.096 (0.179)
PGD	0.035 (0.001)	0.035 (0.001)	0.035 (0.001)	0.035 (0.001)	0.035 (0.001)

Table A.71: Performance on differences of tree graphs, p=100, n=60, δ=0.20

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.06 (0.020)	0.056 (0.013)	0.000 (0.000)	0.093 (0.148)	0.023 (0.026)
DIFFERENCE:TPR	0.107 (0.056)	0.152 (0.040)	0.000 (0.000)	0.008 (0.014)	0.144 (0.227)
EGD	0.065 (0.031)	0.099 (0.025)	0.000 (0.000)	0.002 (0.003)	0.146 (0.232)
PGD	0.036 (0.001)	0.036 (0.001)	0.036 (0.001)	0.036 (0.001)	0.036 (0.001)

Table A.72: Performance on differences of scale-free graphs, p=100, n=60, δ=0.05

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.266 (0.025)	0.254 (0.020)	0.075 (0.238)	0.36 (0.337)	0.124 (0.083)
DIFFERENCE: TPR	0.120 (0.025)	0.209 (0.040)	0.000 (0.000)	0.006 (0.007)	0.231 (0.331)
EGD	0.075 (0.013)	0.139 (0.033)	0.000 (0.000)	0.002 (0.003)	0.227 (0.332)
PGD	0.167 (0.002)	0.167 (0.002)	0.167 (0.002)	0.167 (0.002)	0.167 (0.002)

Table A.73: Performance on differences of scale-free graphs, p=100, n=60, δ=0.20

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.519 (0.021)	0.510 (0.014)	0.025 (0.109)	0.599 (0.329)	0.161 (0.272)
DIFFERENCE: TPR	0.141 (0.048)	0.218 (0.016)	0.000 (0.000)	0.006 (0.008)	0.076 (0.187)
EGD	0.112 (0.042)	0.174 (0.012)	0.000 (0.000)	0.004 (0.004)	0.074 (0.185)
PGD	0.406 (0.003)	0.406 (0.003)	0.406 (0.003)	0.406 (0.003)	0.406 (0.003)

A.3 Performance results: model selection with fixed densities

A.3.1 Separate and Joint GSL

Table A.74: Performance on random graphs (π = 0.1) with fixed densities, p=30, n=200, δ=0.05

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.453	0.500	0.469	0.460
TPR	0.453	0.511	0.482	0.460
MATCHING PRESENCE: PPV	0.609	0.469	0.373	0.458
MATCHING PRESENCE: TPR	0.286	0.469	0.510	0.449
MISMATCHING: PPV	0.204	0.205	0.556	0.233
MISMATCHING: TPR	0.463	0.220	0.122	0.244
EAGD	0.160	0.163	0.164	0.160
PAGD	0.160	0.160	0.160	0.160

Table A.75: Performance on Markov-chain graphs with fixed densities, p=30, n=200, δ=0.05

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.440	0.495	0.500	0.467
TPR	0.449	0.531	0.500	0.439
MATCHING PRESENCE: PPV	0.667	0.515	0.419	0.448
MATCHING PRESENCE: TPR	0.414	0.586	0.621	0.448
MISMATCHING: PPV	0.219	0.205	0.583	0.294
MISMATCHING: TPR	0.350	0.200	0.175	0.250
EAGD	0.115	0.121	0.113	0.106
PAGD	0.113	0.113	0.113	0.113

Table A.76: Performance on tree graphs with fixed densities, p=30, n=200, δ=0.05

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.316	0.509	0.439	0.361
TPR	0.414	0.466	0.431	0.379
MATCHING PRESENCE: PPV	0.857	0.471	0.320	0.389
MATCHING PRESENCE: TPR	0.429	0.571	0.571	0.500
MISMATCHING: PPV	0.177	0.421	0.571	0.280
MISMATCHING: TPR	0.367	0.267	0.133	0.233
EAGD	0.087	0.061	0.066	0.070
PAGD	0.067	0.067	0.067	0.067

Table A.77: Performance on scale-free graphs with fixed densities, p=30, n=200, δ=0.05

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.423	0.495	0.441	0.475
TPR	0.431	0.451	0.441	0.461
MATCHING PRESENCE: PPV	0.714	0.462	0.362	0.500
MATCHING PRESENCE: TPR	0.323	0.387	0.548	0.484
MISMATCHING: PPV	0.224	0.317	0.625	0.333
MISMATCHING: TPR	0.425	0.325	0.125	0.325
EAGD	0.120	0.107	0.117	0.114
PAGD	0.117	0.117	0.117	0.117

Table A.78: Performance on random graphs ($\pi = 0.1$) with fixed densities, p=30, n=40, δ=0.20

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.388	0.394	0.370	0.376
TPR	0.367	0.396	0.379	0.383
MATCHING PRESENCE: PPV	0.250	0.190	0.188	0.159
MATCHING PRESENCE: TPR	0.191	0.319	0.404	0.298
MISMATCHING: PPV	0.374	0.398	0.477	0.391
MISMATCHING: TPR	0.397	0.226	0.144	0.185
EAGD	0.261	0.277	0.283	0.282
PAGD	0.276	0.276	0.276	0.276

Table A.79: Performance on tree graphs with fixed densities, p=30, n=40, δ=0.20

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.186	0.231	0.235	0.203
TPR	0.190	0.207	0.207	0.207
MATCHING PRESENCE: PPV	0.000	0.000	0.095	0.083
MATCHING PRESENCE: TPR	0.000	0.000	1.000	0.500
MISMATCHING: PPV	0.204	0.233	0.444	0.229
MISMATCHING: TPR	0.185	0.130	0.074	0.148
EAGD	0.068	0.060	0.059	0.068
PAGD	0.067	0.067	0.067	0.067

Table A.80: Performance on scale-free graphs with fixed densities, p=30, n=40, δ=0.20

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.321	0.322	0.286	0.299
TPR	0.320	0.345	0.284	0.299
MATCHING PRESENCE: PPV	0.030	0.048	0.087	0.048
MATCHING PRESENCE: TPR	0.038	0.115	0.269	0.115
MISMATCHING: PPV	0.392	0.425	0.361	0.408
MISMATCHING: TPR	0.352	0.255	0.090	0.200
EAGD	0.225	0.243	0.225	0.226
PAGD	0.226	0.226	0.226	0.226

Table A.81: Performance on random graphs ($\pi = 0.1$) with fixed densities, p=30, n=20, δ=0.05

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.270	0.277	0.321	0.281
TPR	0.266	0.266	0.282	0.290
MATCHING PRESENCE: PPV	0.357	0.171	0.295	0.171
MATCHING PRESENCE: TPR	0.119	0.143	0.310	0.167
MISMATCHING: PPV	0.149	0.082	0.190	0.109
MISMATCHING: TPR	0.350	0.100	0.100	0.125
EAGD	0.140	0.137	0.125	0.147
PAGD	0.143	0.143	0.143	0.143

Table A.82: Performance on Markov-chain graphs with fixed densities, p=30, n=20, δ=0.05

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.183	0.143	0.159	0.167
TPR	0.186	0.127	0.167	0.167
MATCHING PRESENCE: PPV	0.167	0.043	0.114	0.057
MATCHING PRESENCE: TPR	0.032	0.032	0.161	0.065
MISMATCHING: PPV	0.152	0.111	0.263	0.125
MISMATCHING: TPR	0.350	0.125	0.125	0.100
EAGD	0.120	0.105	0.123	0.117
PAGD	0.117	0.117	0.117	0.117

Table A.83: Performance on tree graphs with fixed densities, p=30, n=20, δ=0.05

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.155	0.153	0.123	0.158
TPR	0.158	0.228	0.123	0.158
MATCHING PRESENCE: PPV	0.000	0.045	0.038	0.067
MATCHING PRESENCE: TPR	0.000	0.083	0.083	0.083
MISMATCHING: PPV	0.160	0.098	0.000	0.111
MISMATCHING: TPR	0.242	0.121	0.000	0.091
EAGD	0.067	0.098	0.066	0.066
PAGD	0.066	0.066	0.066	0.066

Table A.84: Performance on scale-free graphs with fixed densities, p=30, n=20, δ=0.05

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.181	0.194	0.272	0.207
TPR	0.185	0.207	0.272	0.207
MATCHING PRESENCE: PPV	0.000	0.120	0.243	0.111
MATCHING PRESENCE: TPR	0.000	0.115	0.346	0.115
MISMATCHING: PPV	0.122	0.083	0.167	0.053
MISMATCHING: TPR	0.250	0.100	0.075	0.050
EAGD	0.108	0.113	0.106	0.106
PAGD	0.106	0.106	0.106	0.106

Table A.85: Performance on random graphs ($\pi = 0.1$) with fixed densities, p=100, n=400, δ=0.05

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.407	0.446	0.436	0.414
TPR	0.410	0.452	0.423	0.409
MATCHING PRESENCE: PPV	0.589	0.451	0.350	0.364
MATCHING PRESENCE: TPR	0.259	0.380	0.458	0.352
MISMATCHING: PPV	0.208	0.217	0.437	0.247
MISMATCHING: TPR	0.429	0.290	0.147	0.254
EAGD	0.136	0.138	0.132	0.134
PAGD	0.136	0.136	0.136	0.136

Table A.86: Performance on Markov-chain graphs with fixed densities, p=100, n=400, δ=0.05

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.333	0.388	0.317	0.317
TPR	0.336	0.372	0.327	0.332
MATCHING PRESENCE: PPV	0.419	0.292	0.159	0.170
MATCHING PRESENCE: TPR	0.176	0.343	0.480	0.314
MISMATCHING: PPV	0.277	0.331	0.564	0.343
MISMATCHING: TPR	0.350	0.286	0.094	0.239
EAGD	0.068	0.065	0.070	0.071
PAGD	0.068	0.068	0.068	0.068

Table A.87: Performance on random graphs ($\pi = 0.1$) with fixed densities, p=100, n=60, δ=0.20

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.319	0.316	0.307	0.320
TPR	0.319	0.319	0.311	0.317
MATCHING PRESENCE: PPV	0.138	0.125	0.135	0.133
MATCHING PRESENCE: TPR	0.117	0.225	0.304	0.231
MISMATCHING: PPV	0.339	0.339	0.384	0.353
MISMATCHING: TPR	0.370	0.180	0.100	0.191
EAGD	0.258	0.260	0.260	0.256
PAGD	0.258	0.258	0.258	0.258

Table A.88: Performance on Markov-chain graphs with fixed densities, p=100, n=60, δ=0.20

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.277	0.275	0.274	0.272
TPR	0.278	0.269	0.272	0.274
MATCHING PRESENCE: PPV	0.099	0.084	0.096	0.071
MATCHING PRESENCE: TPR	0.117	0.215	0.324	0.198
MISMATCHING: PPV	0.342	0.334	0.367	0.321
MISMATCHING: TPR	0.325	0.164	0.093	0.145
EAGD	0.212	0.206	0.209	0.212
PAGD	0.211	0.211	0.211	0.211

Table A.89: Performance on scale-free graphs with fixed densities, p=100, n=60, δ=0.20

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.312	0.313	0.315	0.313
TPR	0.311	0.313	0.317	0.313
MATCHING PRESENCE: PPV	0.129	0.120	0.132	0.131
MATCHING PRESENCE: TPR	0.111	0.221	0.327	0.252
MISMATCHING: PPV	0.343	0.343	0.387	0.335
MISMATCHING: TPR	0.367	0.192	0.091	0.172
EAGD	0.244	0.244	0.246	0.244
PAGD	0.244	0.244	0.244	0.244

Table A.90: Performance on tree graphs with fixed densities, p=100, n=60, δ=0.20

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL
PPV	0.025	0.024	0.020	0.026
TPR	0.121	0.247	0.035	0.061
MATCHING PRESENCE: PPV	0.000	0.000	0.000	0.000
MATCHING PRESENCE: TPR	0.000	0.000	0.000	0.000
MISMATCHING: PPV	0.031	0.033	0.029	0.034
MISMATCHING: TPR	0.129	0.144	0.005	0.036
EAGD	0.097	0.210	0.036	0.046
PAGD	0.020	0.020	0.020	0.020

A.3.2 Differential GSL

Table A.91: Performance on differences of random graphs (π = 0.1) with fixed densities, p=30, n=200, δ=0.05

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.455	0.535	0.317	0.453	0.151
DIFFERENCE: TPR	0.465	0.535	0.302	0.453	0.151
EGD	0.202	0.198	0.189	0.198	0.198
PGD	0.198	0.198	0.198	0.198	0.198

Table A.92: Performance on differences of Markov-chain graphs with fixed densities, p=30, n=200, δ=0.05

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.397	0.476	0.279	0.397	0.156
DIFFERENCE: TPR	0.397	0.476	0.270	0.397	0.159
EGD	0.145	0.145	0.140	0.145	0.147
PGD	0.145	0.145	0.145	0.145	0.145

Table A.93: Performance on differences of scale-free graphss with fixed densities, p=30, n=200, δ=0.05

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.424	0.492	0.333	0.439	0.167
DIFFERENCE: TPR	0.431	0.492	0.338	0.446	0.169
EGD	0.152	0.149	0.152	0.152	0.152
PGD	0.149	0.149	0.149	0.149	0.149

Table A.94: Performance on differences of random graphs ($\pi = 0.1$) with fixed densities, p=30, n=40, δ=0.20

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.508	0.518	0.492	0.508	0.437
DIFFERENCE: TPR	0.508	0.523	0.492	0.508	0.860
EGD	0.444	0.448	0.444	0.444	0.874
PGD	0.444	0.444	0.444	0.444	0.444

Table A.95: Performance on differences of Markov-chain graphs with fixed densities, p=30, n=40, δ=0.20

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.472	0.508	0.448	0.475	0.409
DIFFERENCE: TPR	0.472	0.511	0.455	0.478	0.393
EGD	0.409	0.411	0.416	0.411	0.393
PGD	0.409	0.409	0.409	0.409	0.409

Table A.96: Performance on differences of tree graphs with fixed densities, p=30, n=40, δ=0.20

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.204	0.236	0.218	0.208	0.208
DIFFERENCE: TPR	0.204	0.241	0.222	0.204	0.204
EGD	0.124	0.126	0.126	0.122	0.122
PGD	0.124	0.124	0.124	0.124	0.124

Table A.97: Performance on differences of scale-free graphs with fixed densities, p=30, n=40, δ=0.20

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.465	0.468	0.419	0.468	0.421
DIFFERENCE: TPR	0.462	0.474	0.421	0.468	0.421
EGD	0.391	0.398	0.395	0.393	0.393
PGD	0.393	0.393	0.393	0.393	0.393

Table A.98: Performance on differences of random graphs ($\pi = 0.1$) with fixed densities, p=30, n=20, δ=0.05

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.282	0.282	0.256	0.325	0.213
DIFFERENCE: TPR	0.282	0.282	0.256	0.321	0.167
EGD	0.179	0.179	0.179	0.177	0.140
PGD	0.179	0.179	0.179	0.179	0.179

Table A.99: Performance on differences of Markov-chain graphs with fixed densities, p=30, n=20, δ=0.05

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.234	0.222	0.206	0.254	0.158
DIFFERENCE: TPR	0.238	0.222	0.206	0.254	0.048
EGD	0.147	0.145	0.145	0.145	0.044
PGD	0.145	0.145	0.145	0.145	0.145

Table A.100: Performance on differences of tree graphs with fixed densities, p=30, n=20, δ=0.05

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.125	0.212	0.152	0.189	0.156
DIFFERENCE: TPR	0.121	0.212	0.152	0.212	0.212
EGD	0.074	0.076	0.076	0.085	0.103
PGD	0.076	0.076	0.076	0.076	0.076

Table A.101: Performance on differences of scale-free graphs with fixed densities, p=30, n=20, δ=0.05

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.234	0.238	0.177	0.226	0.138
DIFFERENCE: TPR	0.242	0.242	0.177	0.226	0.145
EGD	0.147	0.145	0.143	0.143	0.149
PGD	0.143	0.143	0.143	0.143	0.143

Table A.102: Performance on differences of random graphs ($\pi = 0.1$) with fixed densities, p=100, n=400, δ=0.05

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.432	0.485	0.320	0.439	0.197
DIFFERENCE: TPR	0.435	0.469	0.329	0.442	0.186
EGD	0.181	0.174	0.185	0.181	0.170
PGD	0.180	0.180	0.180	0.180	0.180

Table A.103: Performance on differences of Markov-chain graphs with fixed densities, p=100, n=400, δ=0.05

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.343	0.395	0.308	0.347	0.141
DIFFERENCE: TPR	0.350	0.404	0.279	0.341	0.137
EGD	0.113	0.114	0.100	0.109	0.107
PGD	0.111	0.111	0.111	0.111	0.111

Table A.104: Performance on differences of tree graphs with fixed densities, p=100, n=400, δ=0.05

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
MISMATCHING: PPV	0.214	0.251	0.185	0.210	0.073
MISMATCHING: TPR	0.210	0.256	0.188	0.210	0.062
EGD	0.035	0.036	0.036	0.036	0.030
PGD	0.036	0.036	0.036	0.036	0.036

Table A.105: Performance on differences of scale-free graphs with fixed densities, p=100, n=400, δ=0.05

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.411	0.469	0.312	0.412	0.177
DIFFERENCE: TPR	0.418	0.453	0.306	0.416	0.182
EGD	0.167	0.159	0.161	0.166	0.169
PGD	0.165	0.165	0.165	0.165	0.165

Table A.106: Performance on differences of random graphs ($\pi = 0.1$) with fixed densities, p=100, n=60, δ=0.20

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.458	0.455	0.442	0.460	0.422
DIFFERENCE: TPR	0.457	0.457	0.441	0.459	0.429
EGD	0.417	0.419	0.417	0.418	0.425
PGD	0.418	0.418	0.418	0.418	0.418

Table A.107: Performance on differences of Markov-chain graphs with fixed densities, p=100, n=60, δ=0.20

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.414	0.410	0.380	0.415	0.374
DIFFERENCE: TPR	0.414	0.407	0.383	0.413	0.377
EGD	0.370	0.368	0.374	0.369	0.374
PGD	0.371	0.371	0.371	0.371	0.371

Table A.108: Performance on differences of tree graphs with fixed densities, p=100, n=60, δ=0.20

	SEP-GL	R-JGL	FUSED JGL	GROUP JGL	DIRECT
DIFFERENCE: PPV	0.042	0.058	0.036	0.046	0.019
DIFFERENCE: TPR	0.041	0.057	0.036	0.046	0.015
EGD	0.038	0.039	0.039	0.040	0.033
PGD	0.039	0.039	0.039	0.039	0.039

Appendix B

Script

```
# graph_conc: makes graph from concentration matrix
# input #
# K: a concentration matrix
# dec: the decimals for rounding
# output #
# an undirected graph
graph_conc<- function(K, dec=4){
  return(graph_from_adjacency_matrix(round(K,dec)!=0, diag=F,
     ↪ mode="undir"))
}

# plot_conc: makes plot for graph associated to a
   ↪ concentration matrix
# input #
# K: a concentration matrix
# dec: the decimals for rounding
plot_conc<-function(K,dec=4){
  plot(graph_conc(K, dec=4))
}

# initialization: publicly initializes empty lists that are
   ↪ needed during the simulation
# input #
# diff: logical value, TRUE if the simulation includes a
   ↪ comparison of differences across methods
# set to default as FALSE since currently the simulation does
   ↪  not focus on comparison of differences
```

```
# output #
# no output. the objects are initialized publicly, when
   ↪ initialization() is run on the main script
initialization<- function(diff=FALSE){
  tot_sep_performance<<- list(list())
  tot_guo_performance<<- list(list())
  tot_jgl_fused_performance<<- list(list())
  tot_jgl_grouped_performance<<- list(list())
  tot_zhao_performance<<- list(list())

  sep_performance<<- list(list())
  guo_performance<<- list(list())
  jgl_fused_performance<<- list(list())
  jgl_grouped_performance<<- list(list())
  zhao_performance<<- list(list())

  mean_sep_performance<<- list(list())
  mean_guo_performance<<- list(list())
  mean_jgl_fused_performance<<- list(list())
  mean_jgl_grouped_performance<<- list(list())
  mean_zhao_performance<<- list(list())

  sd_sep_performance<<- list(list())
  sd_guo_performance<<- list(list())
  sd_jgl_fused_performance<<- list(list())
  sd_jgl_grouped_performance<<- list(list())
  sd_zhao_performance<<- list(list())

  if(diff){
    sep_diff_performance<<- list(list())
    guo_diff_performance<<- list(list())
    jgl_fused_diff_performance<<- list(list())
    jgl_grouped_diff_performance<<- list(list())

    tot_sep_diff_performance<<- list(list())
    tot_guo_diff_performance<<- list(list())
    tot_jgl_fused_diff_performance<<- list(list())
    tot_jgl_grouped_diff_performance<<- list(list())

    mean_sep_diff_performance<<- list(list())
    mean_guo_diff_performance<<- list(list())
```

```
    mean_jgl_fused_diff_performance<<- list(list())
    mean_jgl_grouped_diff_performance<<- list(list())

    sd_sep_diff_performance<<- list(list())
    sd_guo_diff_performance<<- list(list())
    sd_jgl_fused_diff_performance<<- list(list())
    sd_jgl_grouped_diff_performance<<- list(list())
  }

}

# tridiag_1: creates a tridiagonal matrix with 1s on the main
    ↪ , lower and upper diagonal
# input #
# p: dimension of the pxp matrix
tridiag_1<- function(p){
  mat<- matrix(0,p,p)
  diag(mat)<- 1
  indx <- seq.int(p-1)
  mat[cbind(indx+1,indx)]<- mat[cbind(indx,indx+1)]<- 1
  return(mat)
}

# generate_graph_structures: generates four (or less if
    ↪ specified) graph structures
# input #
# p: number of vertices
# pi: parameter for random graph sparsity. default to 1/p
# m: multiplier for the number of edges in the scale-free
    ↪ model
generate_graph_structures<- function(p, pi=1/p){
  random_graph<- sample_gnp(p,pi)
  chain_network<- graph_from_adjacency_matrix(tridiag_1(p),mode
     ↪ ="undir",diag=F)
  tree_network<- make_tree(p, children=2, mode="und")
  scale_free<- sample_pa(p,power=1.1,m=(p%/%30+sum(p%%30!=0)),
     ↪ directed=FALSE)

  return(list(ER=random_graph,CHAIN=chain_network,TREE=tree_
     ↪ network,BA=scale_free))
}
```

```
# get_zero_structure: recovers the zero structure of graphs
   ↪ as a list of zero edges
# input #
# graphs: a list of graphs
# output #
# zeros: a list of edgelists
get_zero_structure<- function(graphs,p){
  FULL<- make_full_graph(p)
  zeros<- lapply(graphs, function(x) get.edgelist(difference(
     ↪ FULL,x)))
  return(zeros)
}

# generate_Sigma: generates a covariance matrix Sigma
# input #
# p: dimension of covariance matrix
# mode: method for generating Sigma
# rho: parameter of equi-correlation
# set to default to a random number between -1/(p-1) and 1 (
   ↪ required for positive definiteness)
generate_Sigma<- function(p,mode=c("equicorr","beta","eigen","
   ↪ wishart"),rho=runif(1,-1/(p-1),1)){
  if(mode=="equicorr"){
    K<- matrix(rho,p,p)
    diag(K)<- 1
    Sigma<- solve(K)
  }
  else if(mode=="beta")
    Sigma<-genPositiveDefMat(p,covMethod="unifcorrmat",
       ↪ rangeVar= c(1.5,10))
  else if(mode=="eigen")
    Sigma<-genPositiveDefMat(p,covMethod="eigen", lambdaLow=2)
  else if(mode=="wishart")
    Sigma<- rWishart(1, df=500, diag(p))
  else stop("Invalid␣mode")
  return(Sigma)
}

# completed_matrices_H1: completes a covariance matrix using
   ↪ a given zero structure
```

```
# it uses the Lasso method with lambda = 0 (possible since
   ↪ the zero structure is given and Sigma is constructed to
   ↪  be positive-definite)
# input #
# Sigma: the covariance matrix
# zeros: the list of zero edgelists
# output #
# completion: a list of completed covariance matrices (W) and
   ↪  their inverse (K)
completed_matrices_H1<- function(Sigma,zeros){
  out<- lapply(zeros, function(x,Sigma1) glasso(Sigma1, 0, x)
     ↪ [1:2], Sigma1=Sigma)
  return(out)
}

# changed_edges: generates random edges IDs by sampling
   ↪ change_prop*p*(p-1)/2 elements as a vector.
# These will be the IDs on the upper/lower triangular
   ↪ adjacency matrix
# input #
# p: the dimension of the adjacency matrix
# change_prop: the probability of an edge changing
# H: the number of groups (i.e. networks)
# output #
# out: a list of H vectors of change IDs
changed_edges<- function(p, change_prop, H=2){
  no_changed_edges<- ceiling(p*(p-1)/2*change_prop)
  out<- list()
  for(h in 1:H)
    out[[h]]<- sample(1:(p*(p-1)/2),no_changed_edges,replace=F)
  return(out)
}

# group zeros: constructs edgelists for zero edges given the
   ↪ IDs for changed edges in each group
# input #
# graphs: a list of graphs (length 4)
# changed_edges: a list (length H) of ID vector for the lower
   ↪ -tri adjacency matrix elements to be changed
# output #
# zeros: the list (4xH) of new edgelists corresponding to
```

```
   ↪ zero elements in the adjacency matrices
group_zeros<- function(graphs,changed_edges){
  zeros<- list()
  group<- list()
  for(l in 1:length(graphs)){
    for(h in 1:length(changed_edges)){
      adj<- as.matrix(get.adjacency(graphs[[l]]))
      adj[lower.tri(adj)][changed_edges[[h]]]<- 1- adj[lower.
         ↪ tri(adj)][changed_edges[[h]]]
      adj[upper.tri(adj)]<-t(adj)[upper.tri(adj)]
      z<- allEdges(1-diag(p)-adj)
      group[[h]]<- z
    }
    zeros[[l]]<- group
  }
  names(zeros)<-names(graphs)
  return(zeros)
}

# completed_matrices_H: completes covariance matrices (for
   ↪ each group) using a given a list of lists of zero
   ↪ structures
# it uses the Lasso method with lambda = 0 (possible since
   ↪ the zero structure is given and Sigma is constructed to
   ↪  be positive-definite)
# input #
# Sigma: the covariance matrix
# zeros: the list of lists of zero edgelists
# output #
# completion: a list of lists of completed covariance
   ↪ matrices (W) and their inverse (K)
completed_matrices_H<- function(Sigma,zeros){
  out<- lapply(zeros, function(out_list, Sigma1) lapply(out_
     ↪ list, function(x,Sigma1) glasso(Sigma1, 0, x)$w,
     ↪ Sigma1=Sigma))
  return(out)
}

# make_mst_graph: finds the minimum spanning trees
# input #
# tree_zeros: a list containing the zero edgelists for the
```

```
   ↪ group-distincted tree graphs (as many as H)
# output #
# graph: a list of msts
make_mst_graph<- function(tree_zeros){
  FULL<- make_full_graph(p)
  graph<- lapply(tree_zeros, function(x) mst(difference(FULL,
     ↪ graph_from_edgelist(x,directed = F))))
  return(graph)
}

# make_mst_matrices: reconstructs the covariance and its
   ↪ inverse for all the tree graphs
# input #
# graph: a list of tree graphs
# Sigma: the common covariance matrix
# output #
# matrices: a list of cov/inv matrices after for the mst for
   ↪ each group (length H)
make_mst_matrices<- function(graph,Sigma){
  FULL<- make_full_graph(p)
  zeros<- lapply(graph, function(x) get.edgelist(difference(
     ↪ FULL,x)))
  matrices<- lapply(zeros, function(x, Sigma1) glasso(Sigma1,
     ↪ 0, x)$w,Sigma1=Sigma)
  return(matrices)
}

# AIC calculation
# df_graph: it calculates the degrees of freedom of a model
   ↪ based on its estimated inverse covariance matrix
# input #
# K: the estimated inv-cov matrix for the model
df_graph<- function(K){
  entries<- round(K[lower.tri(K)],4)
  return(sum(entries!=0))
}

# SL_AIC: calculates the AIC for a model, based on the sample
   ↪  cov and the estimated inv-cov
# input #
# estimated_k: the estimated inv-cov by the model
```

```
# sample_s: the sample cov matrix
# n: the number of observations
SL_AIC<- function(estimated_k,sample_s,n){
  return((-log(det(estimated_k))+sum(diag(sample_s%*%estimated
     ↪ _k)))*n+2*df_graph(estimated_k))
}

SL_BIC<- function(estimated_k,sample_s,n){
  return((-log(det(estimated_k))+sum(diag(sample_s%*%estimated
     ↪ _k)))*n+log(n)*df_graph(estimated_k))
}
# guo_glasso: estimates the covariance matrices and their
   ↪ inverse using method by guo (2011)
# input #
# S: a list of H sample covariance matrices (H=number of
   ↪ groups)
# rho: the penalization parameter, default = 0.05
# nu: the initial penalization on the diagonal, default= 0.05
# tol: threshold for determining convergence, default=1e-5
# it.max: maximum number of iterations allowed, default=1e+3
# output #
# w: a list of H estimated covariance matrices
# wi: a list of H estimated inverse covariance matrices
guo_glasso<- function(S, rho=0.05, nu=0.05, tol=1e-5, it.max=1
   ↪ e+3){
  p<- nrow(S[[1]])
  W<- lapply(S, function(x,nu1,p1) x+nu1*diag(p1), nu1=nu, p1=
     ↪ p)
  delta<- Inf
  K_old<- lapply(W, solve)
  sum_ab<- Reduce("+", lapply(K_old, abs)) # this does abs(K_
     ↪ old[[1]][i,j])+abs(K_old[[2]][i,j])
  Tau<- ifelse(sum_ab^(1/2)<10e-9,10e+9,sum_ab^(-1/2))
  it<- 0
  while(delta>tol & it<it.max){
    RHO<- rho*Tau
    update<- lapply(W, function(x,RHO1) glasso(x, rho=RHO1),
       ↪ RHO1=RHO)
    W<- lapply(update, function(x) x$w)
    K<- lapply(update, function(x) x$wi)
    delta<- Reduce("+",mapply(function(x,x_old) norm(x-x_old,"1
```

```
      ↪ "), K, K_old)) / Reduce("+", lapply(K_old, function(
      ↪ x_old) norm(x_old,"1")))
    K_old<- K
    sum_ab<- Reduce("+", lapply(K_old, abs))
    Tau<- ifelse(sum_ab^(1/2)<10e-9,10e+9,sum_ab^(-1/2))
    it<- it+1
  }
  return(list(w=W,wi=K))
}

# JOINT_AIC: calculates the joint AIC, as sum of individual
   ↪ AICs
# input #
# estimated_k: a list of H estimated inverse covariance
   ↪ matrices
# sample_s: a list of H sample covariance matrices
# n: the number of observation for the data (assumed equal
   ↪ across groups)
# output #
# sum(l): the sum of individual likelihoods
JOINT_AIC<- function(estimated_k,sample_s,n){
  l<- mapply(function(x,y, n1) (-log(det(x))+sum(diag(y%*%x)))
     ↪ *n1+2*df_graph(x), estimated_k, sample_s, MoreArgs =
     ↪ list(n1=n))
  return(sum(l))
}

JOINT_BIC<- function(estimated_k,sample_s,n){
  l<- mapply(function(x,y, n1) (-log(det(x))+sum(diag(y%*%x)))
     ↪ *n1+log(n)*df_graph(x), estimated_k, sample_s,
     ↪ MoreArgs = list(n1=n))
  return(sum(l))
}

# lower_tri_adjacency: it takes the inverse of a covariance
   ↪ matrix x as input and returns the lower triangle of the
   ↪  adjacency matrix
# the adjacency matrix is considered the matrix having 1
   ↪ where x!=0 and 0 where x=0
# input #
# x: the inv-cov matrix
```

```
# dec: the precision (decimal digit) for a value to be
   ↪ considered zero
# output #
# lo_tri: a vector of elements of the lower triangle of the
   ↪ adjacency matrix (does not include main diagonal)
lower_tri_adjacency<- function(x,dec=4){
  adj_matrix<- round(x,dec)!=0
  lo_tri<- adj_matrix[lower.tri(adj_matrix)]
  return(lo_tri)
}

# dens: calulates the density of a given graph
# input #
# K: inverse covariance matrix associated to the graph
# output #
# sum(lower_tri_adjacency(K))/d: a scalar, the density
   ↪ associated with the graph
# notice it's not the same as df_graph, this is relative
   ↪ frequency
dens<- function(K,dec=4){
  d<- ncol(K)*(ncol(K)-1)/2
  return(sum(lower_tri_adjacency(K,dec))/d)
}

# precision_recall: computes the precision (PPV), recall (TPR
   ↪ ) and sensitivity (TNR) for a couple of logical vectors
   ↪ , corresponding to population and estimated matrices
# a TRUE value in the input corresponds to a present edge, a
   ↪ FALSE value to a missing edge (after a chosen rounding)
# input #
# population: a (p(p-1)/2)x1 logical vector of population
   ↪ edge presence (according to some criterion)
# estimated: a (p(p-1)/2)x1 logical vector of estimated edge
   ↪ presence (according to some criterion)
# output #
# a list of precision, recall and specificity
# notice that when the denominator is zero, the value for the
   ↪  index is set to zero
precision_recall<- function(population, estimated){
  population_pos_id<- which(population==TRUE)
  population_neg_id<- which(population==FALSE)
```

```
  # precision
  TP<- sum(estimated[population_pos_id])
  FP<- sum(estimated[population_neg_id])
  precision<- TP/(TP+FP)
  if(is.nan(precision))
    precision<-0

  # recall
  FN<- sum(!estimated[population_pos_id])
  recall<- TP/(TP+FN)
  if(is.nan(recall))
    recall<-0

  return(list(precision,recall))
}

# joint_precision_recall: calculates the joint precision (PPV
   ↪ ), recall (TPR) and specificity (TNR) for a set of H
   ↪ estimated and population inverse covariance matrices
# input #
# true_adj: a list of H (p(p-1)/2)x1 logical vectors of
   ↪ population edge presence
# est_adj: a list of H (p(p-1)/2)x1 logical vectors of
   ↪ estimated edge presence
# ouput #
# list of either:
# - matching_presence (list of 3 indexes), matching_absence (
   ↪ list of 3 indexes), mismatching (list of 3 indexes)
# - matching (list of 3 indexes), mismatching (list of 2
   ↪ indxes)
# the former is for SL, GUO, JGL fused/group. the latter is
   ↪ for zhao
# NOTICE: the input must be in form of list. if coming from
   ↪ zhao, call the function with list(x),list(y)
joint_precision_recall<- function(true_adj,est_adj){

  if(length(true_adj)>1){
    # precision-recall for matching presence
    PP<- apply(as.data.frame(true_adj), 1, function(x) all(x))
    EP<- apply(as.data.frame(est_adj), 1, function(x) all(x))
```

```
    matching_presence<- precision_recall(PP,EP)

    # precision-recall for matching absence
    PA<- apply(as.data.frame(true_adj), 1, function(x) all(!x)
       ↪ )
    EA<- apply(as.data.frame(est_adj), 1, function(x) all(!x))
    matching_absence<- precision_recall(PA,EA)

    # precision-recall for mismatch
    PM<- apply(as.data.frame(true_adj), 1, function(x) sum(x)
       ↪ ==1)
    EM<- apply(as.data.frame(est_adj), 1, function(x) sum(x)
       ↪ ==1)
    matching_mismatching<- precision_recall(PM,EM)
  }

  else{
    matching<- precision_recall(!(unlist(true_adj)),!(unlist(
       ↪ est_adj)))
    mismatching<- precision_recall(unlist(true_adj),unlist(est_
       ↪ adj))
  }
  if(length(true_adj)>1)
    return(list(matching_presence,matching_absence,matching_
       ↪ mismatching))
  else
    return(list(matching,mismatching))
}

# simple_precision_recall: calculates precision (PPV), recall
   ↪  (TPR) and specificity (TNR) for graphs considered
   ↪ separately
# input #
# true_adj: a list of H (p(p-1)/2)x1 logical vectors of
   ↪ population edge presence
# est_adj: a list of H (p(p-1)/2)x1 logical vectors of
   ↪ estimated edge presence
# ouput #
# list of precision, recall, specificity (scalars)
# notice that when the denominator is zero, the value for the
   ↪  index is set to zero
```

```
simple_precision_recall<- function(true_adj,est_adj){
  # True Positives
  TP<- mapply(function(x,y) sum(x & y),true_adj,est_adj)
  # True Negatives
  TN<- mapply(function(x,y) sum(!(x) & !(y)),true_adj,est_adj)
  # False Positives
  FP<- mapply(function(x,y) sum(!(x) & y),true_adj,est_adj)
  # False Negatives
  FN<- mapply(function(x,y) sum(x & !(y)),true_adj,est_adj)

  precision<- sum(TP)/(sum(TP)+sum(FP))
  if(is.nan(precision))
    precision<-0
  recall<- sum(TP)/(sum(TP)+sum(FN))
  if(is.nan(recall))
    recall<-0

  return(list(precision,recall))
}

# performance: wrapper function to calculate joint and
   ↪ separate PPV,TPR,TNR and density
# input #
# TrueOmega: a list of H population inverse covariance
   ↪ matrices
# EstOmega: a list of H estimated inverse covariance matrices
# dec: the decimal position after which a small entry is to
   ↪ be rounded to zero
# output #
# out: a vector containing 6 or 9 indexes for Joint
   ↪ performance (zhao or not), 3 indexes for separate
   ↪ performance, 1 estimated average density, 1 population
   ↪ average density
performance<- function(TrueOmega,EstOmega,dec=4){
  true_adj<- lapply(TrueOmega,function(x,dec1) lower_tri_
     ↪ adjacency(x,dec1),dec1=dec)
  est_adj<- lapply(EstOmega,function(x,dec1) lower_tri_
     ↪ adjacency(x,dec1),dec1=dec)

  jpr<- unlist(joint_precision_recall(true_adj, est_adj))
  spr<- unlist(simple_precision_recall(true_adj, est_adj))
```

```
  EGD<- Reduce("+",lapply(EstOmega, function(x,dec1) dens(x,
     ↪ dec1),dec1=dec))/length(est_adj)
  # Population Non-zeros
  PGD<- Reduce("+",lapply(TrueOmega, function(x,dec1) dens(x,
     ↪ dec1),dec1=dec))/length(true_adj)

  out<- c(jpr,spr,EGD,PGD)
  return(out)
}

# AIC_sep_lasso: given a list of sample covariance matrices
   ↪ and setting an initial range of values for the
   ↪ regularization parameters, finds the optimal values for
   ↪  rho based on the AIC, by iteratively updating the
   ↪ range of values to choose from and fitting separate
   ↪ graphical lassos
# input #
# s_cov: a list of H sample covariance matrices
# n: the sample size
# len: the length of the range of values
# update_factor: a multiplicative/divisive factor to update
   ↪ the range extrema
# left: the initial range minimum
# right: the initial range maximum
# it.max: the maximum number of iterations to compute
# output #
# opt_rho: a list of H optimal values for the regularization
   ↪ parameters
AIC_sep_lasso<- function(s_cov, n, len=10, update_factor=10,
   ↪ left=5e-2, right=3, it.max=10){
opt_rho<- list()
H<- length(s_cov)
i<-1
delta<- Inf
I<- list(diag(p), diag(p))
lasso<- list()
aic_rho<- matrix(0,len,H)
old_rho_min_aic<- numeric(H)
dist<-numeric(H)
from<- rep(left,H)
```

```
to<- rep(right,H)

rho<- mapply(function(x,y,len1) list(seq(x, y, len=len1)), x=
   ↪ from, y=to, MoreArgs = list(len))
rhoI<- mapply(function(x,y) x*y, x=rho, y=I, SIMPLIFY = F)
cond<- mapply(function(x,y) ifelse(is.positive.definite(x+y),
   ↪ 1, 0), x=s_cov, y=rhoI)

while(sum(cond)!=H){ # if my starting interval has values too
   ↪ small
  from<- apply(from, function(x) x*2)
  rho<- mapply(function(x,y,len1) list(seq(x, y, len=len1)), x=
     ↪ from, y=to, MoreArgs = list(len))
  rhoI<- mapply(function(x,y) x*y, x=rho, y=I, SIMPLIFY = F)
  cond<- mapply(function(x,y) ifelse(is.positive.definite(x+y)
     ↪ , 1, 0), x=s_cov, y=rhoI)
}

  while(delta>1e-10 & i<it.max){
    for(r in 1:len){
      lasso<- mapply(function(x,y) glasso(x,rho=y[r])[1:2],x=s_
         ↪ cov,y=rho, SIMPLIFY = FALSE)
      aic_rho[r,]<- mapply(function(x,s,nn) SL_AIC(x$wi,s,nn),
         ↪ x=lasso,s=s_cov, MoreArgs=list(n))
    }
    rho_min_aic<- mapply(function(x,y) x[which.min(y)],rho, y=
       ↪ apply(aic_rho,2,as.list))
    for(h in 1:H){
      dist[h]<- min(rho_min_aic[h]-from[h], to[h]-rho_min_aic[h
         ↪ ])
      if(rho_min_aic[h]==from[h]){
        from[h]<- from[h]/update_factor
        to[h]<- to[h]/update_factor
      } else if(rho_min_aic[h]==to[h]){
        from[h]<- from[h]*update_factor
        to[h]<- to[h]*update_factor
      } else{
        from[h]<- rho_min_aic[h]-dist[h]
        to[h]<- rho_min_aic[h]+dist[h]
      }
      rho[[h]]<- seq(from[h], to[h], len=len)
```

```
    }

    rhoI<- mapply(function(x,y) x*y, x=rho, y=I, SIMPLIFY = F)
    cond<- mapply(function(x,y) ifelse(is.positive.definite(x+y
       ↪ ), 1, 0), x=s_cov, y=rhoI)

    while(sum(cond)!=H){ # if my starting interval has values
       ↪ too small
      from<- apply(from, function(x) x*2)
      rho<- mapply(function(x,y,len1) list(seq(x, y, len=len1))
         ↪ , x=from, y=to, MoreArgs = list(len))
      rhoI<- mapply(function(x,y) x*y, x=rho, y=I, SIMPLIFY = F
         ↪ )
      cond<- mapply(function(x,y) ifelse(is.positive.definite(x
         ↪ +y), 1, 0), x=s_cov, y=rhoI)
    }

    delta<- norm(as.matrix(rho_min_aic-old_rho_min_aic),"1")
    old_rho_min_aic<- rho_min_aic
    i<- i+1
  }
  opt_rho<- rho_min_aic
  return(opt_rho)
}

# AIC_sep_lasso: given a list of sample covariance matrices
   ↪ and setting an initial range of values for the
   ↪ regularization parameters, finds the optimal values for
   ↪  rho based on the AIC, by iteratively updating the
   ↪ range of values to choose from and fitting the
   ↪ reparametrization Joint Graphical Lasso
# input #
# s_cov: a list of H sample covariance matrices
# n: the sample size
# nu: the initial scalar added as S+nuI to ensure positive
   ↪ definiteness
# len: the length of the range of values
# update_factor: a multiplicative/divisive factor to update
   ↪ the range extrema
# left: the initial range minimum
# right: the initial range maximum
```

```
# it.max: the maximum number of iterations to compute
# output #
# opt_rho: a scalar of the optimal value for the
   ↪ regularization parameter
AIC_guo<- function(s_cov, n, nu=0.05, len=10, update_factor
   ↪ =10, left=5e-5, right=5e-3, it.max=10){
  from<- left
  to<- right
  rho<- seq(from, to, len=len)
  aic_rho<- numeric(len)
  old_rho_min_aic<- 0
  H<- length(s_cov)
  delta<-Inf
  it<-1
  I<- list(diag(p),diag(p))
  nuI<- mapply(function(x,y) x*y, x=list(nu,nu), y=I, SIMPLIFY
     ↪ = F)
  cond<- mapply(function(x,y) ifelse(is.positive.definite(x+y)
     ↪ , 1, 0), x=s_cov, y=nuI)

  while(sum(cond)!=H){ # if my starting interval has values
     ↪ too small
    nu<- apply(nu, function(x) x*2)
    nuI<- mapply(function(x,y) x*y, x=nu, y=I, SIMPLIFY = F)
    cond<- mapply(function(x,y) ifelse(is.positive.definite(x+y
       ↪ ), 1, 0), x=s_cov, y=nuI)
  }

    while(delta>1e-12 & it<it.max){
    for(r in 1:len){
      guo_lasso<- guo_glasso(s_cov,rho=rho[r],nu=nu)
      aic_rho[r]<- JOINT_AIC(guo_lasso$wi,s_cov,n)
    }
    rho_min_aic<- rho[which.min(aic_rho)]
    dist<- min(rho_min_aic-from, to-rho_min_aic)
    if(rho_min_aic==from){
      from<- from/update_factor
      to<- to/update_factor
    } else if(rho_min_aic==to){
      from<- from*update_factor
      to<- to*update_factor
```

```
    } else{
      from<- rho_min_aic-dist
      to<- rho_min_aic+dist
    }
    rho<- seq(from, to, len=len)
    delta<- abs(rho_min_aic-old_rho_min_aic)
    old_rho_min_aic<- rho_min_aic
    # print(list(aic_rho, rho))
    it<- it+1
  }
  opt_rho<- rho_min_aic
  return(list(rho=opt_rho, nu=nu))
}

# AIC_sep_lasso: given a list of sample covariance matrices
   ↪ and setting an initial range of values for the
   ↪ regularization parameters, finds the optimal values for
   ↪  rho based on the AIC, by iteratively updating the
   ↪ range of values to choose from and fitting the Fused or
   ↪  Group Joint Graphical Lasso
# input #
# data: a list of H nxp data matrices
# n: the sample size
# mode: the penalty for the JGL (accepted: "fused" or "group
   ↪ ")
# len: the length of the range of values
# update_factor: a multiplicative/divisive factor to update
   ↪ the range extrema
# left: the initial range minimum
# right: the initial range maximum
# it.max: the maximum number of iterations to compute
# output #
# opt_rho: a vector of the 2 optimal values for the
   ↪ regularization parameters rho1 and rho2

AIC_jgl_fixed<- function(data, n, mode=c("fused","group"),len
   ↪ =10, update_factor=10, left=5e-2, right=3, it.max=10){
opt_rho<- list()
H<- length(data)
it<-1
delta<- Inf
```

```
from<- rep(left,2)
to<- rep(right,2)
rho1<- seq(from[1], to[1], len=len)
rho2<- seq(from[2], to[2], len=len)
rho2_min_aic<- left
aic_rho<- numeric(len)
old_rho_min_aic<- 0
s_cov<- lapply(data, function(x,n1) cov(x)*(n1-1)/n1,n1=n)
# positive definiteness is given by step size

while(delta>1e-10 & it<it.max){
  for(r1 in 1:len){
      jgl<- JGL(data,penalty=mode,lambda1=rho1[r1],lambda2 =
         ↪ rho2_min_aic,return.whole.theta=TRUE)
      aic_rho[r1]<- JOINT_AIC(jgl$theta,s_cov,n)
  }
  rho1_min_aic<- rho1[which.min(aic_rho)]
  for(r2 in 1:len){
    jgl<- JGL(data,penalty=mode,lambda1=rho2[r2],lambda2 = rho1
       ↪ _min_aic,return.whole.theta=TRUE)
    aic_rho[r2]<- JOINT_AIC(jgl$theta,s_cov,n)
  }
  rho2_min_aic<- rho2[which.min(aic_rho)]
  rho_min_aic<- c(rho1_min_aic,rho2_min_aic)
  dist<- c(min(rho_min_aic[1]-from[1], to[1]- rho_min_aic[1]),
     ↪ min(rho_min_aic[2]-from[2], to[2]-rho_min_aic[2]))
  for(j in 1:2){
    if(dist[j]==0){
      if(rho_min_aic[j]==from[j]){
        from[j]<- from[j]/update_factor
        to[j]<- to[j]/update_factor
      } else {
        from[j]<- from[j]*update_factor
        to[j]<- to[j]*update_factor
      }
    } else{
      from[j]<- rho_min_aic[j]-dist[j]
      to[j]<- rho_min_aic[j]+dist[j]
    }
  }
  rho1<- seq(from[1],to[1],length.out = len)
```

```
  rho2<- seq(from[2],to[2],length.out = len)
  delta<- norm(as.matrix(rho_min_aic-old_rho_min_aic),"1")
  old_rho_min_aic<- rho_min_aic
  it<- it+1
}
opt_rho<- rho_min_aic
return(opt_rho)
}

AIC_jgl_grid<- function(data, n, mode=c("fused","group"),len
    ↪ =10, update_factor=10, left=5e-2, right=3, it.max=10){
  opt_rho<- list()
  H<- length(data)
  i<-1
  delta<- Inf
  from<- rep(left,2)
  to<- rep(right,2)
  rho1<- seq(from[1], to[1], len=len)
  rho2<- seq(from[2], to[2], len=len)
  aic_rho<- matrix(0,len,len)
  old_rho_min_aic<- numeric(2)
  s_cov<- lapply(data, function(x,n1) cov(x)*(n1-1)/n1,n1=n)
  # positive definiteness is given by step size

  while(delta>1e-10 & i<it.max){
    for(r1 in 1:len){
      for(r2 in 1:len){
        jgl<- JGL(data,penalty=mode,lambda1=rho1[r1],lambda2 =
            ↪ rho2[r2],return.whole.theta=TRUE)
        aic_rho[r1,r2]<- JOINT_AIC(jgl$theta,s_cov,n)
      }}
    ind<- arrayInd(which.min(aic_rho),c(len,len))
    rho_min_aic<- c(rho1[ind[1]], rho2[ind[2]])
    dist<- c(min(rho_min_aic[1]-from[1], to[1]- rho_min_aic[1])
       ↪ , min(rho_min_aic[2]-from[2], to[2]-rho_min_aic[2]))
    for(j in 1:2){
      if(dist[j]==0){
        if(rho_min_aic[j]==from[j]){
          from[j]<- from[j]/update_factor
          to[j]<- to[j]/update_factor
        } else {
```

```
        from[j]<- from[j]*update_factor
        to[j]<- to[j]*update_factor
      }
    } else{
      from[j]<- rho_min_aic[j]-dist[j]
      to[j]<- rho_min_aic[j]+dist[j]
    }
  }
  rho1<- seq(from[1],to[1],length.out = len)
  rho2<- seq(from[2],to[2],length.out = len)
  delta<- norm(as.matrix(rho_min_aic-old_rho_min_aic),"1")
  old_rho_min_aic<- rho_min_aic
  i<- i+1
 }
 opt_rho<- rho_min_aic
 return(opt_rho)
}

# rho_finder_sep_lasso: finds the regularization parameters
   ↪ that reproduce a given density and evaluates the
   ↪ corresponding separate graphical Lasso estimates
# input #
# sample_cov: a list of H sample covariance matrices
# true_dens: a list of H densities to reproduce
# left: a scalar determining the initial left extremum of the
   ↪  range from which to choose the parameters from
# right: a scalar determining the initial right extremum of
   ↪ the range from which to choose the parameters from
# s: the length of the range
# it.max: the maximum number of iterations after which to
   ↪ stop the program
# output #
# rho: a list of the H chosen regularization parameters
# model: a list of H concentration matrices, fit using opt_
   ↪ rho
rho_finder_sep_lasso<- function(sample_cov, true_dens, left,
   ↪ right, s, it.max){
 H<- length(sample_cov)
 rho<- list(as.list(seq(left,right,len=s)),as.list(seq(left,
    ↪ right,len=s)))
 fit<- mapply(function(x,r) lapply(r, function(xx,rr) glasso(
```

```
    ↪ xx, rho=rr)$wi, xx=x), sample_cov, rho, SIMPLIFY=FALSE
    ↪ )
  est_dens<- lapply(fit, function(x) lapply(x, dens))
  distances<- mapply(function(true,est) lapply(est, function(
    ↪ tt,ee) abs(tt-ee), tt=true), true_dens, est_dens,
    ↪ SIMPLIFY = FALSE)
  closest<- lapply(distances, which.min)

  it<- 1
  while(it < it.max){
  for(h in 1:H){
    if(distances[[h]][closest[[h]]]!=0){
      if(closest[[h]]==1)
        rho[[h]]<- as.list(seq(left*0.8, left*1.1, len=s))
      else if(closest[[h]]==s)
        rho[[h]]<- as.list(seq(right*0.9, right*1.2, len=s))
      else
        rho[[h]]<- as.list(seq(as.numeric(rho[[h]][closest[[h
          ↪ ]]])*0.9, as.numeric(rho[[h]][closest[[h]]])*1.1,
          ↪  len=s))
    } else next
  }
    fit<- mapply(function(x,r) lapply(r, function(xx,rr) glasso
      ↪ (xx, rho=rr)$wi, xx=x), sample_cov, rho, SIMPLIFY=
      ↪ FALSE)
    est_dens<- lapply(fit, function(x) lapply(x, dens))
    distances<- mapply(function(true,est) lapply(est, function(
      ↪ tt,ee) abs(tt-ee), tt=true), true_dens, est_dens,
      ↪ SIMPLIFY = FALSE)
    closest<- lapply(distances, which.min)
    it<- it+1
  }
  opt_rho<- mapply(function(x,r) r[x], closest, rho)
  opt_fit<- mapply(function(x,r) glasso(x,rho=r)$wi, sample_
    ↪ cov, opt_rho, SIMPLIFY = F)
  return(list(model=opt_fit, rho=opt_rho))
}

# revert_list: given a nested list, reverts the structure of
  ↪ the list. It does so by attributing names to the object
  ↪  of the sublist, using an initial letter and subsequent
```

```
   ↪ numbering. The letter can be chosen by the user
# input #
# x: a nested list
# letter: the letter to attribute to the sub-elements of the
   ↪ list, in order to revert its structure
# output #
# x with reverted nestedness and outer layer of list named
# example: if x is a list of s elements, each being a list of
   ↪ H elements, the output is a list of H, each being a
   ↪ list of s
revert_list<- function(x, letter="H"){
  s<- length(x)
  H<- length(x[[1]])
  hnames<- sapply(1:H, function(x) paste(letter,x))
  for(ss in 1:s) names(x[[ss]])<- hnames
  temp <- lapply(x, `[`, names(x[[1]])) ## Get sub-elements in
     ↪ same order
  return(apply(do.call(rbind, temp), 2, as.list))
}

# revert_list_with_names: reverts a list, when each element
   ↪ of the list already has a name
# input #
# x: a nested list
# output #
# x with reverted nestedness
revert_list_with_names<- function(x){
  xx <- lapply(x, `[`, names(x[[1]]))
  return(apply(do.call(rbind, xx), 2, as.list))
}

# rho_finder_guo: finds the regularization parameters that
   ↪ reproduce a given density and evaluates the
   ↪ corresponding reparametrization JGL estimates
# input #
# sample_cov: a list of H sample covariance matrices
# true_dens: a list of H densities to reproduce
# nu: the initial scalar to ensure positive definiteness
# left: a scalar determining the initial left extremum of the
   ↪ range from which to choose the parameters from
# right: a scalar determining the initial right extremum of
```

```
   ↪ the range from which to choose the parameters from
# s: the length of the range
# it.max: the maximum number of iterations after which to
   ↪ stop the program
# output #
# rho: a list of the H chosen regularization parameters
# model: a list of H concentration matrices, fit using opt_
   ↪ rho
rho_finder_guo<- function(sample_cov, true_dens, nu, left,
   ↪ right, s, it.max){
  rho<- as.list(seq(left,right,len=s))
  fit<- revert_list(lapply(rho, function(x, r, nn) guo_glasso(
     ↪ x, nu=nn, rho=r)$wi, x=sample_cov, nn=nu))
  est_dens<- lapply(fit, function(x) lapply(x, dens))
  distances<- mapply(function(true,est) lapply(est, function(
     ↪ tt,ee) abs(tt-ee), tt=true), true_dens, est_dens,
     ↪ SIMPLIFY = FALSE)
  closest<- lapply(distances, which.min)
  closest_one<- closest[[1]]
  it<- 1
  while(it < it.max){
      if(distances[[1]][closest_one]!=0){
        if(closest_one==1)
          rho<- as.list(seq(left*0.7, left*1.1, len=s))
        else if(closest_one==s)
          rho<- as.list(seq(right*0.9, right*1.3, len=s))
        else
          rho<- as.list(seq(as.numeric(rho[[closest_one]])*0.9,
             ↪  as.numeric(rho[[closest_one]])*1.1, len=s))
      }

    fit<- revert_list(lapply(rho, function(x, r, nn) guo_glasso
       ↪ (x, nu=nn, rho=r)$wi, x=sample_cov, nn=nu))
    est_dens<- lapply(fit, function(x) lapply(x, dens))
    distances<- mapply(function(true,est) lapply(est, function(
       ↪ tt,ee) abs(tt-ee), tt=true), true_dens, est_dens,
       ↪ SIMPLIFY = FALSE)
    closest<- lapply(distances, which.min)
    closest_one<- closest[[1]]
    it<- it+1
    }
```

```
  opt_rho<- rho[[closest_one]]
  opt_fit<- guo_glasso(sample_cov,rho=opt_rho,nu=nu)$wi
  return(list(model=opt_fit, rho=opt_rho))
}

# rho_finder_jgl: finds the regularization parameters that
   ↪ reproduce a given density and evaluates the
   ↪ corresponding fused/group JGL estimates
# input #
# data: a list of H nxp data matrices
# true_dens: a list of H densities to reproduce
# mode: the penalty for the JGL (accepted "fused" or "group")
# left: a scalar determining the initial left extremum of the
   ↪  range from which to choose the parameters from
# right: a scalar determining the initial right extremum of
   ↪ the range from which to choose the parameters from
# s: the length of the range
# it.max: the maximum number of iterations after which to
   ↪ stop the program
# output #
# rho: a list of the H chosen regularization parameters
# model: a list of H concentration matrices, fit using opt_
   ↪ rho
rho_finder_jgl<- function(data, true_dens, mode, left, right,s
   ↪ , it.max){
  rho<- revert_list(list(as.list(seq(left,right,len=s)),as.
     ↪ list(seq(left,right,len=s))), letter="s")
  fit<- revert_list(lapply(rho, function(x, r) JGL(x,penalty=
     ↪ mode, lambda1= r[[1]], lambda2= r[[2]], return.whole.
     ↪ theta = TRUE)$theta, x=data))
  est_dens<- lapply(fit, function(x) lapply(x, dens))
  distances<- mapply(function(true,est) lapply(est, function(
     ↪ tt,ee) abs(tt-ee), tt=true), true_dens, est_dens,
     ↪ SIMPLIFY = FALSE)
  closest<- lapply(distances, which.min)

  it<- 1
  while(it < it.max){
    rho<- revert_list(rho) # now it's a 2 x s list
    for(h in 1:2){
      if(distances[[h]][closest[[h]]]!=0){
```

```
        if(closest[[h]]==1)
          rho[[h]]<- as.list(seq(left*0.8, left*1.1, len=s))
        else if(closest[[h]]==s)
          rho[[h]]<- as.list(seq(right*0.9, right*1.2, len=s))
        else
          rho[[h]]<- as.list(seq(as.numeric(rho[[h]][closest[[h
             ↪ ]]])*0.9, as.numeric(rho[[h]][closest[[h]]])*
             ↪ 1.1, len=s))
      } else next
    }
    rho<- revert_list(rho, letter="s") # now it's an s x 2

    fit<- revert_list(lapply(rho, function(x, r) JGL(x, penalty
       ↪ =mode, lambda1= r[[1]], lambda2= r[[2]], return.whole
       ↪ .theta = TRUE)$theta, x=data))
    est_dens<- lapply(fit, function(x) lapply(x, dens))
    distances<- mapply(function(true,est) lapply(est, function(
       ↪ tt,ee) abs(tt-ee), tt=true), true_dens, est_dens,
       ↪ SIMPLIFY = FALSE)
    closest<- lapply(distances, which.min)
    it<- it+1
  }
  rho<- revert_list_with_names(rho)
  opt_rho<- mapply(function(id,r) as.numeric(r[id]),id=closest
     ↪ , r=rho)
  opt_fit<- JGL(data, penalty=mode, lambda1= opt_rho[1],
     ↪ lambda2= opt_rho[2], return.whole.theta = TRUE)$theta
  return(list(model=opt_fit, rho=opt_rho))
}

# rho_finder_diff: finds the regularization parameters that
   ↪ reproduce a given density for a graph of differences
   ↪ and evaluates the corresponding estimates
# input #
# data: a list of H nxp data matrices
# true_dens_diff: a scalar, the density to reproduce
# mode: the fitting method (accepted "separate", "guo", "zhao
   ↪ ")
# left: a scalar determining the initial left extremum of the
   ↪  range from which to choose the parameters from
# right: a scalar determining the initial right extremum of
```

```
   ↪ the range from which to choose the parameters from
# s: the length of the range
# nu: the initial scalar to ensure positive definiteness
# it.max: the maximum number of iterations after which to
   ↪ stop the program
# output #
# rho: a list of the H chosen regularization parameters (H =
   ↪ 1 if mode="zhao")
# model: a list of H concentration matrices, fit using opt_
   ↪ rho (H = 1 if mode="zhao")
rho_finder_diff<- function(data, true_dens_diff, mode, left,
   ↪ right, s, nu, it.max){
  n<- nrow(data[[1]])
  p<- ncol(data[[1]])
  rho<- as.list(seq(left,right,len=s))
  sample_cov<- lapply(data, function(x, nn) cov(x)*(nn-1)/nn,
     ↪ nn=n)

  if(mode=="separate"){
    fit<- lapply(rho, function(x, r) lapply(x, function(xx,rr)
       ↪  glasso(xx, rho=rr)$wi,rr=r), x=sample_cov)
    est_dens<- lapply(fit, function(x) dens(x[[1]]-x[[2]]))
  } else if(mode=="guo"){
    fit<- lapply(rho, function(xx,rr,nn) guo_glasso(xx, rho=rr,
       ↪  nu=nn)$wi,xx=sample_cov, nn=nu)
    est_dens<- lapply(fit, function(x) dens(x[[1]]-x[[2]]))
  } else if(mode=="zhao"){
    fit<- lapply(rho, function(x,r) dpm(x[[1]],x[[2]], lambda=r
       ↪ )$dpm, x=data)
    est_dens<- lapply(fit, function(x,pp) dens(matrix(unlist(x
       ↪ ), ncol=p, byrow=TRUE)), pp=p) # list of s
  }

  distances<- lapply(est_dens, function(tt,ee) abs(tt-ee), tt=
     ↪ true_dens_diff)
  closest<- which.min(unlist(distances))

  it<- 1
  while(it < it.max){
    if(distances[[closest]]!=0){
      if(closest==1)
```

```
      rho<- as.list(seq(left*0.8, left*1.1, len=s))
    else if(closest==s)
      rho<- as.list(seq(right*0.9, right*1.2, len=s))
    else
      rho<- as.list(seq(rho[[closest]]*0.9, rho[[closest]]*
        ↪ 1.1, len=s))
  }

  if(mode=="separate"){
    fit<- lapply(rho, function(x, r) lapply(x, function(xx,rr
      ↪ ) glasso(xx, rho=rr)$wi,rr=r), x=sample_cov)
    est_dens<- lapply(fit, function(x) dens(x[[1]]-x[[2]]))
  } else if(mode=="guo"){
    fit<- lapply(rho, function(xx,rr,nn) guo_glasso(xx, rho=
      ↪ rr, nu=nn)$wi,xx=sample_cov, nn=nu)
    est_dens<- lapply(fit, function(x) dens(x[[1]]-x[[2]]))
  } else if(mode=="zhao"){
    fit<- lapply(rho, function(x,r) dpm(x[[1]],x[[2]], lambda
      ↪ =r)$dpm, x=data)
    est_dens<- lapply(fit, function(x,pp) dens(matrix(unlist(
      ↪ x), ncol=p, byrow=TRUE)), pp=p) # list of s
  }

  distances<- lapply(est_dens, function(tt,ee) abs(tt-ee), tt
    ↪ =true_dens_diff)
  closest<- which.min(unlist(distances))
  it<- it+1
 }
 opt_rho<- as.numeric(rho[[closest]])
 opt_fit<- fit[[closest]]
 return(list(model=opt_fit, rho=opt_rho))
}

# rho_finder_JGL_diff: finds the regularization parameters
  ↪ that reproduce a given density for a graph of
  ↪ differences and evaluates the corresponding JGL
  ↪ estimates
# input #
# data: a list of H nxp data matrices
# true_dens_diff: a scalar, the density to reproduce
# mode: the penalty to be used (accepted "fused" and "group")
```

```
# left: a scalar determining the initial left extremum of the
   ↪  range from which to choose the parameters from
# right: a scalar determining the initial right extremum of
   ↪ the range from which to choose the parameters from
# s: the length of the range
# it.max: the maximum number of iterations after which to
   ↪ stop the program
# output #
# rho: a list of the H chosen regularization parameters
# model: a list of H concentration matrices, fit using rho
rho_finder_JGL_diff<- function(data, true_dens_diff, mode,
   ↪ left, right, s, it.max){
  rho1<- as.list(seq(left,right,len=s))
  rho2<- as.list(seq(left,right,len=s))
  rho2_fixed<- rho2[[as.integer(s/2)]]
  fit<- lapply(rho1, function(x,r,r2) JGL(x, penalty=mode,
     ↪ lambda1=r, lambda2=r2, return.whole.theta = TRUE)$
     ↪ theta, x=data, r2=rho2_fixed)
  est_dens<- lapply(fit, function(x) dens(x[[1]]-x[[2]]))
  distances1<- mapply(function(tt,ee) abs(tt-ee), tt=true_dens_
     ↪ diff, ee=est_dens)
  closest1<- which.min(distances1)
  rho1_fixed<- as.numeric(rho1[[closest1]])
  fit<- lapply(rho2, function(x,r,r1) JGL(x, penalty=mode,
     ↪ lambda1=r1, lambda2=r, return.whole.theta = TRUE)$
     ↪ theta, x=data, r1=rho1_fixed)
  est_dens<- lapply(fit, function(x) dens(x[[1]]-x[[2]]))
  distances2<- mapply(function(tt,ee) abs(tt-ee), tt=true_dens_
     ↪ diff, ee=est_dens)
  closest2<- which.min(distances2)
  rho2_fixed<- as.numeric(rho2[[closest2]])

  it<- 1
  while(it < it.max){
    if(distances1[closest1]!=0){
      if(closest1==1)
        rho1<- as.list(seq(as.numeric(rho1[[1]])*0.8,as.numeric
           ↪ (rho1[[1]])*1.1,len=s))
      else if(closest1==s)
        rho1<- as.list(seq(as.numeric(rho1[[s]])*0.9,as.numeric
           ↪ (rho1[[s]])*1.2,len=s))
```

```
    else
      rho1<- as.list(seq(as.numeric(rho1[[closest1]])*0.9,as.
         ↪ numeric(rho1[[closest1]])*1.1,len=s))
  }
  if(distances2[closest2]!=0){
    if(closest2==1)
      rho2<- as.list(seq(as.numeric(rho2[[1]])*0.8,as.numeric
         ↪ (rho2[[1]])*1.1,len=s))
    else if(closest2==s)
      rho2<- as.list(seq(as.numeric(rho2[[s]])*0.9,as.numeric
         ↪ (rho2[[s]])*1.2,len=s))
    else
      rho2<- as.list(seq(as.numeric(rho2[[closest2]])*0.9,as.
         ↪ numeric(rho2[[closest2]])*1.1,len=s))
  }

  rho2_fixed<- rho2[[as.integer(s/2)]]
  fit<- lapply(rho1, function(x,r,r2) JGL(x, penalty=mode,
     ↪ lambda1=r, lambda2=r2, return.whole.theta = TRUE)$
     ↪ theta, x=data, r2=rho2_fixed)
  est_dens<- lapply(fit, function(x) dens(x[[1]]-x[[2]]))
  distances1<- mapply(function(tt,ee) abs(tt-ee), tt=true_
     ↪ dens_diff, ee=est_dens)
  closest1<- which.min(distances1)
  rho1_fixed<- as.numeric(rho1[[closest1]])
  fit<- lapply(rho2, function(x,r,r1) JGL(x, penalty=mode,
     ↪ lambda1=r1, lambda2=r, return.whole.theta = TRUE)$
     ↪ theta, x=data, r1=rho1_fixed)
  est_dens<- lapply(fit, function(x) dens(x[[1]]-x[[2]]))
  distances2<- mapply(function(tt,ee) abs(tt-ee), tt=true_
     ↪ dens_diff, ee=est_dens)
  closest2<- which.min(distances2)
  rho2_fixed<- as.numeric(rho2[[closest2]])

  it<- it+1
 }
 opt_rho<- c(rho1_fixed,rho2_fixed)
 opt_fit<- JGL(data, penalty=mode, lambda1 = rho1_fixed,
    ↪ lambda2 = rho2_fixed, return.whole.theta = TRUE)$theta
 return(list(model=opt_fit, rho=opt_rho))
}
```