Benchmarks for spectralGraphTopology

Vinícius

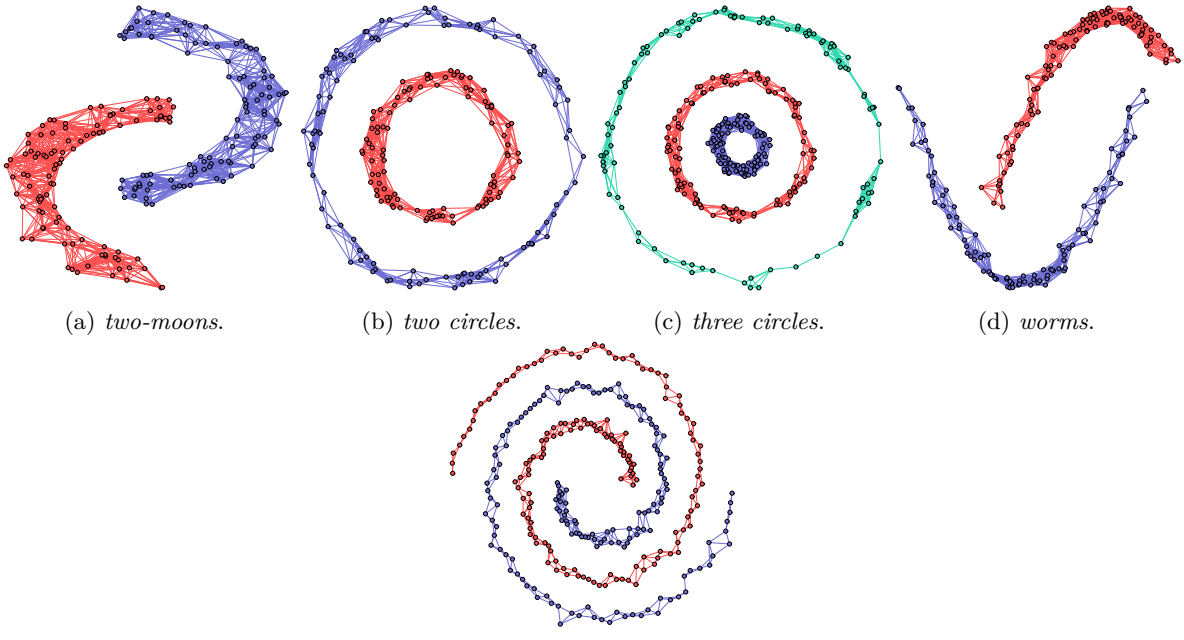
February 24, 2019

1 Clustering: warm-up

To warm-up, we consider the classical problem of clustering artificial nodes randomly distributed in $\mathbb{R}^2$ according to some geometry. More precisely, we use the package clusterSim to generate 100 nodes per cluster distributed according to structures colloquially known as *two-moons*, *two-circles*, *three-circles*, and *worms*.

Figure 1 depicts the results of learning the clusters structures using the algorithm *Spectral Graph Learning*, hereafter denoted as (SGL). As we can note, SGL is able to perfectly distinguish the cluster membership of all the nodes for all datasets. Additionally, Figure 2 depicts the convergence trend of the terms in the objective function for the *worms* dataset.

On what concers hyperparameter selection, we use $K = 3$ for the *three-circles* dataset, otherwise we use $K = 2$. We fix $\beta = 0.25$ and $\alpha = 0$ for all the datasets.



(a) *two-moons*. (b) *two circles*. (c) *three circles*. (d) *worms*.

(e) *spirals*.

Figure 1: Clustering results learned by the SGL algorithm for synthethic datasets.

2 Grid graphs

A comprehensive performance evaluation of the SGL algorithm was conducted considering grid graph models with 64 nodes denoted as $\mathcal{G}_{\text{grid}}^{(64)}$. We compare the performance of the SGL algorithm against state-of-the-art algorithms, namely CGL, CGL($\mathbf{A}$). We recollect that CGL($\mathbf{A}$) stands for the CGL algorithm equipped with

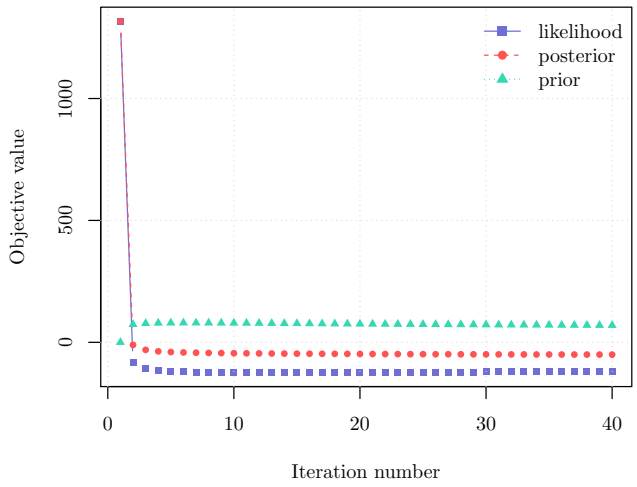


Figure 2: Convergence trend of the SGL algorithm for the *worms* dataset.

knowledge of the connectivity matrix $\mathbf{A}$, which gives the exact information of which nodes are connected to which ones.

The experimental setup is as follows. The edges of the graph model are sampled from $\mathsf{Uniform}(0.1, 3)$. The Laplacian matrix estimation is carried out on the basis of T samples distributed according to $\mathcal{N}(\mathbf{0}, \mathbf{L}^{\dagger}_{\mathsf{grid}})$. We repeat that experiment 20 times for every value of T and we average out the relative errors and F-scores[1].

Some hyperparameter tunning is required. For the SGL algorithm, we fix $\beta = 10$ for the values of T such that $T/N > 5$. Otherwise, we start with $\beta = 10^{-2}$, and we exponentially increase it up to $\beta = 4$. Additionally, we fix $\alpha = 0$.

For the CGL and CGL($\mathbf{A}$) algorithms, we follow the recipe by Eglimez, i.e., we choose α from $\{0\} \cup \left\{0.75^r \left(s_{\max}\sqrt{\log(N)/T}\right) | r = 1, 2, ..., 14\right\}$, such that the relative error between the estimated Laplacian and the ground truth is minimized, where $s_{\max}$ is the maximum value among the off-diagonal elements of the sample covariance matrix.

Figure 3 compares the performance of the algorithms for different sample size regimes for the grid graph model. As it can be noted, the SGL algorithm outperforms the CGL in both relative error and F-score senses. More precisely, for the case when the sample size is equal to the number of nodes, the difference in relative error and in F score are around 12% and 23%, respectively.

As expected, with the additional prior knowledge of the connectivity matrix $\mathbf{A}$, the CGL($\mathbf{A}$) algorithm basically attains a perfect F-score for $T/N \geq 10$. However, the connectivity matrix is not always available in practical problems, especially in clustering tasks where the goal is precisely to understand the connectivity membership among the nodes.

Nonetheless, the proposed SGL algorithm presents a comparable performance against CGL($\mathbf{A}$). For instance, at $T/N = 5$, the difference in relative error is only around 2.5%, and it keeps decreasing even further until virtually equal performance after $T/N = 100$, where the difference in relative error is around 1.2%. Additionally, we noted that the SGL algorithm requires far less tunning than CGL. At last, we include the relative error and the F-score for the Moore-Penrose inverse of the sample covariance matrix (ISCM) for completeness.

Figure 4 pictorially compares the graph structures learned by SGL and CGL when $T/N = 100$.

3 Erdos-Renyi graphs

The experiments involving Erdos-Renyi graph model is carried out in a similar fashion as for the grid graph. We consider an Erdo-Renyi graph with 64 nodes, denoted as $\mathcal{G}^{(64,p)}_{\mathsf{ER}}$, with $p = 0.1$, where p is the probability that a particular node is connected to any other node.

[1]For the computation of the F-score, we ignore edge weight values which are less than 10^{-1}.

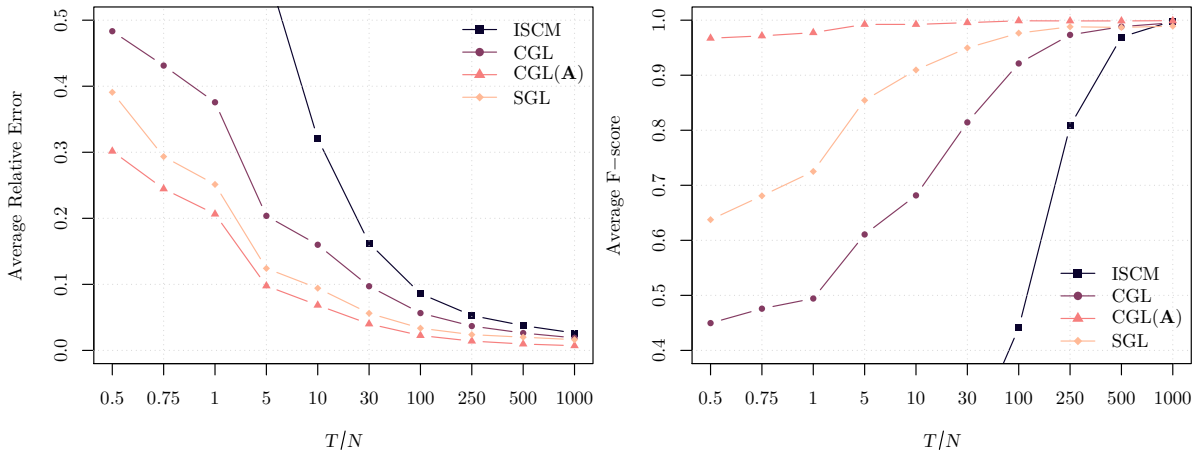
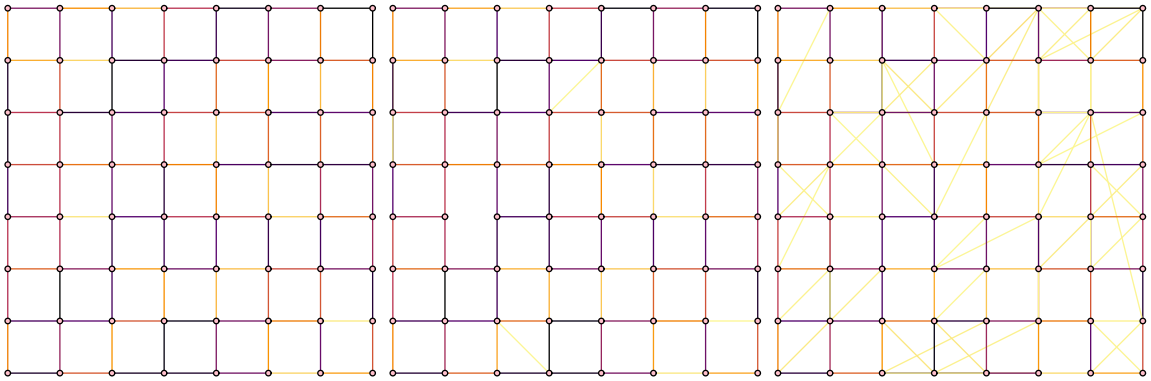


Figure 3: Average performance results for learning Laplacian matrix of a $\mathcal{G}_{\mathsf{grid}}^{(64)}$.



(a) Instance of a grid graph. (b) SGL $(\mathsf{RE}, \mathsf{FS}) = (0.045, 0.982)$. (c) CGL $(\mathsf{RE}, \mathsf{FS}) = (0.069, 0.812)$.

Figure 4: Sample results of learning $\mathcal{G}_{\mathsf{grid}}^{(64)}$ for $T/N = 100$. Edges smaller than 0.05 were removed. For SGL, we fix $\beta = 10$, whereas for CGL we perform a grid search in α to find α that maximizes the F-score. We found $\alpha = 3.6 \cdot 10^{-3}$.

On what concerns hyperparameter tunning, we fix $\alpha = 1.3 \cdot 10^{-2}$ and start with $\beta = 10^{-1}$ and we exponentially increase it up to $\beta = 1$ for the values of T such that $T/N \leq 1$. For $1 < T/N \leq 30$, we set $\alpha = 0$ and $\beta = 1.5$. Finally, for $T/N > 30$, we set $\alpha = 0$ and $\beta = 10$.

The hyperparameter tuning for the CGL and CGL($\mathbf{A}$) algorithms is done as described for the grid graph model.

Figure 5 reveals that the algorithms SGL and CGL obtain a similar performance across the sample size regimes. Unsurprisingly, the algorithm CGL($\mathbf{A}$) attains nearly perferct F-score for $T/N > 10$, because, again, it assumes that the connectivity information is fully available, which may be a non trivial assumption in practice.

4 Modular graphs

We consider a modular graph with 64 nodes and four modules, denoted as $\mathcal{G}_{\mathsf{M}}^{(64,p_1,p_2)}$, in which $p_1 = 0.01$, and $p_2 = 0.5$, where p_1 is the probability that a particular node of a given module is connected to any other node of a different module, and p_2 is the probability that a particular node is connected to any other node of the same module.

On what concerns hyperparameter tunning, we fix $\beta = 4$ for all values of T/N.

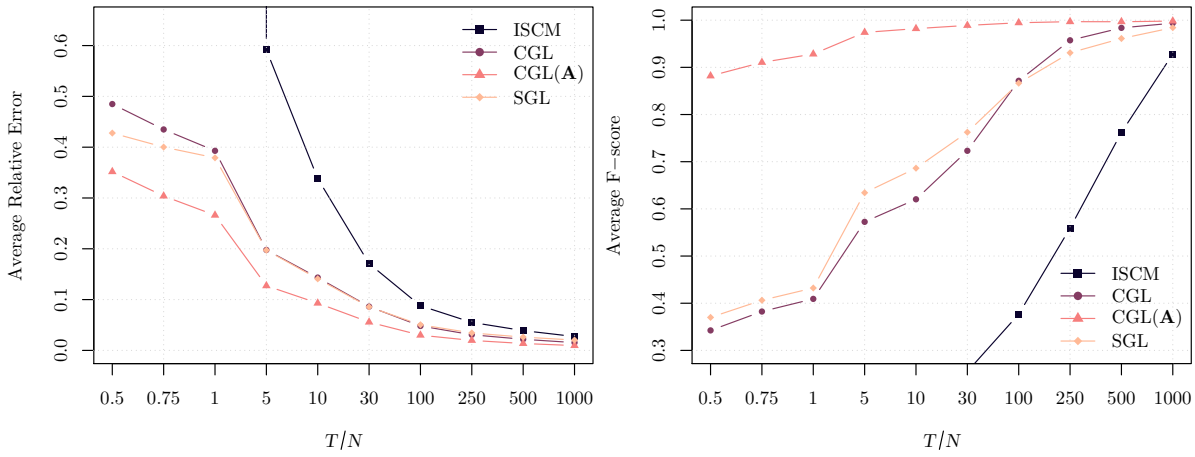


Figure 5: Average performance results for learning Laplacian matrix of a $\mathcal{G}_{\mathsf{ER}}^{(64,0.1)}$.

Figure 6 shows the performance of the proposed algorithms and the benchmarks. As we can note, the SGL algorithm outperforms the CGL algorithm in terms of F-score, while having a comparable performance in terms of relative error.

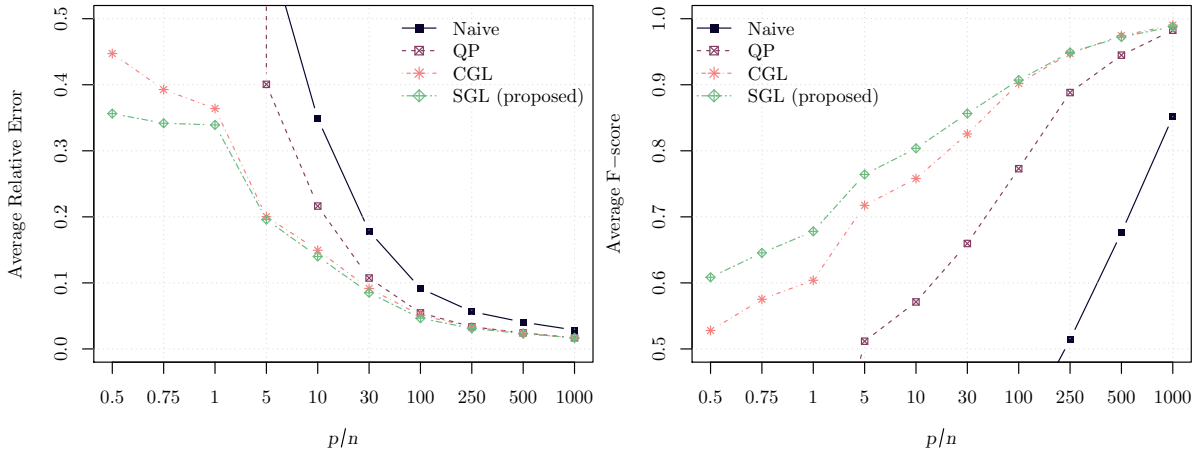
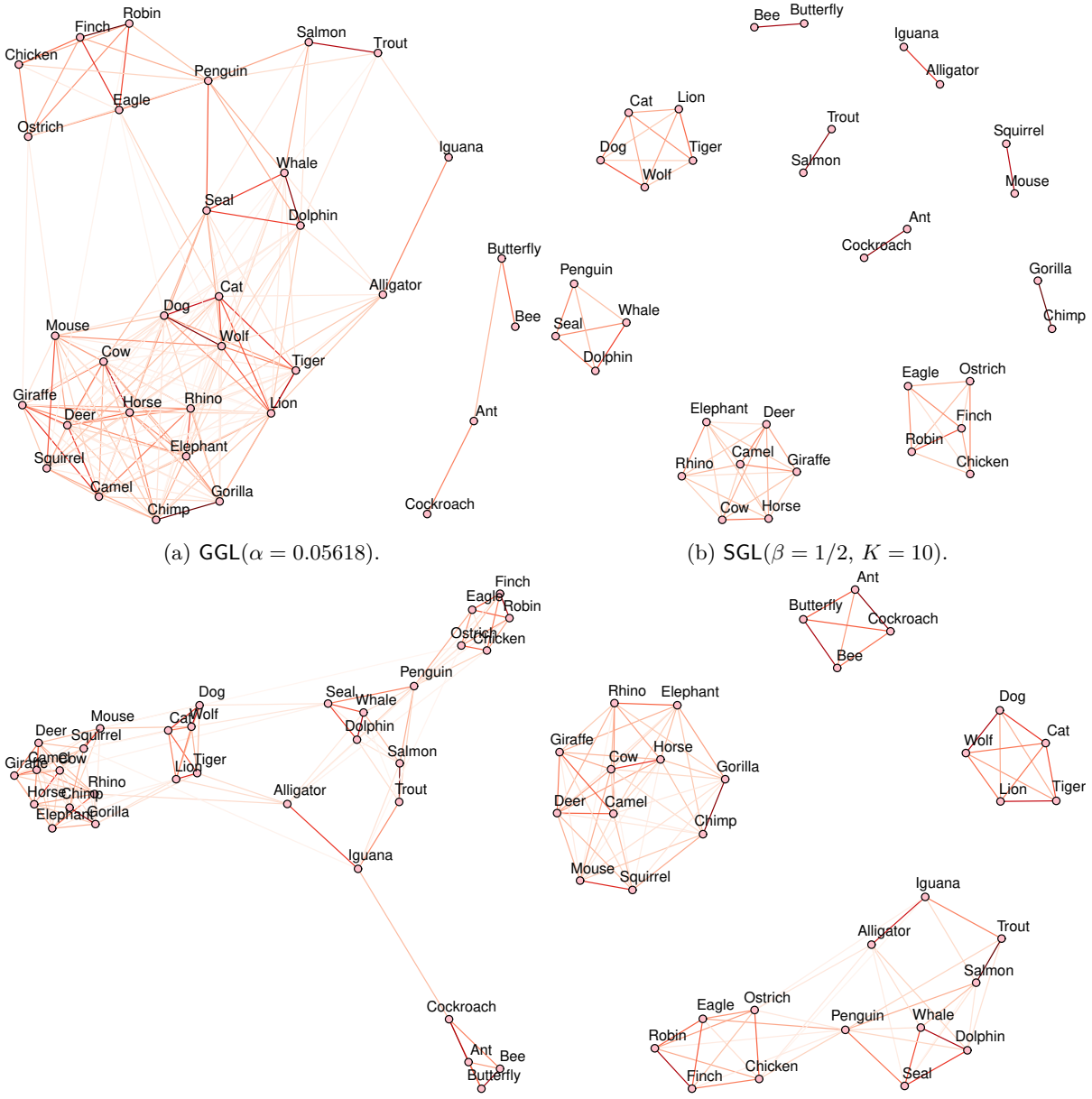


Figure 6: Average performance results for learning Laplacian matrix of a $\mathcal{G}_{\mathsf{M}}^{(64,0.01,0.5)}$ with four modules.

5 Animals dataset

The animals dataset consists of binary values which are the answers to questions such as "is warm-blooded?", "has lungs?", etc. There are in total 102 such questions, which make up the features, and 33 animal categories, which are the nodes. Figure 7 shows the results of estimating the graph Laplacian of the animals dataset using the GGL algorithm with $\alpha = 0.1$ and the proposed SGL algorithm with $\beta = 1/2$ and $\alpha = 10^{-1}$. The input for all the algorithms is the sample covariance matrix plus an identity matrix scaled by $1/3$ (see Elgimez paper for an explanation of the input).

The evaluation of the estimated graphs is done by natural intuition, i.e., we expect that similar animals such as (*ant*, *cockroach*), (*bee*, *butterfly*), and (*trout*, *salmon*), would be clustered together, while presently virtually zero connection to other (group of) animals. Under this sense, it can be seen that the SGL algorithm yields a more intutive graph than the ones learned by GGL and Glasso.



(a) GGL($\alpha = 0.05618$). (b) SGL($\beta = 1/2$, $K = 10$).

(c) SGL($\beta = 1/2$, $K = 1$). (d) SGL($\beta = 1/2$, $K = 4$).

Figure 7: Learning the connectivity of the animals dataset for different values of K.

6 Does a better initialization affect the final estimation?

The standard initialization for the SGL algorithm is to take $\mathbf{w}_0$ as the off-diagonal elements of the pseudoinverse of the sample covariance matrix, denoted as $\hat{\mathbf{\Theta}}_{\text{scm}}$. This is tantamount to solving the following optimization problem

$$\begin{array}{ll} \underset{\mathbf{w}}{\text{minimize}} & ||\hat{\mathbf{\Theta}}_{\text{scm}} - \mathcal{L}\mathbf{w}||^2_{\text{off},F} \\ \text{subject to} & \mathbf{w} \geq \mathbf{0}. \end{array}$$

In order to improve the initial estimate, we consider the following optimization problem

$$\begin{array}{ll} \underset{\mathbf{w}}{\text{minimize}} & ||\hat{\mathbf{\Theta}}_{\text{scm}} - \mathcal{L}\mathbf{w}||^2_{F} \\ \text{subject to} & \mathbf{w} \geq \mathbf{0}. \end{array} \tag{1}$$

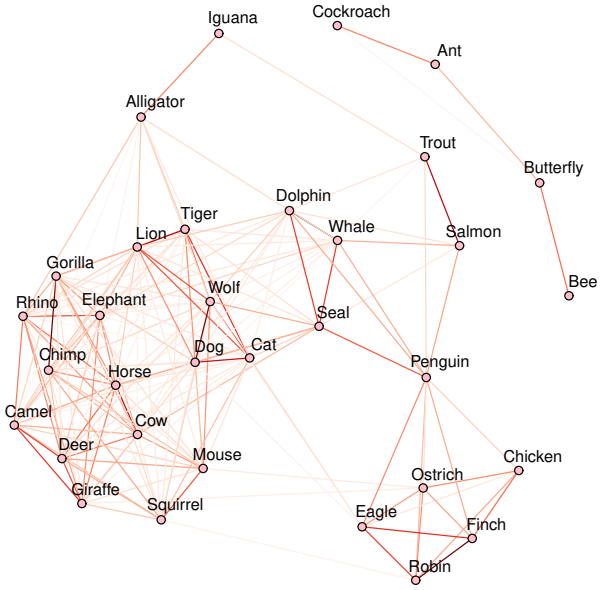


Figure 8: Learning the connectivity of the animals dataset with GLasso. Regularization parameter $\alpha = 0.05618$.

Note that,

$$||\hat{\mathbf{\Theta}}_{\mathsf{scm}} - \mathcal{L}\mathbf{w}||_F^2 = ||\mathrm{vec}\left(\hat{\mathbf{\Theta}}_{\mathsf{scm}}\right) - \mathrm{vec}(\mathcal{L}\mathbf{w})||_2^2 = ||\mathrm{vec}\left(\hat{\mathbf{\Theta}}_{\mathsf{scm}}\right) - \mathbf{R}\mathbf{w}||_2^2, \tag{2}$$

where $\mathbf{R} \triangleq \mathrm{vec} \circ \mathcal{L}$.

We denote the above algorithm as LLQP. Figure 9 shows the performance of the SGL algorithm when initialized by LLQP and ISCM. This experiment indicates that the SGL algorithm remains neutral to the different initilization procedures discussed here, even though it is clear that the LLQP shows a far better performance than ISCM.

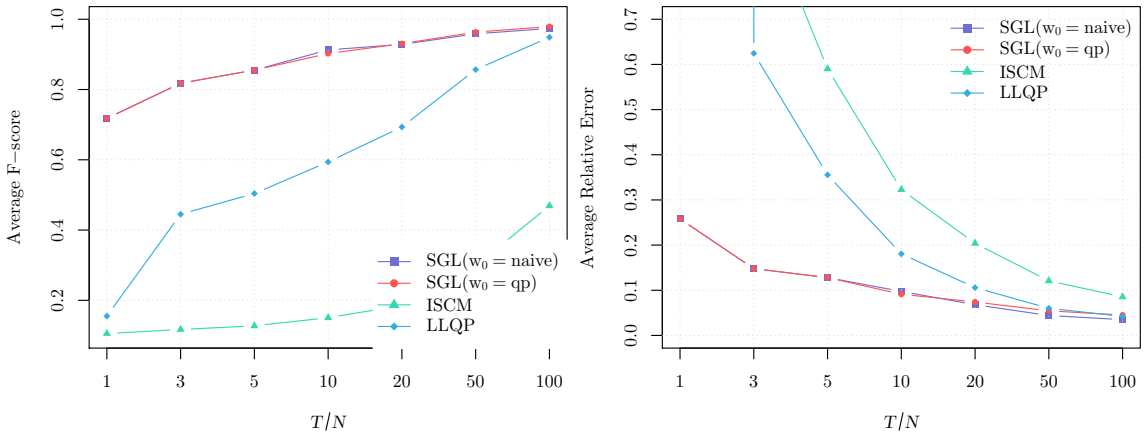


Figure 9: Relative error and F-score attained by SGL with different initialization strategies. SGL($\mathbf{w}_0 = \mathrm{naive}$) and SGL($\mathbf{w}_0 = \mathrm{qp}$) stand for the initialization with ISCM and LLQP, respectively.

7 4-component graph

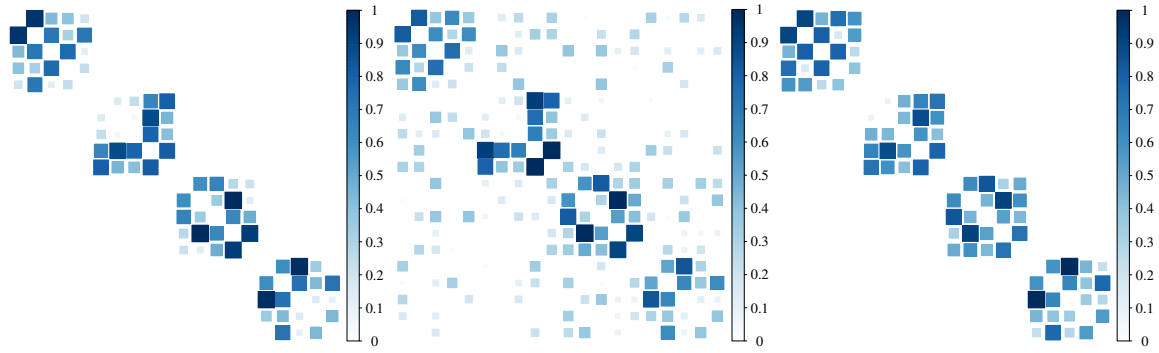
An experiment with synthetic data generated by a 4-component graph was conducted. We consider the following model:

$$\mathbf{L}_{\mathsf{noisy}} = \mathbf{L}_{\mathsf{true}} + \mathbf{L}_{\mathsf{ER}}, \tag{3}$$

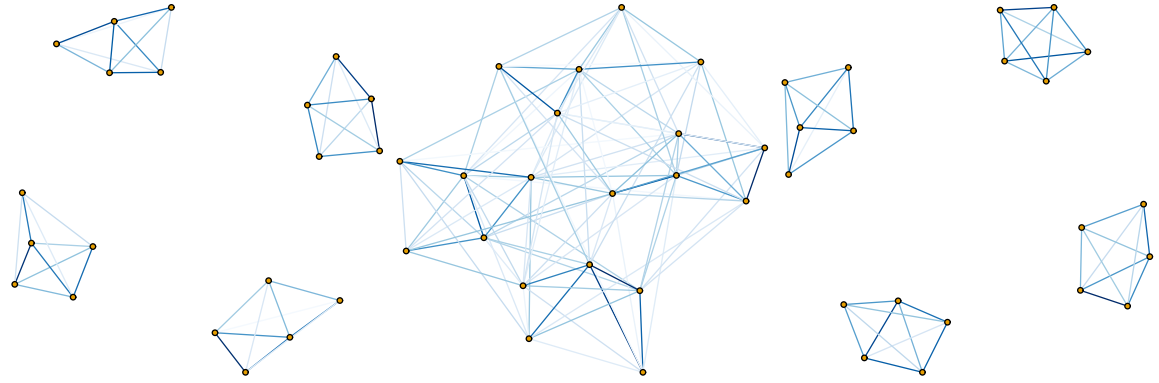
where $\mathbf{L}_{\mathsf{true}}$ represents the Laplacian matrix of a K-component graph (for this example, $K = 4$) denoted as $\mathcal{G}_K^{(p_1,p_2)}$, in which p_1 and p_2 represent the probabilities of node connections across components and within components, respectively; $\mathbf{L}_{\mathsf{ER}}$ represents the Laplacian of an Erdos-Renyi graph $\mathcal{G}_{\mathsf{ER}}^{(p)}$, in which p is the probability of a node connecting to any other node. The weighted edges of $\mathcal{G}_K^{(p_1,p_2)}$ were drawn from $\mathsf{Uniform}(0,1)$, whereas the ones of $\mathcal{G}_{\mathsf{ER}}^{(p)}$ where drawn from $\mathsf{Uniform}(0,\kappa)$, in which $\kappa \in (0,1)$ controls how much noise is added to the true graph. Finally, we set $p_1 = 0$, $p_2 = 1$, $p = 0.35$, and $\kappa = 0.45$.

Then, data were sampled in the form of $\mathbf{Y} \sim \mathcal{N}(\mathbf{0}, \mathbf{L}_{\mathsf{noisy}}^{\dagger})$, where $\mathbf{A}^{\dagger}$ denotes the generalized inverse of the matrix $\mathbf{A}$. The total number of nodes N and the number of drawn samples T were set to $N = 20$ (5 nodes per component) and $T/N = 30$.

Figure 10 illustrates the ground truth model, its noisy version, and the model learned by our spectral topology algorithm with $\beta = 20 \cdot N$ and $\alpha = 0.1$. We compute the performance of the learning process by means of the relative error (RE) and the F-score (FS). For this example, our algorithm achives $(\mathsf{RE}, \mathsf{FS}) = (0.210, 1)$ which means a perfect clustering accuracy even in a noisy model that heavily suppress the ground truth weights.



(a) Ground Truth Laplacian matrix (b) Noisy Laplacian matrix (c) Learned Laplacian matrix



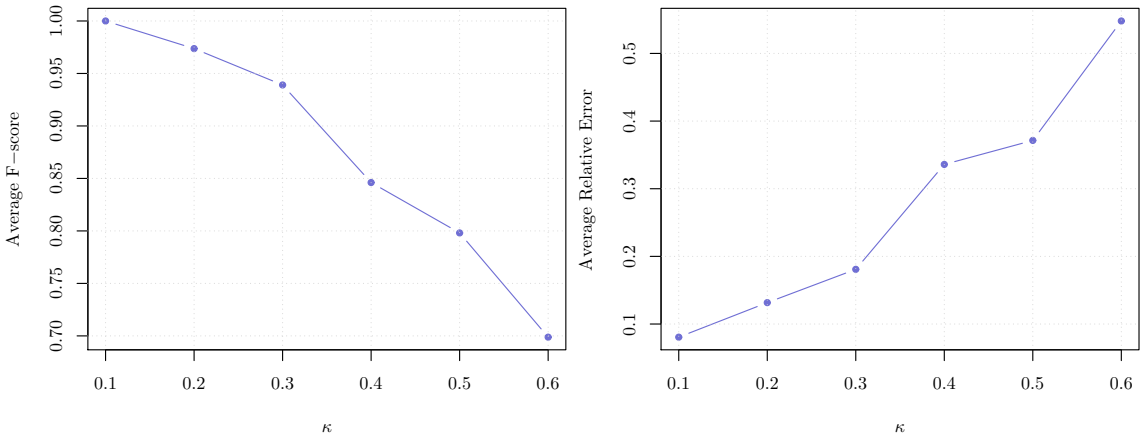
(d) Ground Truth graph (e) Noisy graph (f) Learned graph

Figure 10: An example of estimating a 4-component graph. (a) the ground truth graph Laplacian matrix ($\mathbf{L}_{\mathsf{true}}$), (b) $\mathbf{L}_{\mathsf{true}}$ after being corrupted by noise, (c) the learned graph Laplacian with a performance of $(\mathsf{RE}, \mathsf{FS}) = (0.210, 1)$. The panels (d), (e), and (f) correspond to the graphs represented by the Laplacian matrices in (a), (b), and (c), respectively.

Figure 12 shows the performance of our algorithm, using the same settings as the experiment above, but for different noise regimes.

8 Performance

To infer the performance as a function of the ratio T/N, we consider a 4-component graph in which every component has four nodes connected with probability 1. The edge weights among nodes are drawn from

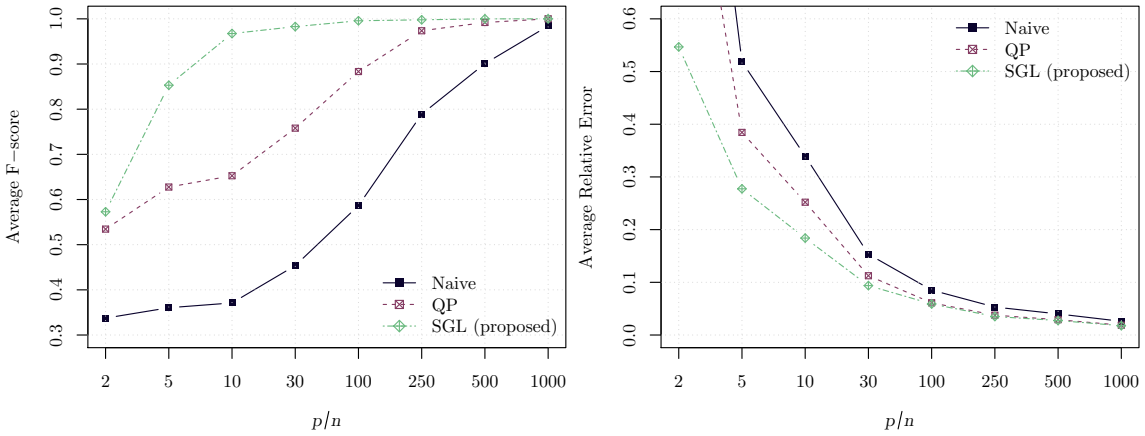


(a) Average F-score vs noise factor

(b) Relative error vs noise factor

Figure 11: Average performance results, as a function of the noise factor, for learning Laplacian matrix of a modular graph embedded in noise.

Uniform(0.1, 3). Additionally, we perform a grid search on the sparsity parameter α, just like the one performed by Elgimez.



(a) Average F-score vs T/N

(b) Average Relative Error vs T/N

Figure 12: Average performance results as, a function of the number of samples, for learning Laplacian matrix of a 4-component graph.

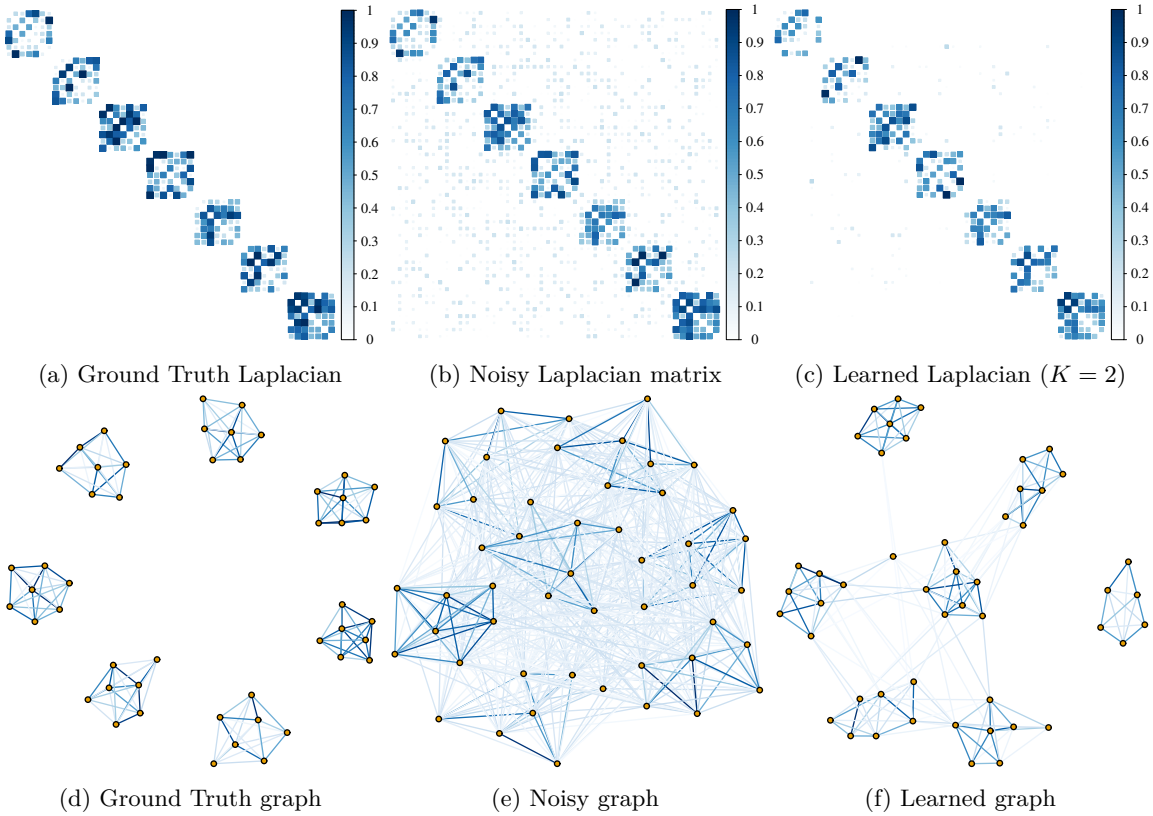
9 Model mismatch: k-component graph

In many applications, specially those in fully automated and unsupervised settings, the number of clusters is either unknown or it needs to be inferred from the data. In this section we consider an experiment involving model mismatch: the underlying Laplacian matrix that generates the data has K number of components but we actually use J, $J < K$, number of components to estimate it. Additionally, we consider that the generative model is noisy, i.e., $\mathbf{L}_{\mathsf{noisy}} = \mathbf{L}_{\mathsf{bm}} + \mathbf{L}_{\mathsf{ER}}$, where the parameters of the Erdos-Renyi model that acts as noise are $p = 0.25$ and $\kappa = 0.45$. The block-model $\mathbf{L}_{\mathsf{bm}}$ is constructed with seven blocks, each of which with seven nodes that are fully connected among themselves. The weights on the edges are drawn from $\mathsf{Uniform}(0, 1)$.

Figure 13 shows an example where the underlying graph has seven components, and we apply the SGL algorithm with $K = 2$. As we can see, even though the number of components is mismatched and the data is

noisy, the SGL algorithm is still able to indentify the true structure with a resonable performance in terms of F-score and average relative error.

Additionally, we investigate the performance as a function of the number of components, as depicted in Figure 14. As we can note, the performance is monotonically increasing and eventually reaches a perfect F-score when $K = 7$.



(a) Ground Truth Laplacian (b) Noisy Laplacian matrix (c) Learned Laplacian ($K = 2$)

(d) Ground Truth graph (e) Noisy graph (f) Learned graph

Figure 13: Example of model mismatching. (a) the ground truth graph Laplacian of a seven-component graph ($\mathbf{L}_{\mathsf{true}}$), (b) $\mathbf{L}_{\mathsf{true}}$ after being corrupted by noise, (c) the learned graph Laplacian with a performance of $(\mathsf{RE}, \mathsf{FS}) = (0.18, 0.81)$. The panels (d), (e), and (f) correspond to the graphs represented by the Laplacian matrices in (a), (b), and (c), respectively.

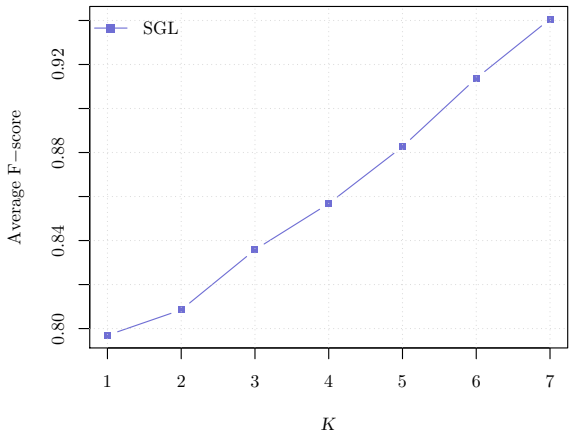


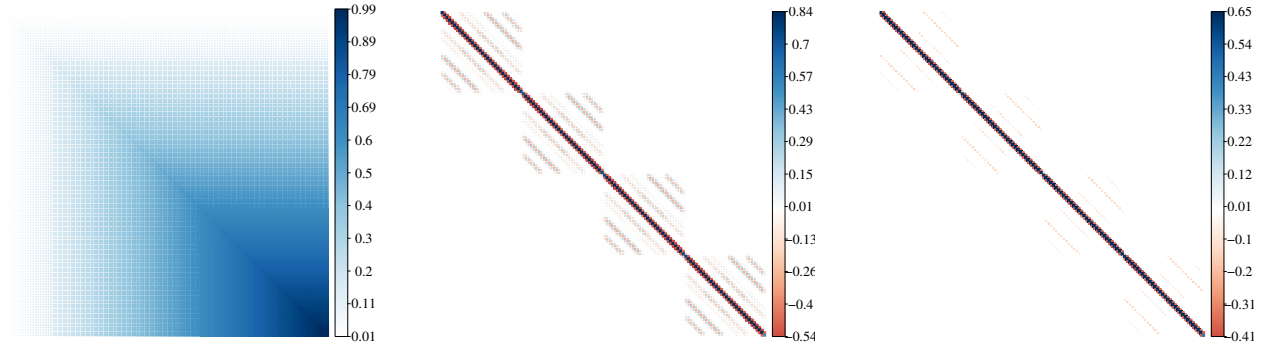
Figure 14: F-score as a function of the number of components K.

10 Model mismatch: Gaussian Process

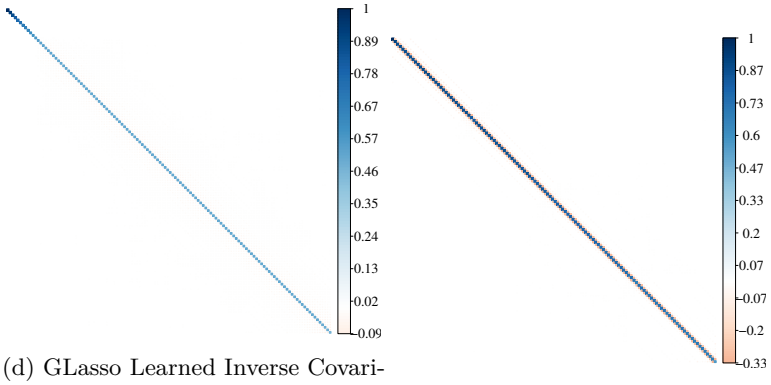
In case of model mismatch, i.e., when the observed data is not distributed according a Gaussian Markov Random Field (GMRF) with a Laplacian matrix as its precision matrix, then our method estimates the closest graph Laplacian fit with respect to the original model, as done similarly by GGL (see Egilmez *et. al.* 2017).

In this section, we show results from experiments in which the underlying covariance matrix is modeled as two-dimensional functions that often appear in the literature of Gaussian process regression.

We consider that the data $\boldsymbol{Y}$ is distributed according to a multivariate Gaussian with zero mean vector and covariance matrix $[\boldsymbol{\Sigma}]_{ij} = \sigma_1 \exp\left(-l_1 \sin^2\left(\frac{\pi|\boldsymbol{x}_i - \boldsymbol{x}_j|}{p}\right)\right) + \sigma_2 \exp\left(-l_2|\boldsymbol{x}_i - \boldsymbol{x}_j|^2\right)$.



(a) Ground Truth Covariance (b) Ground Truth Precision (c) SGL Learned Laplacian



(d) GLasso Learned Inverse Covariance (e) GGL Learned Laplacian

Figure 15: Learning the inverse covariance matrix of a Gaussian process.

11 Cancer RNA gene dataset

As an example of clustering on another real dataset, we consider the RNA-Seq PANCAN dataset available at the UC-Irvine Machine Learning Database[2].

This dataset is a subset of a larger dataset designed by the Cancer Genome Atlas Research Network[3]. It consists of genetic features which map twoelve types of cancer, namely: glioblastoma multiformae (GBM), lymphoblastic acute myeloid leukemia (LAML), head and neck squamous carcinoma (HNSC), lung adenocarcinoma (LUAD), lung squamous carcinoma (LUSC), breast carcinoma (BRCA), kidney renal clear-cell carcinoma (KIRC), ovarian carcinoma (OV), bladder carcinoma (BLCA), colon adenocarcinoma (COAD), uterine cervical and endometrial carcinoma (UCEC) and rectal adenocarcinoma (READ).

In the subset available at UCI, however, only five types of cancer are available. The dataset consists of 801 labeled samples, in which every sample has 20531 genetic features. The goal in this dataset is to classify and cluster the samples according to their tumor type on the basis of those genetic features.

[2] `https://archive.ics.uci.edu/ml/datasets/gene+expression+cancer+RNA-Seq`
[3] https://www.nature.com/articles/ng.2764

In our experiments, we first apply the SGL algorithm to the first 70 nodes using $K = 5$ and $\beta = 5$. Figure 18 shows a perfect clustering, which indicates that SGL may be used to automatically predict types of cancer tumors based on gene features.

We also compare the classification performance against the GGL algorith. As depicted in Figure 17, the GGL algorithm correctly classifies most of the nodes, but fails the segregrate the clusters as it is evidenced by the vast number of mistaken edges placed accross clusters.

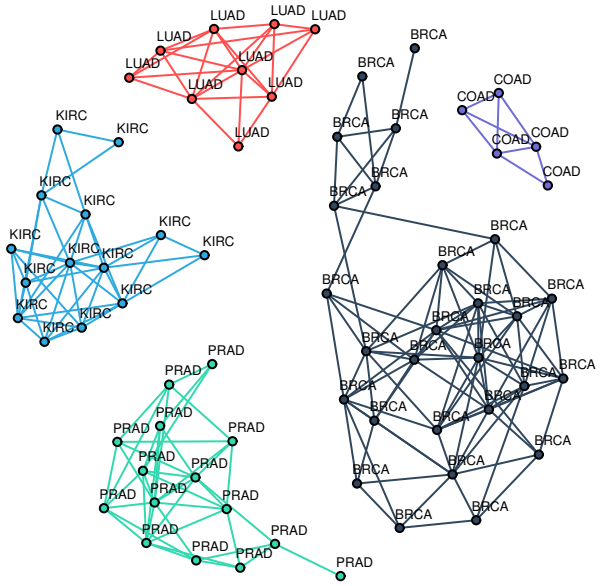


Figure 16: Clustering the first 70 nodes of the PANCAN dataset with SGL($K = 5, \beta = 5$).

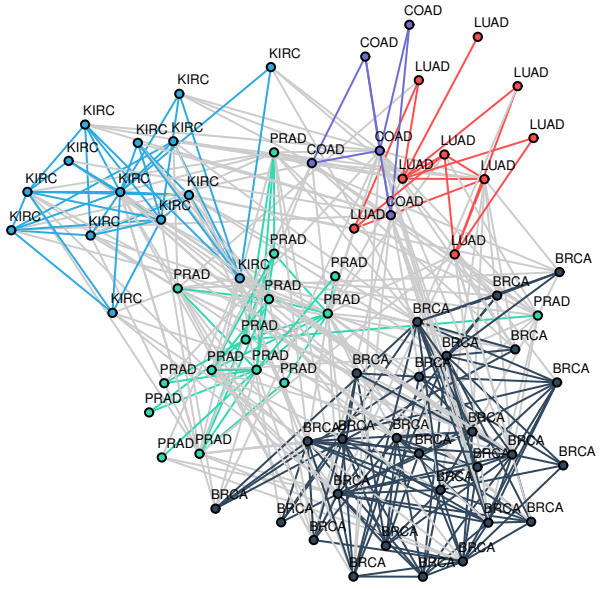


Figure 17: Clustering the first 70 nodes of thePANCAN dataset with GGL($\alpha = 0.1$). Grey colored edges indicate wrong connections.

Finally, we perform clustering for the full dataset. The result is depicted in Figure **??**.

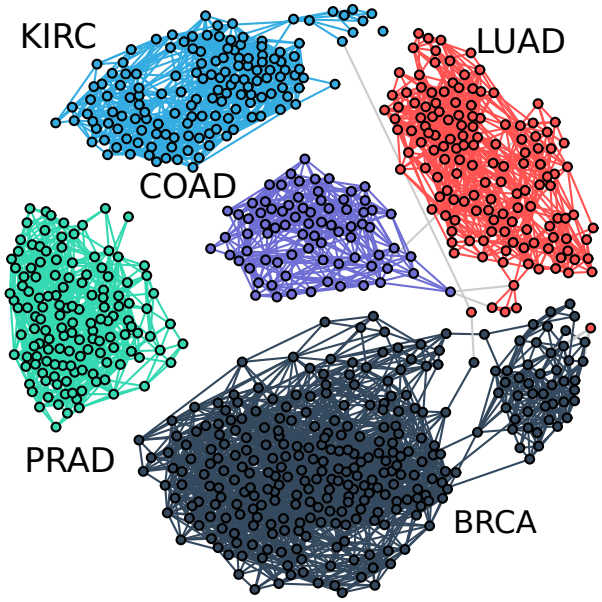


Figure 18: Clustering the full 801 nodes of the PANCAN dataset with SGL($K = 5, \beta = 10$).