

An introduction to network inference and mining

Résumé

This vignette needs no specific prerequisites : it deals with elementary concepts related to graphs or networks. It has been created more specifically for application in biology but can be used by anybody interested by this topic, though the second part, about network inference, is more specific to biostatistics. Also, this vignette is written in English but the most important words are translated in French and formatted this way : *en français* !

Table des matières

1 Basic definitions and concepts for graphs / networks	1
1.1 Examples	2
1.2 Overview of standard issues for network analysis	3
1.3 More complex relational models	3
2 Network inference	4
2.1 Basics about network inference	4
2.2 Relevance network	4
2.3 Gaussian Graphical Model	5
2.3.1 Estimating partial correlations in GGM	6
2.3.2 Estimating the conditional dependency graph with Graphical LASSO	6
2.4 Other references	7
3 Network mining	7
3.1 Basic definitions	7

3.2 Network visualization	8
3.3 Global characteristics	8
3.4 Individual characteristics	9
3.5 Clustering <i>Classification</i>	12
3.6 A few more references on clustering	12

4 Applications **13**

4.1 Gene network inference	13
4.1.1 Presentation of the dataset	13
4.1.2 Network inference	16
4.1.3 Building the network	17
4.1.4 Short mining of the network	18
4.2 Facebook network mining	19
4.2.1 Visualization	21
4.2.2 Individual characteristics	23
4.2.3 Clustering	24
4.2.4 Export graph	25

1 Basic definitions and concepts for graphs / networks

A *network* (*un réseau*), also frequently called a *graph* (*un graphe*), is a mathematical object used to model *relations* between *entities*. In its simplest form, it is composed of two sets (V, E) :

- the set $V = \{v_1, \dots, v_p\}$ is a set of p *vertices* (vertexes in American English, *sommets* in French), also called *nodes* (*nœuds*) that represent the entities ;
- the set E is a subset of the set of vertex pairs, $E \subset \{(v_i, v_j), i, j = 1, \dots, p, i \neq j\}$: the vertex pairs in E are called *edges* (*arêtes*) of the graph and model a given type of relations between two entities.

A network is often displayed as in Figure 1 : the vertices are represented with circles and the edges with straight lines connecting two vertices.

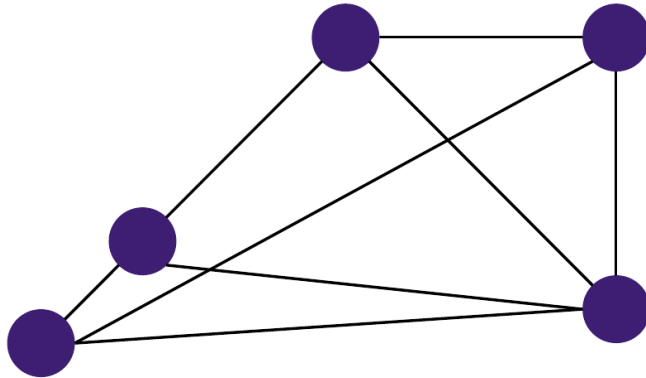


FIGURE 1 – Simple example of a network with 5 vertices (purple circles) and 8 edges (lines connected two vertices).

1.1 Examples

Hereafter, several examples of networks are given, that will be used for illustration purpose in the remaining of this vignette :

my own facebook network¹ (on August, 23rd, 2013) : the nodes of the network are my facebook “friends” (mostly my former students ; only initials are given for confidentiality reasons) and an edge between two nodes represents a facebook relationship between the two corresponding persons. The data as well as the scripts used to study this network are provided and commented in Section 4.2. This network is displayed in Figure 2.

a medieval network built from a large corpus of notarial acts (mostly “*baux à fief*” and more precisely, land charters) established in a former *seigneurie* called “Castelnau-Montratier”, located in the Lot (South

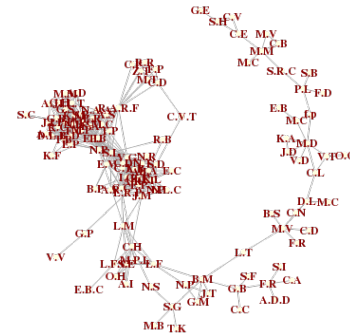


FIGURE 2 – My facebook network on August, 23rd, 2013. See Section 4.2 for further details.

West of France), not far from the nowadays village with the same name. All the documents were established between 1250 and 1500 and were used to model a large network having for vertices, the individuals and the transactions found in the corpus (3 918 nodes). An edge between an individual and a transaction indicates that the individual has been directly involved in the transaction (6 455 edges). This network contains no edge between two individuals or between two transactions. This network will be used to illustrate some notions on a large network in the following sections. It is displayed in Figure 3 and more details on this corpus and the data used to build this network can be found in [Boulet et al., 2008, Rossi and Villa-Vialaneix, 2011] ;

a network inferred from gene expression data The `nutrimouse` dataset, provided in the R package `mixOmics` contains the expression of 120 genes potentially involved in nutritional problems and the concentrations of 21 hepatic fatty acids for forty mice. The data set comes from a nutrigenomic study in the mouse [Martin et al., 2007] in which the effects of two genotypes and five diets with contrasted fatty acid compositions on

1. <http://www.facebook.com>

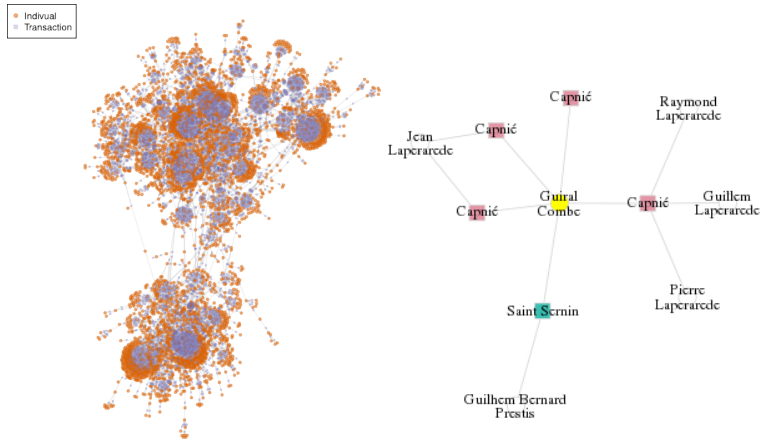


FIGURE 3 – A network built from a large corpus of notarial acts : full representation (left) and zoom on few number of nodes (right) where transactions are represented by squares and individuals by circles.

liver lipids and hepatic gene expression in mice were considered. Network inference and mining is performed in Section 4.1

1.2 Overview of standard issues for network analysis

This vignette will address two main issues posed by network analysis :

- the first one will be discussed in Section 2 and is called *network inference* (*inférence de réseau*) : giving data (i.e., variables observed for several subjects or objects), how to build a network whose edges represent the “direct links” between the variables ? A famous example of this kind of problem is network inference designed to recover the regulation network from gene expression data : in this problem, gene expression are measured for several subjects (e.g., by microarray or by RNAseq techniques). The vertices in the inferred network are the genes and the edges represent a strong “direct link” between the two gene expressions ;
- the second issue comes when the network is already built or directly given : the practitioner then wants to understand the network’s main characteristics and extract its main nodes, groups, etc. This ensemble

of methods, studied in Section 3, can be called under the generic name of *network mining* (*fouille de réseau*) and comprises (among other problems) :

- *network visualization* : when representing a network, no *a priori* position is associated to its vertices and the network can thus be displayed in many different ways, as illustrated in Figure 4 where my facebook network is displayed in two different ways : with a random layout and with a layout produced by an algorithm designed for positioning connected nodes close to each other ;



FIGURE 4 – My facebook network displayed with a random layout (left) and with a layout produced by a force-directed placement algorithm [Fruchterman and Reingold, 1991] (right).

- *vertex clustering* : a intuitive way to understand a network structure is to focus not on individual connections between nodes but on connections between densely connected groups of nodes. These groups are often called *clusters* or *communities* (*communautés*) or *modules* (*modules*) and many work in the literature have focused on the problem of extracting these clusters.

1.3 More complex relational models

This lesson’s scope is restricted to simple networks, i.e., undirected networks with no loop (i.e., no edge between a given node and itself) and simple edges. But graphs (and relational models) can deal with many other types of real-life situations :

directed graphs (*graphe orienté*) in which the edges have a direction, i.e.,

the edge from the vertex v_i to the vertex v_j is not the same than the edge from the vertex v_j to the vertex v_i . In this case, the edges are often called *arcs* (*arcs*);

weighted graphs (*graphe pondéré*) in which a (often positive) weight is associated to each edge;

graphs with multiple edges where a pair of vertices can be linked by several edges, that can eventually have different labels or weights to model different types of relationships;

labeled graphs (*graphe étiqueté*) where one or several labels are associated to each node, labels can be factors (e.g., gender) or numeric values (e.g., age).

2 Network inference

The purpose of this section is to explain how to estimate a regulatory network from gene expression data. This issue is called *network inference*. We will restrict ourselves to the very simple case where a large set of genes have been measured on a few number of individuals. The data set is represented by the matrix \mathbf{X} :

$$\begin{array}{l}
 \text{individuals} \\
 n \simeq 30/50
 \end{array}
 \left\{ \mathbf{X} = \begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & X_i^j & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix} \right.$$

variables (genes expression), $p \simeq 10^{3/4}$

Even restricting to a small subset of genes, having $n < p$ is the standard situation.

From these data, we want to build a network where :

- the nodes represent the p genes;
- the edges represent a “direct” and “significant” co-expression between two genes. This kind of relations aims at tracking transcription relations.

The main advantage of using networks over raw data is that such a model focuses on “significant” links and is thus more robust. Also, inference can be combined/compared with/to bibliographic networks to incorporate prior knowledge into the model but, unlike bibliographic networks, networks inferred

from one of the model presented below can handle even unknown (i.e., not annotated) genes into the analysis.

2.1 Basics about network inference

Even if alternative approaches exist, a common way to infer a network from gene expression data is to use the steps described in Figure 5 :

1. at first, the user calculates pairwise similarities (e.g., correlations in the simplest case) between pairs of genes;
2. second, the smallest (or less significant) similarities are thresholded (using a simple threshold chosen by a given heuristic or a test or other more sophisticated methods);
3. last, the network is built from the remaining similarities.

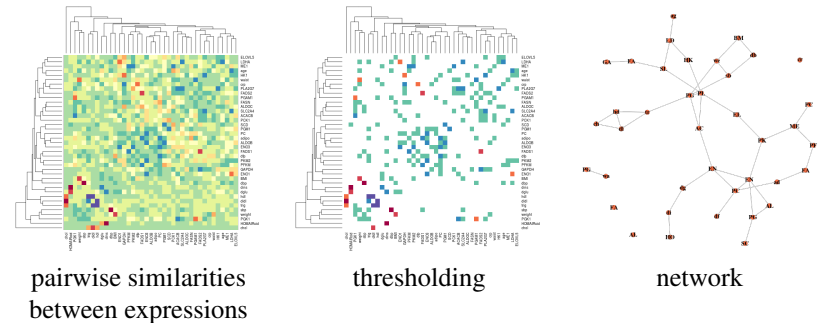


FIGURE 5 – Main steps in network inference

This approach leads to produce *undirected* networks. Additionally, the strength of the relation (i.e., the intensity of the similarity) can be obtained (weighted networks).

2.2 Relevance network

A simple, naive approach to infer a network from gene expression data is to calculate pairwise correlations between genes and then to simply threshold the smallest ones, eventually, using a test of significance. This approach is known under the term *relevance network* [Butte and Kohane, 1999,

Butte and Kohane, 2000]. They can be built with e.g., the R packages WGCNA² [Langfelder and Horvath, 2008] or **hug**. However, if easy to interpret, this approach may lead to strongly misunderstand the regulation relationships between genes. Let's describe the simple following situations displayed in Figure 6. In this model, a single gene, denoted by x , strongly regulates the

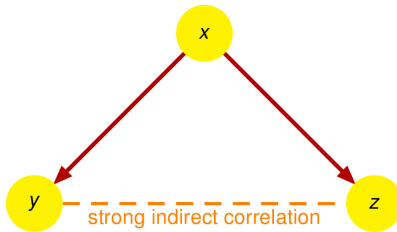


FIGURE 6 – Small model showing the limit of the correlation coefficient to track regulation links : when two genes y and z are regulated by a common gene x , the correlation coefficient between the expression of y and the expression of z is strong as a consequence.

expression between two other genes, y and z . The correlation between the expressions of x and y and the correlation between the expressions of x and z are strong but, as a consequence, the correlation between y and z is also strong : using the simple mathematical model

$$X \sim \mathcal{U}[0, 1], \quad Y \sim 2X + 1 + \mathcal{N}(0, 0.1) \text{ and } Z \sim -X + 2 + \mathcal{N}(0, 0.1)$$

a quick simulation with R gives the following results :

```

set.seed(2807)
x <- rnorm(100)
y <- 2*x+1+rnorm(100, 0, 0.1)
cor(x, y)
# [1] 0.9988261
z <- -x+2+rnorm(100, 0, 0.1)
  
```

```

cor(x, z)
# [1] -0.9950618
cor(y, z)
# [1] -0.9948234
  
```

Hence, even though there is not direct (regulation) link between z and y , these two variables are highly correlated (the correlation coefficient is larger than 0.99) as a result of their common regulation by x . This result is unwanted and using a *partial coefficient* can deal with such strong indirect correlation coefficients.

2.3 Gaussian Graphical Model

In Gaussian Graphical Models, the *conditional dependency graph* is estimated. This graph is defined as follows :

$$j \longleftrightarrow j' (\text{genes } j \text{ and } j' \text{ are linked}) \Leftrightarrow \text{Cor} \left(X^j, X^{j'} | (X^k)_{k \neq j, j'} \right) \neq 0$$

The last quantity is called *partial correlation* and it is equal to the correlation between the residuals of the following linear models :

$$X^j = \sum_{k \neq j, j'} \beta_k^j X^k + \epsilon_j,$$

and

$$X^{j'} = \sum_{k \neq j, j'} \beta_k^{j'} X^k + \epsilon_{j'}.$$

In the above example computed with R, the partial correlations are equal to :

2. <http://labs.genetics.ucla.edu/horvath/CoexpressionNetwork/Rpackages/WGCNA/>

```
cor(lm(x~z)$residuals, lm(y~z)$residuals)
# [1] 0.7801174
cor(lm(x~y)$residuals, lm(z~y)$residuals)
# [1] 0.7639094
cor(lm(y~x)$residuals, lm(z~x)$residuals)
# [1] -0.1933699
```

i.e., to a large value for (x, y) and (x, z) and to a low value for (y, z) , which is exactly what was expected.

2.3.1 Estimating partial correlations in GGM

To build a network using partial correlations, a standard framework is that of Gaussian Graphical Model : the matrix of gene expression \mathbf{X} is supposed to be generated from a Gaussian p -dimensional Gaussian distribution $\mathcal{N}(0, \Sigma)$. In this framework, $\mathbf{S} = \Sigma^{-1}$ is called the *concentration matrix* and is related to partial correlation between X^j and $X^{j'}$, $\pi_{jj'}$, by the following relation :

$$\pi_{jj'} = -\frac{\mathbf{S}_{jj'}}{\sqrt{\mathbf{S}_{jj}\mathbf{S}_{j'j'}}}, \quad (1)$$

which indicates that non zero partial correlations (i.e., edges in the conditional dependency graph) are also non zero entries of the correlation matrix \mathbf{S} . But a major issue when using Σ^{-1} for estimating S is that the empirical estimator of Σ , with entries

$$\hat{\Sigma}_{jj'} := \frac{1}{n} \sum_i (\mathbf{X}_i^j - \bar{X}^j) (\mathbf{X}_i^{j'} - \bar{X}^{j'}) \quad \text{with} \quad \bar{X}^j = \frac{1}{n} \sum_i \mathbf{X}_i^j,$$

calculated from the observations \mathbf{X} , is ill-conditioned because it is calculated with only a few number n of observations. Hence, $\hat{\Sigma}^{-1}$ is a poor estimate of \mathbf{S} and must not be used.

[Schäfer and Strimmer, 2005a, Schäfer and Strimmer, 2005b] proposes to use shrinkage, i.e., to estimate \mathbf{S} with $\hat{\mathbf{S}} = \left(\hat{\Sigma} + \lambda \mathbb{I}\right)^{-1}$ (for a given small $\lambda \in \mathbb{R}^+$) and, optionally, to combine this approach with bootstrap. This method produces a matrix \mathbf{S} with only non zero entries and the problem to decide which of these entries are “significantly” non null is still to be faced.

It can be solved twofold :

- by choosing a given thresholding value or a given number of edges and discarding the smallest partial correlations according to this choice. This approach is not supported by any theoretical framework or any probabilistic model ;
- using a test statistics presented in [Schäfer and Strimmer, 2005a] which is based on a Bayesian model : the partial correlations are supposed to fit a mixture model :

$$\eta_0 f_0 + (1 - \eta_0) f_A$$

where η_0 is the prior for the null hypothesis (i.e., the frequency of null partial correlations), which is supposed to be large. η_0 is then estimated from an EM algorithm (see the original article for further details). This method is implemented in the R package **GeneNet**.

2.3.2 Estimating the conditional dependency graph with Graphical LASSO

The previous method is a two-step method which first estimates the partial correlations and then selects the most significant ones. An alternative method is to simultaneously estimate and select the partial correlations using a sparse penalty. It is known under the name *Graphical LASSO* (or *GLasso*).

Actually, under a Gaussian framework, partial correlation is also related to the estimation of the following linear models :

$$X^j = \sum_{k \neq j} \beta_k^j X^k + \epsilon_j \quad (2)$$

by the relation

$$\beta_k^j = -\frac{\mathbf{S}_{jk}}{\mathbf{S}_{jj}}$$

which, combined with Equation (1) shows again that non zero entries of the linear model coefficients correspond exactly to non zero partial correlations.

Hence, several authors [Meinshausen and Bühlmann, 2006, Friedman et al., 2008] propose to integrate a sparse penalty in the estimation of (2) by ordinary least squares (OLS) :

$$\forall j = 1, \dots, p, \quad \arg \min_{\beta^j} \left[\sum_{i=1}^n \left(\mathbf{X}_i^j - \sum_{k \neq j} \beta_k^j \mathbf{X}_i^k \right)^2 + \lambda \|\beta^j\|_{L^1} \right] \quad (3)$$

where $\|\beta^j\|_{L^1} = \sum_{k \neq j} \beta_k^j$ is the L_1 -norm of $\beta^j \in \mathbb{R}^{p-1}$ which is added to the OLS minimization problem (in black in Equation (3)) in order to force only a restricted number of non zero entries in β^j . λ is a regularization parameter that controls the sparseness of β^j (the larger λ , the fewer the number of non zero entries in β^j). It is generally varied during the learning process and the most adequate value (based on some theoretical results or on a target number of edges) is selected. This method is implemented in the R packages **glasso** or **huge**.

Finally, several approaches have been proposed to deal with the choice of a proper λ : [Liu et al., 2010] proposes the StARS approach which is based on a stability criterion while [Lysen, 2009] and [Foygel and Drton, 2010] propose approaches based on a modification of the BIC criterion. All these methods are implemented in the R package **huge**.

2.4 Other references

For the sake of simplicity, this vignette only deals with undirected network inference and the GGM method. But a number of other approaches are used to infer networks :

- *Bayesian networks* will be able to deal with directed graphs and more precisely, with Directed Acyclic Graphs (DAG, i.e., graphs in which, following a path along the directed edges, you can not come back to the starting node). These methods will not be presented in this vignette and for further information about them, you can refer to [Pearl, 1998, Pearl and Russel, 2002, Scutari, 2010] (also, see the R package **bnlearn**) ;
- when using *longitudinal expression data*, one can infer a directed network with GGM [Opgen-Rhein and Strimmer, 2006, Charbonnier et al., 2010]. The first approach is implemented in the R package **GeneNet** ;
- other methods have been used to infer networks such as, mutual information ([Meyer et al., 2008] and the R package **minet**), random forest [Huynh-Thu et al., 2010], sparse CCA or sparse PLS [Gonzàles et al., 2012] (as in the R package **mixOmics**)...
- network inference can also deal with network with prior structures, such as e.g., latent clusters [Chiquet et al., 2009] or joint estimations [Chiquet et al., 2011, Danaher et al., 2014, Mohan et al., 2012,

Villa-Vialaneix et al., 2014]...

These references are far from being exhaustive.

3 Network mining

In this section, a graph $\mathcal{G} = (V, E)$ is supposed to be given, where $V = \{x_1, \dots, x_p\}$ is the set of vertices and E is the set of edges. Mining a network is the process in which the user extract information about the most important nodes or about groups of nodes that are densely connected.

3.1 Basic definitions

DÉFINITION 1. — [*connectivity*] A graph is said to be connected (*connexe*) if any node in the graph can be reached from any other node by a path (*chemin*) along the edges.

The connected components (*composantes connexes*) of a graph are all its maximal connected subgraphs.

Exemple. —

- My facebook network (see Figure 7) contains 21 connected components having, respectively, 122 vertices (containing my students, some professional contacts, my family and close friends), 5 vertices (who are a group of researchers in computer science), 3 vertices (a group of friends from my neighborhood) and then 2 or 1 vertices (isolated contacts).
- The medieval network's largest connected component has 10 025 vertices, which corresponds to 95% of its vertices ; in this case, the largest connected component is often called a *giant component* / *composante géante*).

DÉFINITION 2. — [*bipartite graph*] A graph is said to be bipartite (*biparti*) if it is composed of two types of vertices with links between vertices with two different types but no link between vertices with the same type.

Standardly, two *projected graphs* can be defined from a bipartite graph : these graphs contain the vertices of a given type and two vertices are linked if they share common links with vertices of the other type. Sometimes, the number of shared links weight this kind of graph.

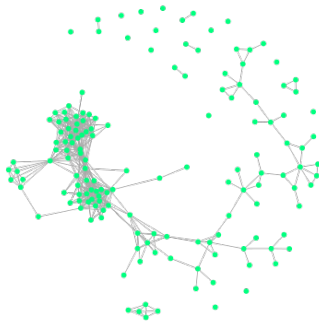


FIGURE 7 – My facebook network with all its 21 connected components

Exemple. — The medieval network is a bipartite graph because it only has edges between a person (first type of vertex) and a transaction (second type of vertex). Two projected graphs can be derived from this graph (or, more interestingly from its largest connected component) :

- the graph of individuals : vertices in this graph are individuals that are actively involved in a transaction (3 755 vertices) and edges are weighted by the number of common transactions between two persons (which can be viewed as a kind of social relationship) ;
- the graph of transactions : vertices are the 6 270 transactions and the edges are model the fact that two transactions involve at least one common person.

3.2 Network visualization


Visualization tools are used to display the graph in a meaningful and aesthetic way. Standard approaches in this area use *force directed placement* algorithms (*algorithmes de force*, see [Fruchterman and Reingold, 1991], among others). The principle of these algorithms is to :

- attach attractive forces to the edges of the graph (similar to springs) to

force connected nodes to be represented close to each others ;

- attach repulsive forces between all pair of vertices (similar to electric forces) to force vertices to be displayed separately.

The algorithm perform iteratively from a (usually random) initial position of the vertices until stabilization. The R package **igraph** (see [Csardi and Nepusz, 2006]) implements several layouts and even several FDP based layouts for static representation of the network.

The free software **Gephi**  can also be used to visualize a network interactively (it supports zooming and panning, among other features). An short video about how Gephi performs the iterative FDP algorithm, is available at <http://youtu.be/naTmvXbr5t4>.

3.3 Global characteristics

This section gives the definition of two global numerical characteristics that can help to understand the network structure and compare one network to another.

DÉFINITION 3. — [*density*] The density (*densité*) of a network is the number of edges divided by the number of pairs of vertices, $\frac{|E|}{p(p-1)/2}$.

Because it is equal to the frequency of edges over the number of possible edges, the density is a measure of how densely connected the network is.

Exemple. —

- My facebook network has 551 edges for 152 vertices, its density is thus equal to $\frac{551}{152 \times 151 / 2} \simeq 4.8\%$. Its largest connected component has 535 edges for 122 vertices, its density is thus equal to $\frac{535}{122 \times 121 / 2} \simeq 7.2\%$, which is much larger.
- The largest connected component of the medieval network network has 10, 025 vertices and 17 612 edges. Its density is equal to 0.035%. The projected networks based on relations between individuals contains 3, 755 vertices and 8, 315 edges : its density is thus much larger, equal to 0.12%.

It is expected that the density tends to decrease with the number of edges because, what is usually observed with real world networks, is that the number of edges is approximately of the order the number of vertices.

DÉFINITION 4. — [transitivity] The transitivity (*transitivité*) of a network is the number of triangles in the network divided by the number of triplets that are connected by at least two edges.

Speaking about a social network, the transitivity thus measures the probability that two of my friends are also friends. A transitivity much larger than the density indicates that the vertices are not connected *at random* but on the contrary that there is a strong local connectivity (a kind of “modular structure”), which is often the case in real-world networks.

Exemple. —

- In the toy example given in Figure 8, the transitivity is equal to $\frac{1}{3}$.

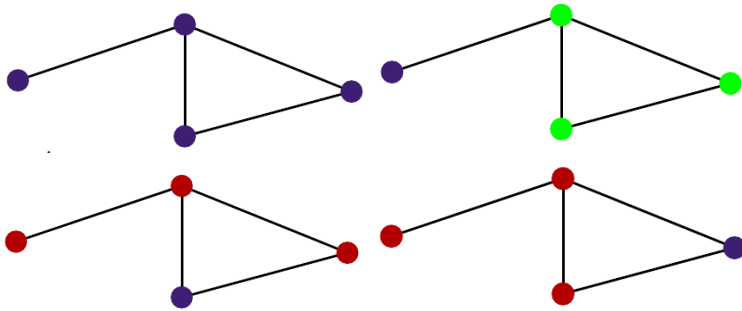


FIGURE 8 – Transitivity calculated on a toy network (top left) : 1 triangle (green vertices, top right) and two triplets with two edges (red vertices, bottom)

- My facebook network has a transitivity equal to 56.2% and its largest connected component has a transitivity equal to 56.0%.
- The medieval network has a transitivity equal to 6.1%.

3.4 Individual characteristics

Once the network structure is analyzed globally, one may want to focus more precisely on vertices individually so has to extract the most “important” ones. Some simple numeric characteristics can be used to do so.

DÉFINITION 5. — [degree] The degree (*degré*) of a vertex v_i is the number of

edges adjacent to this vertex : $d_i = |\{(v_i, v_j) \in E\}|$.

Vertices that have a large degree are called hubs.

The degree is a measure of the vertex “popularity”.

Exemple. —

- In my facebook networks, the four vertices with the highest degrees are four students, two of them have spent three years (instead of just two) to obtain their degree and the two other are from my most recent class (the one with which I still have the most connexions, see Figure 9).

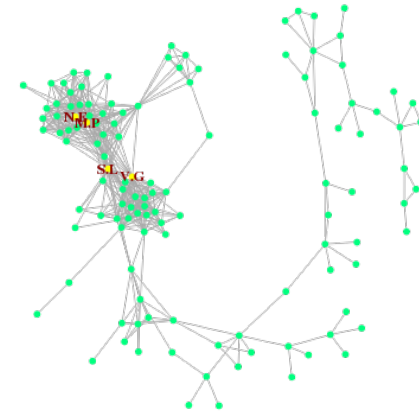


FIGURE 9 – My facebook network with the hubs emphasized : SL and VG are students who have been hold back during university and NE and MP are students from my most recent class.

- In the medieval network, hubs have been studied and were found to be lords, most of them from the most important family in the *seigneurie*

(see Figure 10 for a representation of this network in which the hubs are highlighted).



FIGURE 10 – The medieval social network with names of hubs.

The *degree distribution* (i.e., the values $(P(k))_k$ that count the number of nodes with a given degree k) of many real-world network was found to fit a *power law* : $P(k) \sim k^{-\gamma}$ for a given $\gamma > 0$. Thus, degree distributions are often displayed with log-log scales (i.e., $\log P(k)$ versus $\log k$) : in this case, a good linear fit indicates a power law distribution.

Exemple. —

- The degree distribution of my facebook network’s largest connected component is given in Figure 11. It doesn’t seem to fit a power law but the size of this network is probably too small for this kind of distribution to be observed. Nevertheless, one can see that most vertices have a very low degree whereas a few vertices have a very high degree.
- When considering the medieval network, the degree distribution restric-

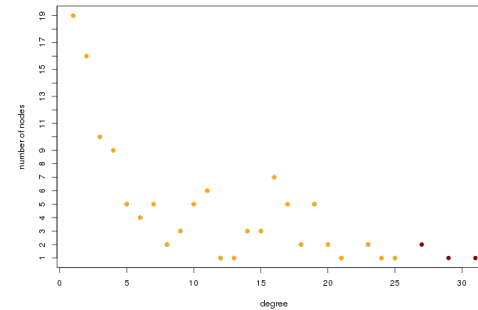


FIGURE 11 – The degree distribution of my facebook network.

ted to the transactions is the number of persons involved in a transaction. It is given in Figure 12 with a log-log scale : it almost fit a power law.

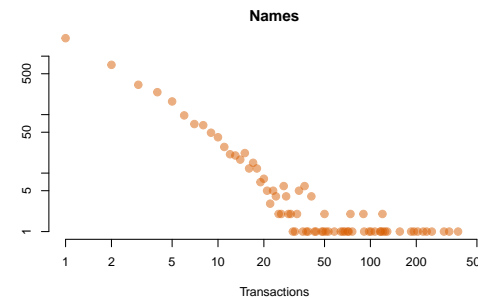


FIGURE 12 – The degree distribution of the transaction network built from the medieval corpus.

DÉFINITION 6. — *[betweenness]* The betweenness (*centralité*) of a vertex v is the number of shortest paths between any pair of vertices that pass through this vertex.

The betweenness is a centrality measure : vertices that have a large between-

ness are those that are the most likely to disconnect the network if removed.

Exemple. —

- In the toy example provided in Figure 13, the betweenness of the orange node is equal to 4 as shown in Figure 14 in which all shortest paths that pass through this vertex are emphasized in yellow.

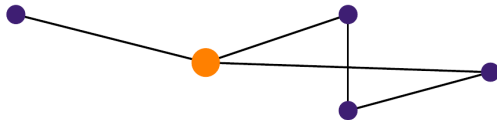


FIGURE 13 – A toy example : the orange node’s betweenness is equal to 4 (see Figure 14).

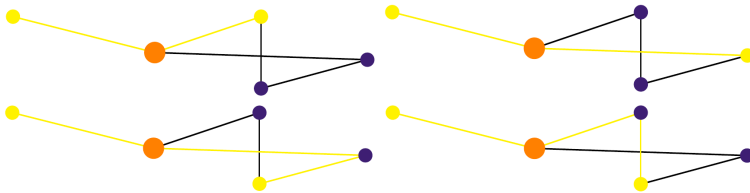


FIGURE 14 – A toy example : Shortest paths between pairs of nodes that pass through the orange vertex are shown in yellow.

- In my facebook network, the betweenness distribution is displayed by its density in Figure 15. As shown in this figure, it is heavily skewed. The two vertices with the highest betweenness are displayed in Figure 16 : they are two public persons who connect vertices very different frameworks.
- In the network built from the medieval network by projecting individuals, most vertices with a high betweenness are also hubs (this result is expected). However, some vertices have a high betweenness with a low degree : they are mostly moral persons (such as “Chapter of Cahors” or “Church of Flaugnac” or transcription errors that connect parts of the networks having different dates).

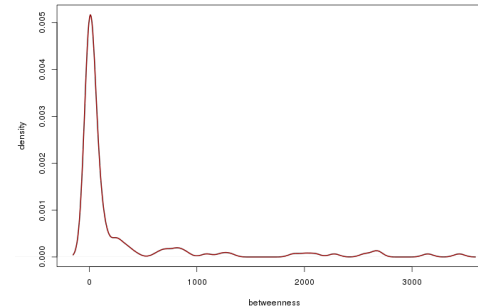


FIGURE 15 – Betweenness distribution in my facebook network (density).

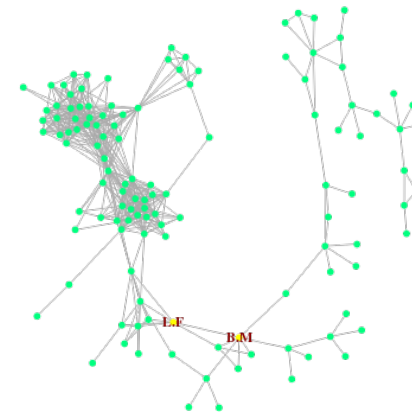


FIGURE 16 – My facebook network with highest betweenness emphasized : BM and LF are two public persons having in political activities and thus connecting very distinct groups of people.

3.5 Clustering *Classification*

Clustering vertices in a network refers to the task that consists in partitioning the network into densely connected groups that share a few number of edges (comparatively) with the other groups. Clusters are often called *communities* (*communautés*) in social sciences and *modules* (*modules*) in biology. A number of methods have been designed to address this issue and this vignette is much too small to go beyond scratching the surface of this topic. For further references on this topic, I suggest the reading of the reviews [Fortunato and Barthélemy, 2007, Schaeffer, 2007].

One of the most popular approach consists in maximizing a quality criterion called *modularity* [Newman and Girvan, 2004] :

DÉFINITION 7. — [*modularity*] Given a partition $(\mathcal{C}_1, \dots, \mathcal{C}_K)$ of the nodes of the graph, the modularity (modularity) of the partition is equal to

$$\mathcal{Q}(\mathcal{C}_1, \dots, \mathcal{C}_K) = \frac{1}{2m} \sum_{k=1}^K \sum_{v_i, v_j \in \mathcal{C}_k} (\mathbb{I}_{(v_i, v_j) \in E} - P_{ij})$$

where $P_{ij} = \frac{d_i d_j}{2m}$ and $m = |E|$ is the number of edges in the network.

In this definition, P_{ij} plays the role of a probability to have an edge between v_i and v_j according to a “null model”. The “null model” refers to a model where the edges depends only on the degrees of each vertex and not on the clusters themselves : the larger the modularity is, the more the edges are concentrated in the clusters $(\mathcal{C}_j)_j$. This model is close to maximizing the number of edges in the clusters but is not exactly identical to it : edges that correspond to vertices with a large degree have a lesser impact in the modularity value : this aims at encompassing in the criterion the notion of *preferential attachment* [Barabási and Albert, 1999] : preferential attachment is the fact that, in real networks, people tend to connect preferably with people who already have a large number of connections. Hence, these kind of edges seem to be less “significant” (or, in other words, less important to define an homogeneous community). In particular, modularity is known to better separate hubs (contrary to a naive approach consisting in minimizing the number of edges between clusters, that leads more frequently to have huge clusters and tiny ones with isolated vertices). Also, the modularity is not monotonous in the number of clusters : it

can thus be useful to find an adequate number of clusters. However, it is also known to fail detecting small communities [Fortunato and Barthélemy, 2007].

Maximizing the modularity is not an easy task because this problem is known to be NP-complete (i.e., when the network size is multiplied by 2, the time needed to compute an exact solution is multiplied by more than a power of 2). The most popular fast algorithms which approximate a solution of the maximization problem are multi-level greedy algorithms, such as those described in [Blondel et al., 2008, Noack and Rotta, 2009, Rossi and Villa-Vialaneix, 2011] (the first one is better known under the name of “Louvain algorithm” ; the last one is available as a Java program at <http://apiacoa.org/research/software/graph/index.en.html>).

Exemple. —

- Using the Louvain algorithm, my facebook network can be partitioned into communities as in Figure 17. The maximum modularity found is equal to 0.567 and communities can be associated with (approximate) labels describing the people who are classified inside.
- The medieval network community detection has already been represented in Figure 10. Underlining communities on the figure help understand its structure and the way the main persons interact.

3.6 A few more references on clustering

As already said, this vignette does not really tackle the problem of node clustering in its whole. If you are interested in the subject, I recommend you this (short and very incomplete) list of references :

- about *spectral clustering* (which is an alternative and very popular method to cluster vertices), the article [von Luxburg, 2007] ;
- about using *generative (Bayesian) models* to cluster vertices, [Zanghi et al., 2008] ;
- about comparing different clustering algorithms for PPI networks, the article [Brohée and van Helden, 2006] ;
- about using SOM to cluster vertices and simultaneously project it on a map to help visualize it, the articles [Boulet et al., 2008, Rossi and Villa-Vialaneix, 2010] ;
- the revues [Fortunato and Barthélemy, 2007, Schaeffer, 2007] also give useful hints on overlapping communities *communautés chevauchantes*,

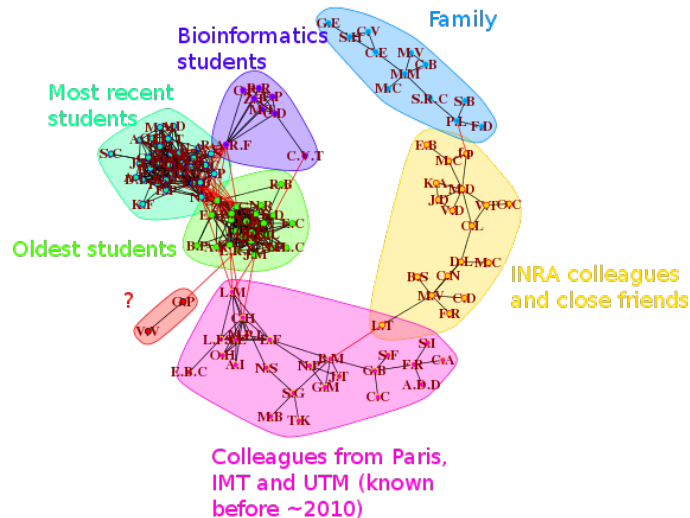


FIGURE 17 – My facebook network with communities emphasized by different colors and labels.

hierarchical clustering, ...

Once the communities found, the network (if big enough) can be represented in a simplified way using these communities. This can help the user understand what are the relations between the main clusters and then to focus more precisely on a few number of clusters. This topic is not shown in this vignette but more references can be found at <http://apiacoa.org/research/software/graph/index.en.html>.

4 Applications

These applications are performed using the free statistical software environment R. The packages used in these applications are :

- **glasso** for network inference ;

- **igraph** for network objects and mining.

The interested reader may want to have a look at the “gRaphical Models in R” task view where he/she will find further interesting packages.

4.1 Gene network inference

This subsection will show basic functions to infer a network from gene expression data. A very short analysis of the obtained result is also provided, mostly aiming at comparing a clustering of genes obtained from the graph and a clustering of genes obtained from hierarchical clustering on expression data. Further commands for graph mining are presented in Section 4.2.

4.1.1 Presentation of the dataset

The dataset used in this section is included in the package **mixOmics** and can be loaded using :

```
# load data (gene expression)
library(mixOmics)
data(nutr mouse)
summary(nutr mouse)

# Length Class      Mode
# gene             120    data.frame list
# lipid             21    data.frame list
# diet              40    factor      numeric
# genotype          40    factor      numeric
```

Although the data contain various information such as the diet or the genotype, only gene expression will be used for graph inference. It could have been interesting to infer different networks depending from, e.g., the genotype and to compare them but this task is out of the scope of this simple vignette.

Gene expression distribution can be visualized with the following command lines and is provided in Figure 18.

```
expr <- nutr mouse$gene
rownames(expr) <- paste(1:nrow(expr),
                        nutr mouse$genotype,
                        nutr mouse$diet,
                        sep="-")
```

```
boxplot(expr, names=NULL)
```

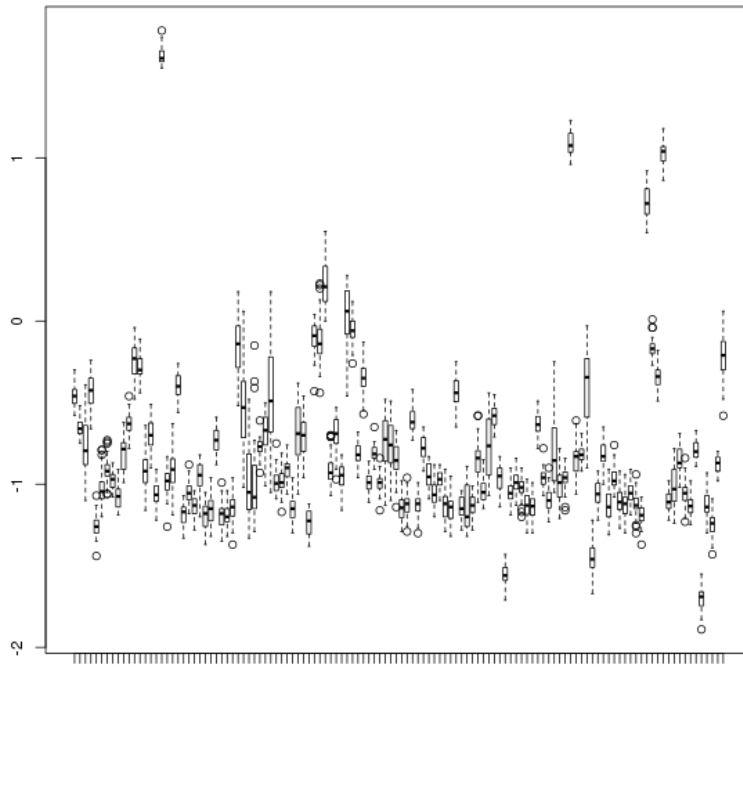


FIGURE 18 – Distribution of the nutrimore gene expressions

The gene expression are mostly symmetric with no remarkable outliers which is fit to use a GGM.

The (simple) correlations between gene expression are displayed in the heatmap of Figure 19, that shows over and under expressed gene depending on the genotype and the diet.

```
expr.c <- scale(expr)
heatmap(as.matrix(expr.c))
```

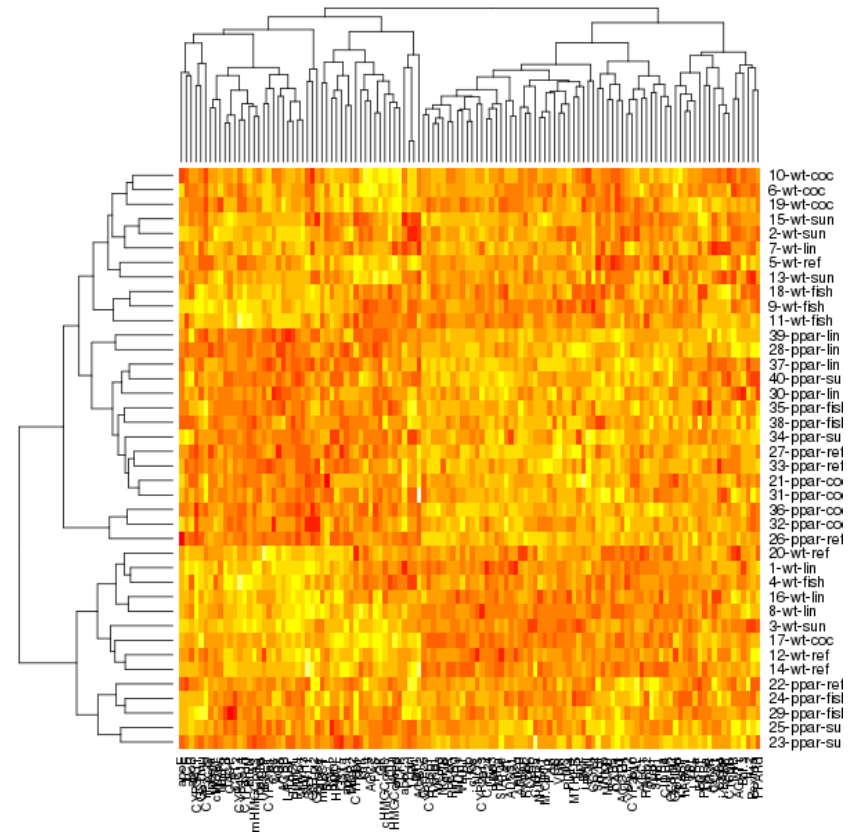


FIGURE 19 – Correlation between nutrimore gene expressions

Finally, using this heatmap, 7 groups are extracted that are shown on the dendrogram of Figure 20 and corresponds to clusters that have a size ranging from 3 to 52.


```

hclust.tree <- hclust(dist(t(expr.c)))
plot(hclust.tree)
rect.hclust(hclust.tree, k=7, border=rainbow(7))
hclust.groups <- cutree(hclust.tree, k=7)
table(hclust.groups)
hclust.groups
# 1 2 3 4 5 6 7
# 18 4 20 52 17 6 3
    
```

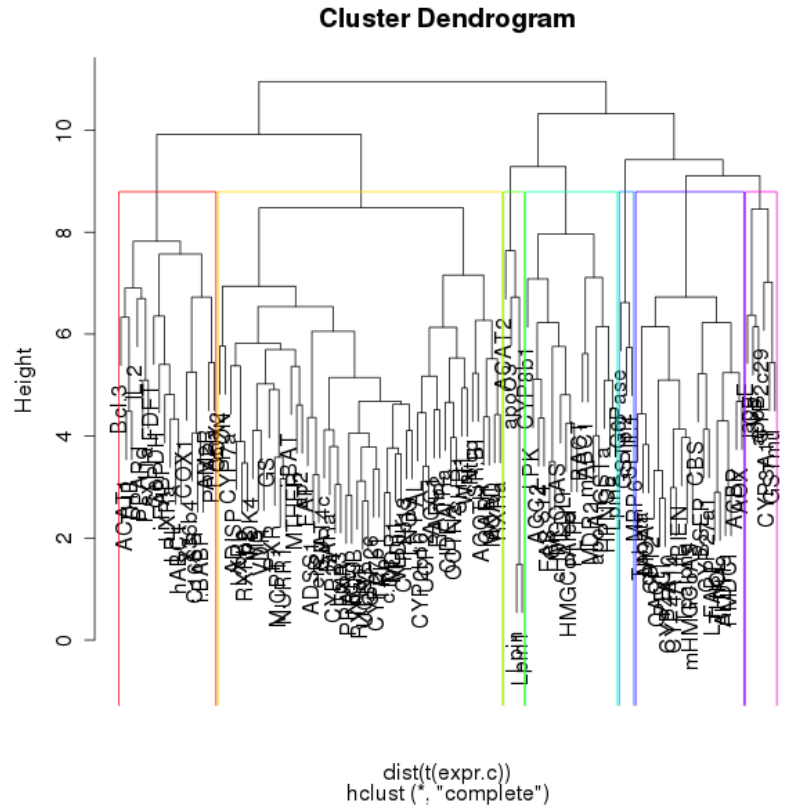


FIGURE 20 – 6 clusters extracted from nutrimouse gene expressions

4.1.2 Network inference

Network inference will be performed using the R package **huge**. This package contains a function able to compute inferred network for a set of different values for the sparse regularization parameter λ of Equation (3), giving networks that are more or less dense. If no values for λ are provided, the function uses its own (hopefully reasonable) values :

```
# Inference
library(huge)
glasso.res <- huge(as.matrix(expr), method="glasso")
glasso.res
# Model: graphical lasso (glasso)
# Input: The Data Matrix
# Path length: 10
# Graph dimension: 120
# Sparsity level: 0 -----> 0.2128852
```

The successive estimated inverses of the covariance matrix (for different λ s) can be evaluated using the function `plot` on the resulting object

```
plot(glasso.res)
```

which displays the evolution of the network density versus the value of λ and a few network visualization along the λ path, as shown in Figure 21.

The “best” network (according to a stability criterion, StARS) can be selected using the function `huge.select` :

```
glasso.sel <- huge.select(glasso.res,
                        criterion="stars")
# glasso.sel
# Model: graphical lasso (glasso)
# selection criterion: stars
# Graph dimension: 120
# sparsity level 0.05490196
```

which returns a graph with a density about 5.5%. The output of the selection processed can be visualized using the function `plot` on the resulting object :

```
plot(glasso.sel)
```

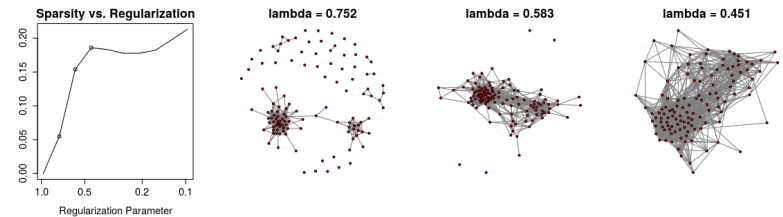


FIGURE 21 – Visualization of the results of the inference with GLasso along the λ path

which gives the Figure 22.

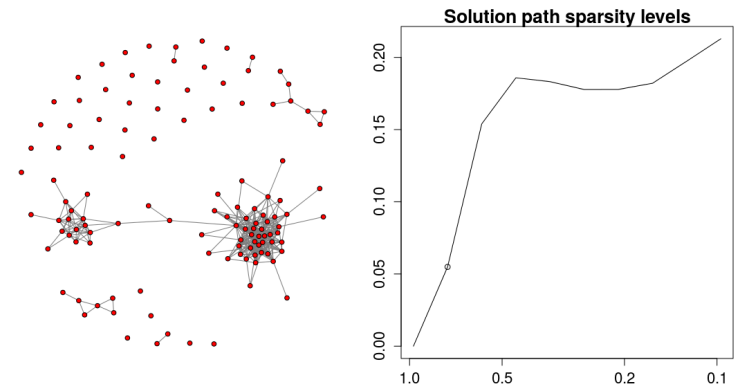


FIGURE 22 – Visualization of the outputs of the λ selection using the StARS approach

The package **huge** can also be used to infer relevance networks (*i.e.*, networks based on thresholding the Pearson correlation) :

```
relevance.res <- huge(as.matrix(expr), method="ct",
                    lambda=seq(0.6, 1, length=10))
relevance.res
# Model: graph estimation via correlation
#   thresholding (ct)
# Input: The Data Matrix
# Path length: 10
# Graph dimension: 120
# Sparsity level: 0 -----> 0.2182073
plot(relevance.res)
```

which gives Figure 23. Similarly as what was previously performed, the func-

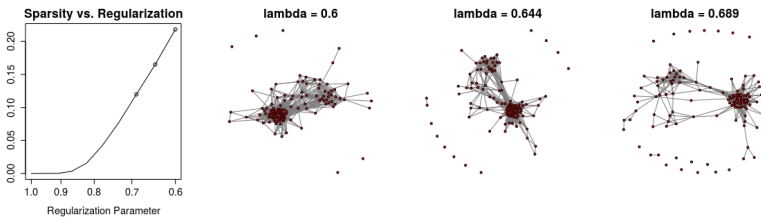


FIGURE 23 – Visualization of the results of the inference with Pearson correlation thresholding along the thresholding path

tion `huge.select` can be used on this object to select the most stable threshold.

The graph with 392 edges coming from the GLasso inference (*i.e.*, the one selected with the StARS method that has a density about 5.5%) is finally kept

and will be analyzed in the following.

4.1.3 Building the network

To define a graph in R, the **igraph** package can be used. Graphs are stored as **igraph** objects and an **igraph** object can be constructed from the functions `graph.edgelist`, `graph.data.frame`, `graph.adjacency`, depending on your data's format. The following commands show how to build a network from an binary (TRUE/FALSE) adjacency matrix :

```
# build graphs
library(igraph)
bin.mat <- as.matrix(glasso.sel$opt.icov) != 0
colnames(bin.mat) <- colnames(expr)
nutrimouse.net <- simplify(graph.adjacency(bin.mat,
mode="max"))
nutrimouse.net
# IGRAPH UN-- 120 392 --
# + attr: name (v/c)
is.connected(nutrimouse.net)
# [1] FALSE
```

As shown by the function `is.connected`, the resulting network is not connected. Its connected components can be found with the function `clusters`:

```
components.nutrimouse <- clusters(nutrimouse.net)
components.nutrimouse
# [1] 1 2 3 2 2 4 5 2 2 2 6 2 7 2
# ...
# $csize
# [1] 7 67 1 6 2 1 1 1 1 1 1 1 1 2
# ...
# $no
# [1] 40
```

The network contains 40 connected components and the largest one has 67 nodes. It can be extracted with the function `induced.subgraph` :

```

nutrimouse.lcc <- induced.subgraph(nutrimouse.net,
  components.nutrimouse$membership==
    which.max(components.nutrimouse$csize))
nutrimouse.lcc
# IGRAPH UN-- 67 375 --
# + attr: name (v/c)

```

4.1.4 Short mining of the network

igraph also comes up with tools to perform graph mining. For instance, the graph can be visualized using the function `plot`³ that displays the graph as in Figure 24 :

```

par(mar=rep(0.1, 4))
set.seed(1832)
nutrimouse.lcc$layout <- layout.kamada.kawai(
  nutrimouse.lcc)
plot(nutrimouse.lcc, vertex.size=2,
  vertex.color="lightyellow",
  vertex.frame.color="lightyellow",
  vertex.label.color="black",
  vertex.label.cex=0.7)

```

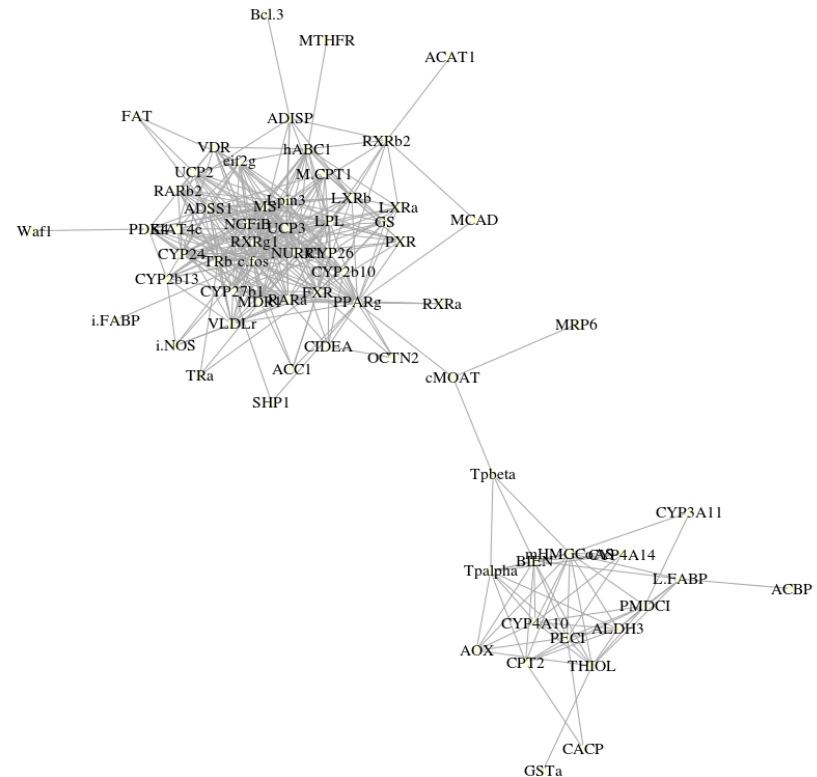


FIGURE 24 – Visualization of the network inferred from nutrimouse gene expressions

3. More information on this function is provided in Section 4.2.

Algorithms that perform node clustering can also be used, such as the one implemented in the function `spinglass.community`⁴. As this function involved a stochastic process, its result depends on the initialization. It is thus recommended to run it several times and to keep the best results (on a modularity point of view). For the sake of simplicity, we will only use one call of the function and set the random seed before to obtain identical results.

```
set.seed(66)
clusters.nutrimouse <-
  spinglass.community(nutrimouse.lcc)
clusters.nutrimouse
# Graph community structure calculated with the
# spinglass algorithm
# Number of communities: 4
# Modularity: 0.3136604
# Membership vector:
#   ACAT1   ACBP   ACC1   ADISP   ADSS1
#     1     3     1     1     2
# ...
table(clusters.nutrimouse$membership)
# 1  2  3  4
# 15 25 17 10
```

4 clusters are extracted by the algorithm for a modularity equal to 0.314 with varying sizes ranging from 10 to 25. This clustering can be compared to the one previously obtained by a simple hierarchical clustering on the gene expressions.

```
induced.hclust.groups <- hclust.groups[
  components.nutrimouse$membership==
  which.max(components.nutrimouse$cszsize)]
table(induced.hclust.groups)
# 1  3  4  5  6
# 6 17 42  1  1
```

Hierarchical clustering provides 5 clusters for the nodes of the largest connected components of the graph, those sizes vary from 1 to 42. The comparison

between the two clusterings can be done by parallel visualizations as shown in Figure 25 or using the function `compare.communities`:

```
par(mfrow=c(1,2))
par(mar=rep(1,4))
plot(nutrimouse.lcc, vertex.size=5,
     vertex.color=rainbow(6)[induced.hclust.groups],
     vertex.frame.color=
       rainbow(6)[induced.hclust.groups],
     vertex.label=NA,
     main="Hierarchical clustering")
plot(nutrimouse.lcc, vertex.size=5,
     vertex.color=rainbow(4)[
       clusters.nutrimouse$membership],
     vertex.frame.color=rainbow(4)[
       clusters.nutrimouse$membership],
     vertex.label=NA, main="Graph clustering")
compare.communities(induced.hclust.groups,
                    clusters.nutrimouse$membership,
                    method="nmi")
# [1] 0.5091568
```

The index used here to compare the two clustering is the normalized mutual information as given in [Danon et al., 2005]. It ranges between 0 and 1, 1 being found when the two partitions are identical. The value found by this function as well as the visualization show that the two clusterings are rather different, the one obtained from the network itself having more balanced clusters and also a larger modularity:

```
modularity(nutrimouse.lcc, induced.hclust.groups)
# [1] 0.2368462
```

4.2 Facebook network mining

This subsection will show basic functions to display a graph, calculate its main numerical characteristics and compute communities with the R package

4. More information on this function and on node clustering is provided in Section 4.2.

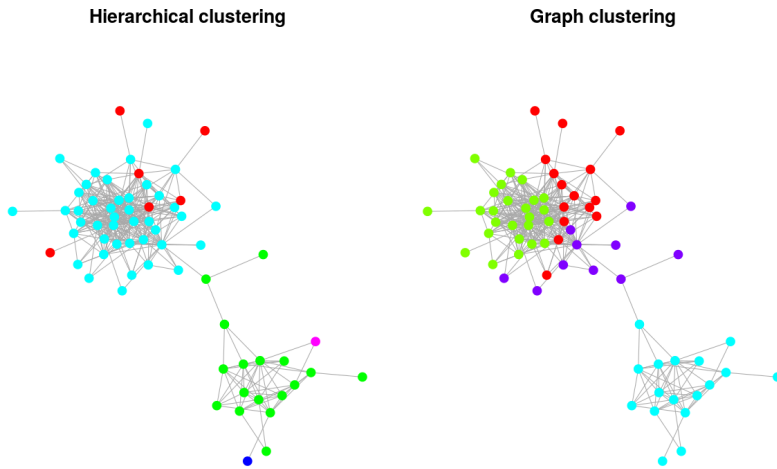


FIGURE 25 – Visualization of two clusterings obtained from nutrilogue gene expressions : hierarchical clustering (left) and node clustering (right)

igraph. All manipulations are done using my facebook network⁵. Most of the information extracted in this section have already been shown and/or commented in the previous sections.

The data can be downloaded from two datasets :

- `fbnet-el-1.txt` that contains the edgelist (text format) ;
- `fbnet-names-1.txt` that contains the names of the vertices (initials actually).

```
edgelist <- as.matrix(read.table("fbnet-el-1.txt"))
vnames <- read.table("fbnet-name-1.txt")
vnames <- as.character(vnames[,1])
```

5. You can use your own facebook network to test this script after having downloaded it with the on-line application <http://shiny.nathalievilla.org/fbs>. This application has been developed using the R package **shiny**, <http://www.rstudio.com/shiny>.

Building an igraph object

igraph object can be constructed from the functions `graph.edgelist`, `graph.data.frame`, `graph.adjacency`, depending on your data's format. The two following commands build a network from an edge list only :

```
# with 'graph.edgelist'
fbnet0 <- graph.edgelist(edgelist, directed=FALSE)
# network summary
fbnet0
# IGRAPH U--- 152 551 --
```

Vertices are accessed using the function `V` and the number of vertices using the function `vcount` :

```
# vertices
V(fbnet0)
vcount(fbnet0)
```

igraph can handle attributes on vertices, edges and also graph attributes. The following lines add the attributes `initials` to the vertices of the graph :

```
# add an attribute for vertices
V(fbnet0)$initials <- vnames
fbnet0
# IGRAPH U--- 152 551 --
# + attr: initials (v/x)
```

Edges are accessed using the function `E` and the number of edges using the function `ecount` :


```
# edges
head(E(fbnet0))
# [1] 11 -- 1
# [2] 41 -- 1
# [3] 52 -- 1
# [4] 69 -- 1
# [5] 74 -- 1
# [6] 75 -- 1
ecount(fbnet0)
# 551
```

(first five edges are printed).

Macro-characteristics of the graph

The first step is usually to check if the network is connected :

```
is.connected(fbnet0)
# [1] FALSE
```

As this network is not connected, the connected components can be extracted :

```
fb.components <- clusters(fbnet0)
names(fb.components)
# [1] "membership" "csize" "no"
head(fb.components$membership, 10)
# [1] 1 1 2 2 1 1 1 1 3 1
fb.components$csize
# [1] 122 5 1 1 2 1 1 1 2 1
# [11] 1 2 1 1 2 3 1 1 1 1
# [21] 1
fb.components$no
# [1] 21
```

As the network is composed of 21 components with only one containing more than 5 nodes, the rest of the analysis will focus on this component only. It is called the *largest connected component of the graph* and can be extracted by :

```
fbnet.lcc <- induced.subgraph(fbnet0,
                             fb.components$membership==
                             which.max(fb.components$csize))
# main characteristics of the LCC
fbnet.lcc
# IGRAPH U--- 122 535 --
# + attr: initials (v/x)
is.connected(fbnet.lcc)
# [1] TRUE
```

Finally, the graph's density and the transitivity can be found this way :

```
graph.density(fbnet.lcc)
# [1] 0.0724834
transitivity(fbnet.lcc)
# [1] 0.5604524
```

The transitivity of the graph is much larger than its density, which indicates dense local patterns.

4.2.1 Visualization

The function `plot` applied to an **igraph** object displays the graph on the graphic device. This function possesses several handy options (that begins with `vertex.`) to handle node positions, size, colors, labels and similarly for edges (these options begins with `edge.`). The layout of the graph can be attached to the **igraph** object, using the attribute `layout`.

The following command lines :

```

par(mfrow=c(2,2))
# default layout
plot(fbnet.lcc, layout=layout.random,
     main="random layout", vertex.size=3,
     vertex.color="pink", vertex.frame.color="pink",
     vertex.label.color="darkred",
     edge.color="grey",
     vertex.label=V(fbnet.lcc)$initials)
plot(fbnet.lcc, layout=layout.circle,
     main="circular layout", vertex.size=3,
     vertex.color="pink", vertex.frame.color="pink",
     vertex.label.color="darkred", edge.color="grey",
     vertex.label=V(fbnet.lcc)$initials)
plot(fbnet.lcc, layout=layout.kamada.kawai,
     main="Kamada Kawai layout",
     vertex.size=3, vertex.color="pink",
     vertex.frame.color="pink",
     vertex.label.color="darkred", edge.color="grey",
     vertex.label=V(fbnet.lcc)$initials)
# set the attribute 'layout' for the graph
fbnet.lcc$layout <-
  layout.fruchterman.reingold(fbnet.lcc)
V(fbnet.lcc)$label <- V(fbnet.lcc)$initials
plot(fbnet.lcc,
     main="Fruchterman & Reingold layout",
     vertex.size=3, vertex.color="pink",
     vertex.frame.color="pink",
     vertex.label.color="darkred", edge.color="grey"

```

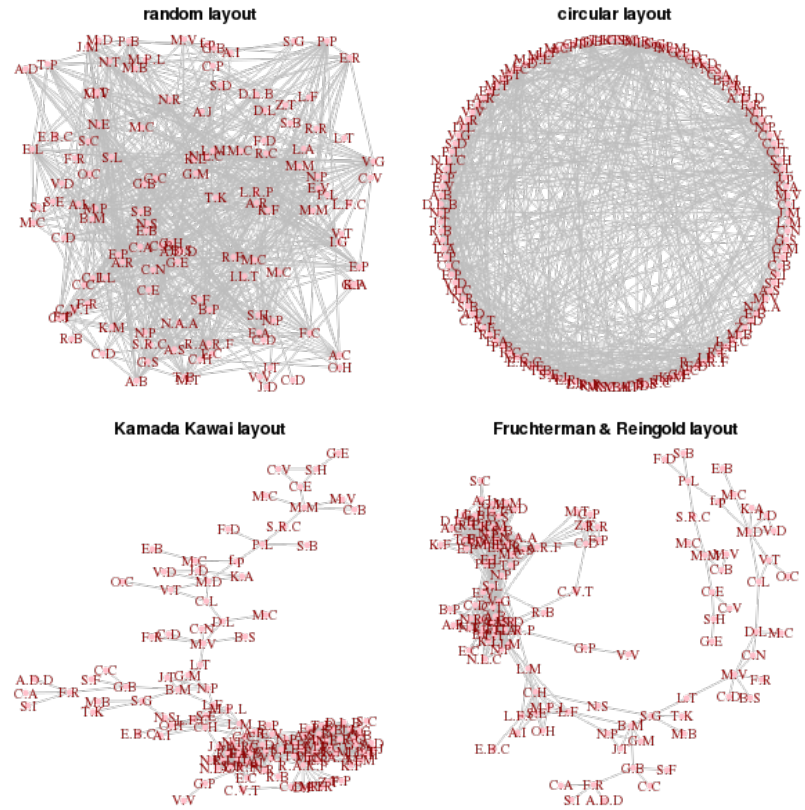



FIGURE 26 – Different layouts obtained with plot.igraph and the functions layout.

give the representations provided in Figure 26. Due to randomness into some of the algorithms that display graph, the results obtained at each execution of these lines may differ a little but the overview should remain the same⁶.

For further details and options on graph visualization with **igraph**, refer to help(igraph.plotting). Interactive graph representation can be obtai-

ned with tkplot but we advice the user to use rather **Gephi**  than **igraph** for interactive visualizations.

6. Using the function set.seed before calling a layout generating function fixes the randomness and provides reproducible results.

4.2.2 Individual characteristics

Degrees are calculated with the command `degree` and betweenness is calculated with the command `betweenness` :

```
fbnet.degrees <- degree(fbnet.lcc)
summary(fbnet.degrees)
#   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#   1.00   2.00   6.00   8.77  15.00  31.00
fbnet.between <- betweenness(fbnet.lcc)
summary(fbnet.between)
#   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#   0.00   0.00  14.03  301.70 123.10 3439.00
```

Both degree and betweenness distribution are skewed as their means are larger (or even much larger) than their betweenness centrality measures. This fact is standard in many real-life networks where most nodes have a small degree (or betweenness) whilst a few number of nodes have a very large degree (or betweenness) in comparison.

The distributions can be visualized using, e.g., a density plot :

```
par(mfrow=c(1,2))
plot(density(fbnet.degrees), lwd=2,
     main="Degree distribution", xlab="Degree",
     ylab="Density", col="darkred")
plot(density(fbnet.between), lwd=2,
     main="Betweenness distribution",
     xlab="Betweenness", ylab="Density",
     col="darkred")
```

that gives Figure 27 and confirms our first guess.

Finally, this information can be combined with graph visualization to emphasize the most important nodes on the network, which gives the representation of Figure 28.

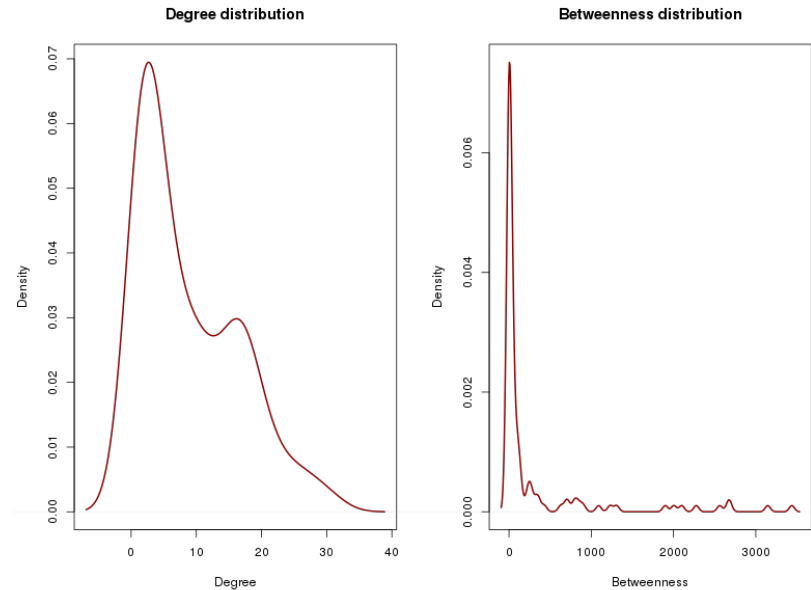


FIGURE 27 – Degree and betweenness densities.

```
par(1)
par(mar=rep(1,4))
V(fbnet.lcc)$size <- 2*sqrt(fbnet.degrees)
bet.col <- cut(log(fbnet.between+1),10,
              labels=FALSE)
V(fbnet.lcc)$color <- heat.colors(10)[11-bet.col]
plot(fbnet.lcc, main="Degree and betweenness",
     vertex.frame.color=heat.colors(10)[bet.col],
     vertex.label=NA, edge.color="grey")
```

This representation is particularly handy since it shows that high betweenness nodes are not necessarily hubs. Actually, the nodes having the highest betweenness are those that are connecting different “parts” of the network, whereas the hubs are mostly located in its densest part.

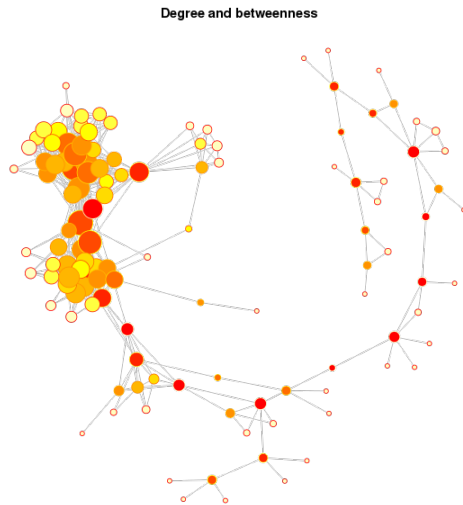


FIGURE 28 – Degree (node sizes are proportional to degrees) and betweenness (node colors represents betweenness with red colors for high betweenness)

4.2.3 Clustering

Several algorithms can be used to cluster networks into communities. Most of them are based on the maximization of the modularity (with an exact or an approach method; exact methods have a prohibited computational time for networks with more than a hundred nodes). In the following the function `spinglass.community` is used. This algorithm is the one described in [Reichardt and Bornholdt, 2006] which optimizes a criterion that generalizes the modularity with a deterministic algorithm (default options lead to modularity optimization). The results obtained when the random seed is not fixed may differ from one call to another but the different results should be very similar. More information on the methods used in the different algorithms is provided with `help(communities)`.

```
fbnet.clusters <- spinglass.community(fbnet.lcc)
fbnet.clusters
# Graph community structure calculated with
# the spinglass algorithm
# Number of communities: 9
# Modularity: 0.5654136
# Membership vector:
# [1] 9 5 6 5 5 9 1 9 1 1 4 9 9 6 2 7 2 7 2 7 5
# [21] 5 3 8 1 4 7 9 4 4 2 7 7 4 5 9 3 6 7 5 7 9
# [41] 4 5 7 3 4 9 9 8 9 6 9 9 5 9 9 9 9 9 4 9 7
# [61] 9 9 9 4 2 9 5 4 9 5 4 3 4 9 4 4 4 1 7 4 4
# [81] 7 4 4 4 9 3 4 4 7 7 4 6 5 4 4 1 4 5 3 4 4
# [101] 4 7 7 3 6 5 4 1 4 2 2 1 3 7 4 6 7
table(fbnet.clusters$membership)
# 1 2 3 4 5 6 7 8 9
# 8 7 8 32 14 7 18 2 26
```

This algorithm leads to partition the graph into 9 clusters with a modularity equal to 0.565 (there is no upper bound to compare the modularity with, although its absolute value is always smaller than 1; [Rossi and Villa-Vialaneix, 2011] proposes to use a test based on random graphs to check whether this value is significant or not). The clusters' size range from 2 to 32. The community number can be added to the data as a node attribute (usefull if you wan to use it in an external program, such as

Gephi :

```
V(fbnet.lcc)$community <- fbnet.clusters$membership
fbnet.lcc
# IGRAPH U--- 122 535 --
# + attr: layout (g/n), initials (v/c),
# label (v/c), size (v/n), color (v/c),
# community (v/n)
```

Finally, the clustering can be visualized on the graph with by using the following command lines :

```
par(mfrow=c(1,1))
par(mar=rep(1,4))
plot(fbnet.lcc, main="Communities",
     vertex.frame.color=
       rainbow(9)[fbnet.clusters$membership],
     vertex.color=
       rainbow(9)[fbnet.clusters$membership],
     vertex.label=NA, edge.color="grey")
```

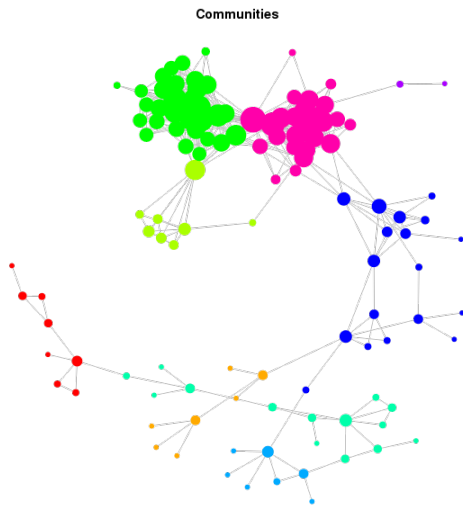


FIGURE 29 – 9 communities found by the algorithm `spinglass.communities`

4.2.4 Export graph

For external use, the graph can be exported as a `graphml` file (that can be read by most of the graph visualization programs and that includes information on node and edge attributes) :

```
write.graph(fbnet.lcc, file="fblcc.graphml",
```

```
format="graphml")
```

Further information on exportation format are provided with `help(write.graph)`.

Références

- [Barabási and Albert, 1999] Barabási, A. and Albert, R. (1999). Emergence of scaling in random networks. *Science*, 286 :509–512.
- [Blondel et al., 2008] Blondel, V., Guillaume, J., Lambiotte, R., and Lefebvre, E. (2008). Fast unfolding of communities in large networks. *Journal of Statistical Mechanics : Theory and Experiment*, P10008 :1742–5468.
- [Boulet et al., 2008] Boulet, R., Jouve, B., Rossi, F., and Villa, N. (2008). Batch kernel SOM and related Laplacian methods for social network analysis. *Neurocomputing*, 71(7-9) :1257–1273.
- [Brohée and van Helden, 2006] Brohée, S. and van Helden, J. (2006). Evaluation of clustering algorithms for protein-protein interaction networks. *BMC Bioinformatics*, 7(488).
- [Butte and Kohane, 1999] Butte, A. and Kohane, I. (1999). Unsupervised knowledge discovery in medical databases using relevance networks. In *Proceedings of the AMIA Symposium*, pages 711–715.
- [Butte and Kohane, 2000] Butte, A. and Kohane, I. (2000). Mutual information relevance networks : functional genomic clustering using pairwise entropy measurements. In *Proceedings of the Pacific Symposium on Biocomputing*, pages 418–429.
- [Charbonnier et al., 2010] Charbonnier, C., Chiquet, J., and Ambroise, C. (2010). Weighted-lasso for structured network inference from time course data. *Statistical Applications in Genetics and Molecular Biology*, 9(1).
- [Chiquet et al., 2011] Chiquet, J., Grandvalet, Y., and Ambroise, C. (2011). Inferring multiple graphical structures. *Statistics and Computing*, 21(4) :537–553.
- [Chiquet et al., 2009] Chiquet, J., Smith, A., Grasseau, G., Matias, C., and Ambroise, C. (2009). SIMoNe : Statistical Inference for MODular NETworks. *Bioinformatics*, 25(3) :417–418.

- [Csardi and Nepusz, 2006] Csardi, G. and Nepusz, T. (2006). The igraph software package for complex network research. *InterJournal, Complex Systems*.
- [Danaher et al., 2014] Danaher, P., Wang, P., and Witten, D. (2014). The joint graphical lasso for inverse covariance estimation across multiple classes. *Journal of the Royal Statistical Society Series B*, 76(2) :373–397.
- [Danon et al., 2005] Danon, L., Diaz-Guilera, A., Duch, J., and Arenas, A. (2005). Comparing community structure identification. *Journal of Statistical Mechanics*, page P09008.
- [Fortunato and Barthélemy, 2007] Fortunato, S. and Barthélemy, M. (2007). Resolution limit in community detection. In *Proceedings of the National Academy of Sciences*, volume 104, pages 36–41. doi :10.1073/pnas.0605965104; URL : <http://www.pnas.org/content/104/1/36.abstract>.
- [Foygel and Drton, 2010] Foygel, R. and Drton, M. (2010). Extended Bayesian information criteria for Gaussian graphical models. In *Proceedings of Neural Information Processing Systems (NIPS 2010)*, pages 604–612, Vancouver, Canada.
- [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.
- [Fruchterman and Reingold, 1991] Fruchterman, T. and Reingold, B. (1991). Graph drawing by force-directed placement. *Software, Practice and Experience*, 21 :1129–1164.
- [Gonzàles et al., 2012] Gonzàles, I., Lê Cao, K., Davis, M., and Déjean, S. (2012). Visualising associations between paired 'omics data sets. *BioData Mining*, 5 :19.
- [Huynh-Thu et al., 2010] Huynh-Thu, V., Irrthum, A., Wehenkel, L., and Geurts, P. (2010). Inferring regulatory networks from expression data using tree-based methods. *PLoS ONE*, 5(9) :e12776.
- [Langfelder and Horvath, 2008] Langfelder, P. and Horvath, S. (2008). WGCNA : an R package for weighted correlation network analysis. *BMC Bioinformatics*, 9 :559.
- [Liu et al., 2010] Liu, H., Roeder, K., and Wasserman, L. (2010). Stability approach to regularization selection for high dimensional graphical models. In *Proceedings of Neural Information Processing Systems (NIPS 2010)*, pages 1432–1440, Vancouver, Canada.
- [Lysen, 2009] Lysen, S. (2009). *Permuted Inclusion Criterion : A Variable Selection Technique*. PhD thesis, University of Pennsylvania.
- [Martin et al., 2007] Martin, P., Guillou, H., Lasserre, F., Déjean, S., Lan, A., Pascussi, J., San Cristobal, M., Legrand, P., Besse, P., and Pineau, T. (2007). Novel aspects of PPAR α -mediated regulation of lipid and xenobiotic metabolism revealed through a multigenomic study. *Hepatology*, 54 :767–777.
- [Meinshausen and Bühlmann, 2006] Meinshausen, N. and Bühlmann, P. (2006). High dimensional graphs and variable selection with the lasso. *Annals of Statistics*, 34(3) :1436–1462.
- [Meyer et al., 2008] Meyer, P., Lafitte, F., and Bontempi, G. (2008). minet : A R/Bioconductor package for inferring large transcriptional networks using mutual information. *BMC Bioinformatics*, 9(461).
- [Mohan et al., 2012] Mohan, K., Chung, J., Han, S., Witten, D., Lee, S., and Fazel, M. (2012). Structured learning of Gaussian graphical models. In *Proceedings of NIPS (Neural Information Processing Systems) 2012*, Lake Tahoe, Nevada, USA.
- [Newman and Girvan, 2004] Newman, M. and Girvan, M. (2004). Finding and evaluating community structure in networks. *Physical Review E*, 69 :026113.
- [Noack and Rotta, 2009] Noack, A. and Rotta, R. (2009). Multi-level algorithms for modularity clustering. In *SEA 2009 : Proceedings of the 8th International Symposium on Experimental Algorithms*, pages 257–268, Berlin, Heidelberg. Springer-Verlag.
- [Opgen-Rhein and Strimmer, 2006] Opgen-Rhein, R. and Strimmer, K. (2006). Inferring gene dependency networks from genomic longitudinal data : a functional data approach. *REVSTAT*, 4 :53–65.
- [Pearl, 1998] Pearl, J. (1998). *Probabilistic reasoning in intelligent systems : networks of plausible inference*. Morgan Kaufmann, San Francisco, California, USA.
- [Pearl and Russel, 2002] Pearl, J. and Russel, S. (2002). *Bayesian Networks*. Bradford Books (MIT Press), Cambridge, Massachusetts, USA.

- [Reichardt and Bornholdt, 2006] Reichardt, J. and Bornholdt, S. (2006). Statistical mechanics of community detection. *Physical Review, E*, 74(016110).
- [Rossi and Villa-Vialaneix, 2010] Rossi, F. and Villa-Vialaneix, N. (2010). Optimizing an organized modularity measure for topographic graph clustering : a deterministic annealing approach. *Neurocomputing*, 73(7-9) :1142–1163.
- [Rossi and Villa-Vialaneix, 2011] Rossi, F. and Villa-Vialaneix, N. (2011). Représentation d’un grand réseau à partir d’une classification hiérarchique de ses sommets. *Journal de la Société Française de Statistique*, 152(3) :34–65.
- [Schaeffer, 2007] Schaeffer, S. (2007). Graph clustering. *Computer Science Review*, 1(1) :27–64.
- [Schäfer and Strimmer, 2005a] Schäfer, J. and Strimmer, K. (2005a). An empirical bayes approach to inferring large-scale gene association networks. *Bioinformatics*, 21(6) :754–764.
- [Schäfer and Strimmer, 2005b] Schäfer, J. and Strimmer, K. (2005b). A shrinkage approach to large-scale covariance matrix estimation and implication for functional genomics. *Statistical Applications in Genetics and Molecular Biology*, 4 :1–32.
- [Scutari, 2010] Scutari, M. (2010). Learning Bayesian networks with the bnlearn R package. *Journal of Statistical Software*, 35(3) :1–22.
- [Villa-Vialaneix et al., 2014] Villa-Vialaneix, N., Vignes, M., Viguerie, N., and San Cristobal, M. (2014). Inferring networks from multiple samples with consensus LASSO. *Quality Technology and Quantitative Management*, 11(1) :39–60.
- [von Luxburg, 2007] von Luxburg, U. (2007). A tutorial on spectral clustering. *Statistics and Computing*, 17(4) :395–416.
- [Zanghi et al., 2008] Zanghi, H., Ambroise, C., and Miele, V. (2008). Fast online graph clustering via erdős-rényi mixture. *Pattern Recognition*, 41 :3592–3599.