Introduction to the Cascade package and application to the GSE39411 dataset

Nicolas Jung, Frédéric Bertrand,
Seiamak Bahram, Laurent Vallat and Myriam Maumy-Bertrand

March 24, 2014

The Cascade package has two vignettes and a manual:

- Introduction to the Cascade package with application to the GSE39411 dataset, available thanks to the R-command: `vignette("Cascade")`
- Additional application of the Cascade package to E-MTAB-1475 dataset, available thanks to the R-command: `vignette("E-MTAB-1475_re-analysis")`
- The manual for the Cascade package is available thanks to the R-command: `vignette("Cascade-manual")`

Contents

1 Introduction

In a cell, after a specific activation, a gene contained in the DNA can be expressed as RNA molecules that are later traduced in proteins that will sustain the cell response (Crick *et al.*, 1970). Cells are in continuous contact with their environment within the organism and display an adapted response to its modifications (Barabási and Oltvai, 2004). For this, each transient environmental modification activates cell' surface receptors (and co-receptors) that induce multiple integrated signaling cascades whose ultimate events are expression of specific transcriptional factors (TF). These first TF induce the expression of other genes within the cell. Some of these genes code themselves for TF or transcriptional regulators (TR) that induce sequential activation of other genes. At the end, concerted expression of these multiple genes induces protein expressions that are the substratum of the adapted cellular reaction to the initial stimulus.

One Common tool to analyze such complex systems is regulatory networks (RN). When studying transcriptional data, this RN is called a gene regulatory network (GRN) in which the vertex represent genes and edges represent potential (orientated) interactions between these genes.

Since the emergence of high-throughput technologies that allow simultaneously measuring mRNA expression of thousands of genes, many tools have been developed to analyze and reverse engineer their underlying GRN (Bansal *et al.*, 2007; Hecker *et al.*, 2009; Bar-Joseph *et al.*, 2012). These methods should be splitted between static and time dependent methods. While the former relies on the assumption than co-expressed genes share some biological characteristics, the latter infers a directed network. In this last case, another important distinction should be made between temporal phenomenom induced by exogenous stimulus (e.g, stress response) or endogenous stimulus (*e.g.*, cell cycle) (Zhu *et al.*, 2007; Luscombe *et al.*, 2004; Yosef and Regev, 2011). These two stimulii result in different network topologies. Indeed, after an exogenous stimulus, networks topologies seem to have larger hubs and shorter paths leading to a quick response to external conditions (Luscombe *et al.*, 2004) and resulting in a cascade topology (Figure 1).

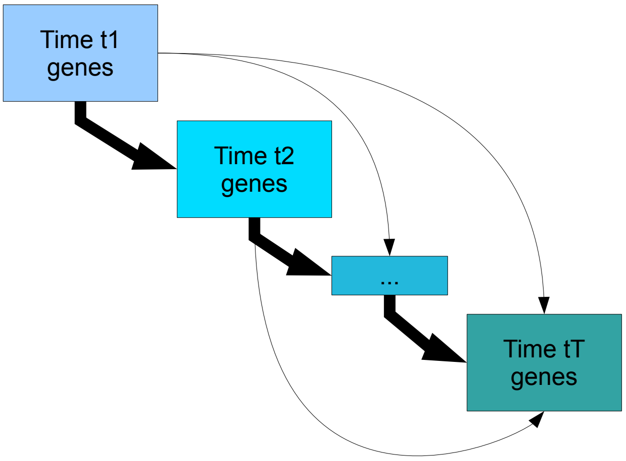

Figure 1: Cascade networks are temporal nested networks

The Cascade package is a tool dedicated to the analysis of microarray data and to the inference cascade networks. The statistical tools provided in this library are based on the methodology described by Vallat *et al.* (Vallat *et al.*, 2013) and contained several major improvements described here.

2 Installation requirements

Following software is required to run the Cascade package:

- R (> 2.14.2). For installation of R, refer to http://www.r-project.org.
- R-packages: abind ; animation ; cluster ; datasets ; graphics ; grDevices ; igraph ; lars ; lattice ; limma* ; magic ; methods ; nnls ; splines ; stats ; stats4 ; survival* ; tnet ; utils ; VGAM.

To install them :

- without stars:

  ```
  > install.packages("name_of_the_package")
  ```

- with one star:

  ```
  > source("http://bioconductor.org/biocLite.R")
  > biocLite("name_of_the_package")
  ```

Once the *Cascade* package is installed, you can load the package by:

```
> library(Cascade)
```

3 Data pre-processing

To illustrate our approach we will analyze a microarray dataset of the transcriptional response of healthy B-cells after B-cell receptor stimulation (Vallat *et al.*, 2007). Our dataset (part of GSE39411, (Vallat *et al.*, 2007)) is separated in two files: the first, micro_S, corresponds to the stimulated gene expressions while the second, micro_US, corresponds to the unstimulated gene expressions. In other words, micro_US is the control dataset. You can load these data by:

```
> data(micro_S)
> data(micro_US)
```

Each of the these dataset corresponds to 54613 genes measured through 4 time points and 6 subjects (we have repeated longitudinal data).
These data need to be coerced into a micro_array class. The matrix with the microarray measurements has to be of size $N \times K$ where N is the number of genes and $K = T \times P$ where T stands for the number of time points and P for the number of subjects. The first T columns are the gene expressions for subject 1, the following T are the gene expressions for subject 2... In our case:

```
> colnames(micro_S)
 [1] "N1_S_T60"  "N1_S_T90"  "N1_S_T210" "N1_S_T390"
 [5] "N2_S_T60"  "N2_S_T90"  "N2_S_T210" "N2_S_T390"
 [9] "N3_S_T60"  "N3_S_T90"  "N3_S_T210" "N3_S_T390"
[13] "N4_S_T60"  "N4_S_T90"  "N4_S_T210" "N4_S_T390"
[17] "N5_S_T60"  "N5_S_T90"  "N5_S_T210" "N5_S_T390"
[21] "N6_S_T60"  "N6_S_T90"  "N6_S_T210" "N6_S_T390"
```

To coerce the data toward a micro_array class, you may just use the as.micro_array function:

```
> micro_S<-as.micro_array(micro_S,time=c(60,90,210,390),subject=6)
> micro_US<-as.micro_array(micro_US,time=c(60,90,210,390),subject=6)
```

In addition of the matrix of microarray measurements, this class also contains the name of genes, their group, the first time at which they are expressed, the time points at which they are measured, and the number of subjects. Primarily, method `print` summarizes these informations:

```
> print(micro_S)

This is a micro_array S4 class. It contains :
 - (@microarray) a matrix of dimension  54613 * 24
          .... [gene expressions]
 - (@name) a vector of length  54613  .... [gene names]
 - (@group) a vector of length  1  .... [groups for genes]
 - (@start_time) a vector of length  1
          .... [first differential expression for genes]
 - (@time)a vector of length  4  .... [time points]
 - (@subject) an integer  .... [number of subject]
```

While method `print` gives the structure of the object, method `head` gives an overview of the data:

```
> head(micro_S)

The matrix :

          N1_S_T60 N1_S_T90 N1_S_T210
1007_s_at    136.1    116.6     127.6
1053_at       32.0     43.3      31.3
117_at        78.0     63.5      57.9
121_at       201.8    209.2     208.8
1255_g_at     16.3      8.0      15.8
1294_at      196.8    198.7     163.9
...

Vector of names :
[1] "1007_s_at" "1053_at"   "117_at"    "121_at"
[5] "1255_g_at" "1294_at"
...
Vector of group :
[1] 0
...
Vector of starting time :
[1] 0
...
Vector of time :
[1]  60  90 210 390

Number of subject :
[1] 6
```

Entries `Vector of group` and `Vector of starting time` are set to 0 because they are no yet defined. They will be completed automatically when using gene selection functions of this package. Otherwise, it should be completed by the user.

Once the data are coerced into the `micro_array` class, this package allows doing gene selection and reverse-engineering of the network.

4 Gene selection

The selection step requires at least two sets of data. The selection function will select genes differentially expressed in one condition compared with the other. If only one experimental condition is provided (e.g., unstimulated control data omitted), it will be compared to a flat and null pattern.

In this package gene selection mainly relies on the R-bioconductor `limma` package (Smyth, 2005). The `limma` package allows selecting genes that are differentially expressed between two conditions. In our case, these two conditions are "*stimulated*" and "*unstimulated*". The method relies on linear models and on improved bayesian t-tests (Smyth, 2005). Basically, to find the 50 more significant expressed genes you will use:

```
> Selection<-geneSelection(x=micro_S,y=micro_US,
  tot.number=50,data_log=TRUE)
```

The `data_log` option (default to TRUE) indicates that the data are logged before analysis. This function returns an object of class `micro_array`, with the difference "stimulated" (S) minus "unstimulated" (US) of the 50 more significant expressed genes ; as the `data_log` option is here activated, we get:

$$\log(S) - \log(US) = \log\left(\frac{S}{US}\right).$$

Notice that the `group` and `start_time` are filled out automatically.

Applying the `summary` method prints the structure of Pearson linear correlation for subjects (see Figure 2) and the structure of Pearson linear correlation for genes (see Figure 3):

```
> summary(Selection)
```

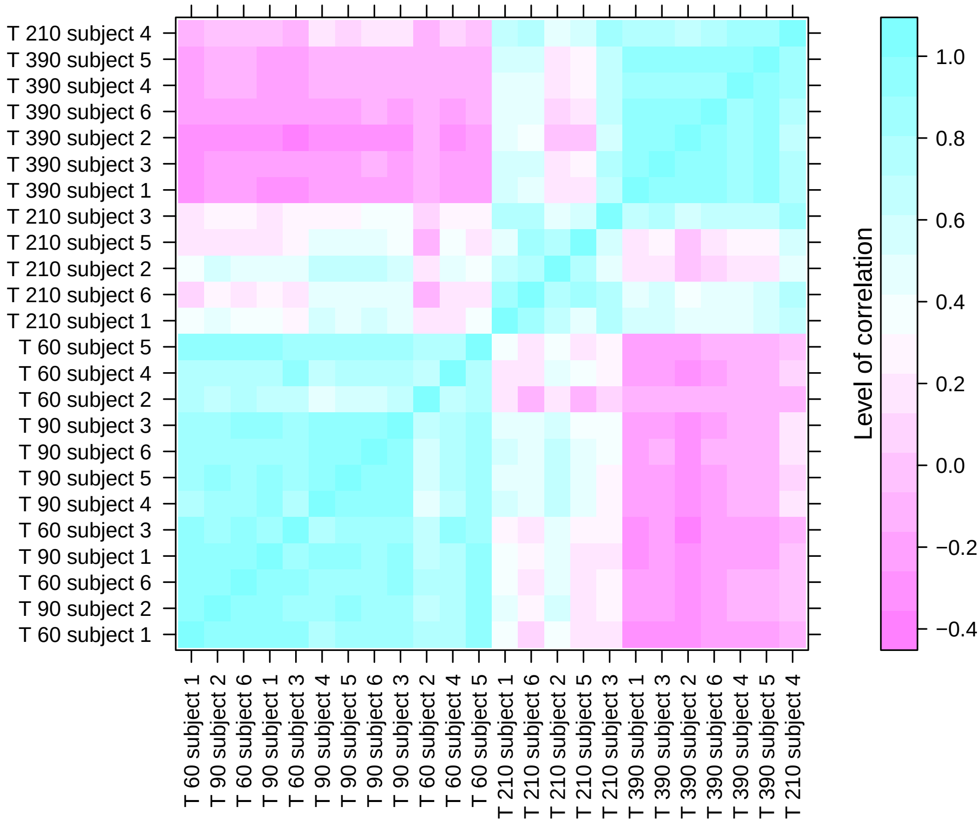

Figure 2: Correlation between subjects

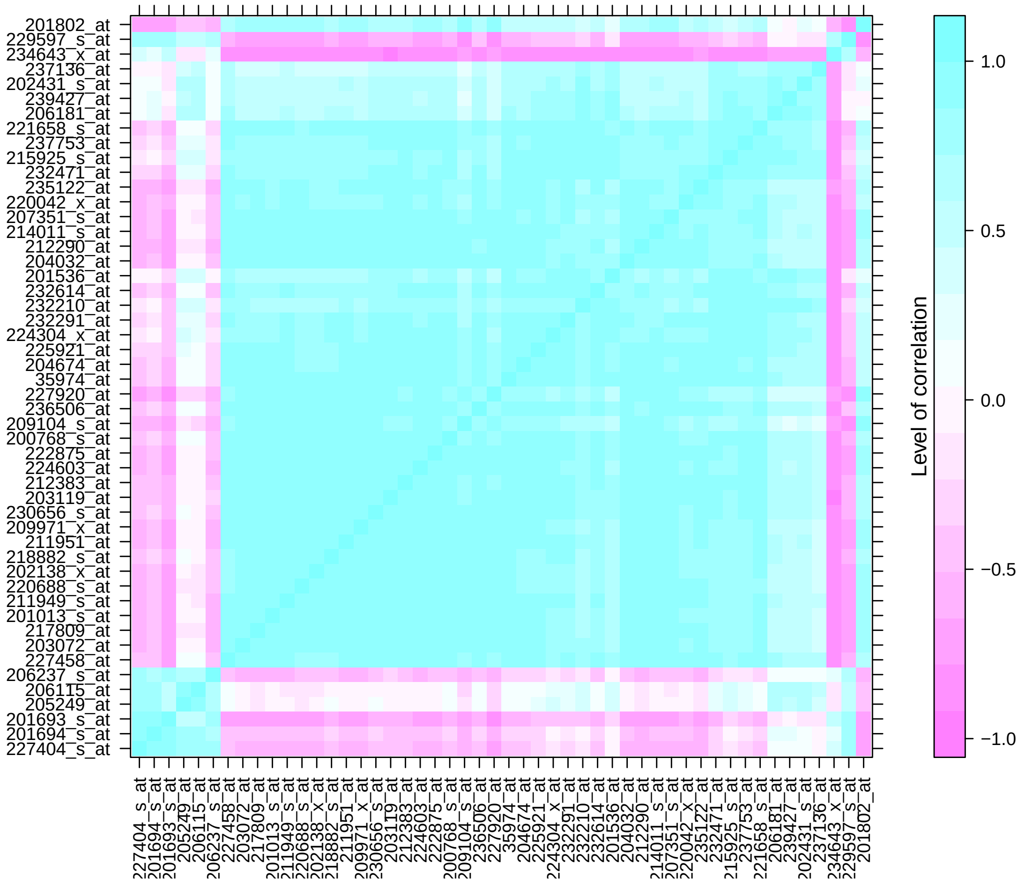

Figure 3: Correlation between genes

Note that a hierarchical clustering (function agnes of package cluster) is performed before plotting the result. This allows pointing out some structures, as correlated objects will be close in the graph.

If we want to select genes that are differentially expressed at specific time points we use the option wanted.patterns:

```
> #If we want to select genes that are differentially
> #at time t60 or t90 :
> Selection<-geneSelection(x=micro_S,y=micro_US,tot.number=30,
  wanted.patterns=
  rbind(c(0,1,0,0),c(1,0,0,0),c(1,1,0,0)))
```

You may want forbid some patterns thanks to the forbidden.patterns option.

If we wish select genes that have a differential maximum of expression at a specific time point, we may use the genePeakSelection method. Basically, this function selects genes that are differentially expressed at desired time point, and which differential expression is significantly higher at this time point:

```
> Selection<-genePeakSelection(x=micro_S,y=micro_US,1,
  abs_val=FALSE,alpha_diff=0.01)
```

If there are more than two microarrays of interest, geneSelection may be used with a list of microarrays as first argument, and a list specifying the contrast as a second argument:

First element: “condition”, “condition&time” or “pattern”. The “condition” specification is used when the overall goal is to compare two conditions. The “condition&time” specification is used when comparing two conditions at two precise time points. The “pattern” specification allows to choose at which time points selected a gene should be expressed or not.

Second element: a vector of length 2, corresponding to the two conditions that should be compared. If a non-temporal dataset is used as control, it should be the first element of the micro_array list and the option “cont=TRUE” should be used.

Third element: depends on the first element. This element is not needed if “condition” has been specified. If “condition&time” has been specified, then this is a vector containing the time point at which the comparison should be done. If “pattern” has been specified, then this is a vector of 0 and 1 of length T, where T is the number of time points. Time points where differential expression is wanted are provided with 1.

We can now compute an effective selection. As shown in Figure 4, the early time points (t_1 =60 and t_2 =90) are correlated together and the later time points (t_3 =210 and t_4 =390) are correlated together; this is a fact that is well known in the literature (Yosef and Regev, 2011).

As an illustrating example, the following selection will be used for reverse-engineering:

```
> #Genes with differential expression at t1
> Selection1<-geneSelection(x=micro_S,y=micro_US,20,wanted.patterns= rbind(c(1,0,0,0)))
> #Genes with differential expression at t2
> Selection2<-geneSelection(x=micro_S,y=micro_US,20,wanted.patterns= rbind(c(0,1,0,0)))
> #Genes with differential expression at t3
> Selection3<-geneSelection(x=micro_S,y=micro_US,20,wanted.patterns= rbind(c(0,0,1,0)))
> #Genes with differential expression at t4
> Selection4<-geneSelection(x=micro_S,y=micro_US,20,wanted.patterns= rbind(c(0,0,0,1)))
> #Genes with global differential expression
> Selection5<-geneSelection(x=micro_S,y=micro_US,20)
```

We then make the union between these different selections:

```
> Selection<-unionMicro(list(Selection1,Selection2,Selection3,Selection4,Selection5))
> print(Selection)
```

```
This is a micro_array S4 class. It contains :
 - (@microarray) a matrix of dimension  74 * 24
          .... [gene expressions]
 - (@name) a vector of length  74  .... [gene names]
 - (@group) a vector of length  74  .... [groups for genes]
 - (@start_time) a vector of length  74
          .... [first differential expression for genes]
 - (@time)a vector of length  4  .... [time points]
 - (@subject) an integer  .... [number of subject]
```

We use the `org.Hs.eg.db` Bioconductor database to match probesets with gene ID:

```
> library(org.Hs.eg.db)
> ff<-function(x){substr(x, 1, nchar(x)-3)}
> ff<-Vectorize(ff)
> #Here is the function to transform the probeset names to gene ID.
>
> library("hgu133plus2.db")
> probe_to_id<-function(n){
   x <- hgu133plus2SYMBOL
   mp<-mappedkeys(x)
   xx <- unlist(as.list(x[mp]))
   genes_all = xx[(n)]
   genes_all[is.na(genes_all)]<-"unknown"
   return(genes_all)
   }
> Selection@name<-probe_to_id(Selection@name)
> #Prints the correlation graphics Figure 4:
> summary(Selection,3)
```

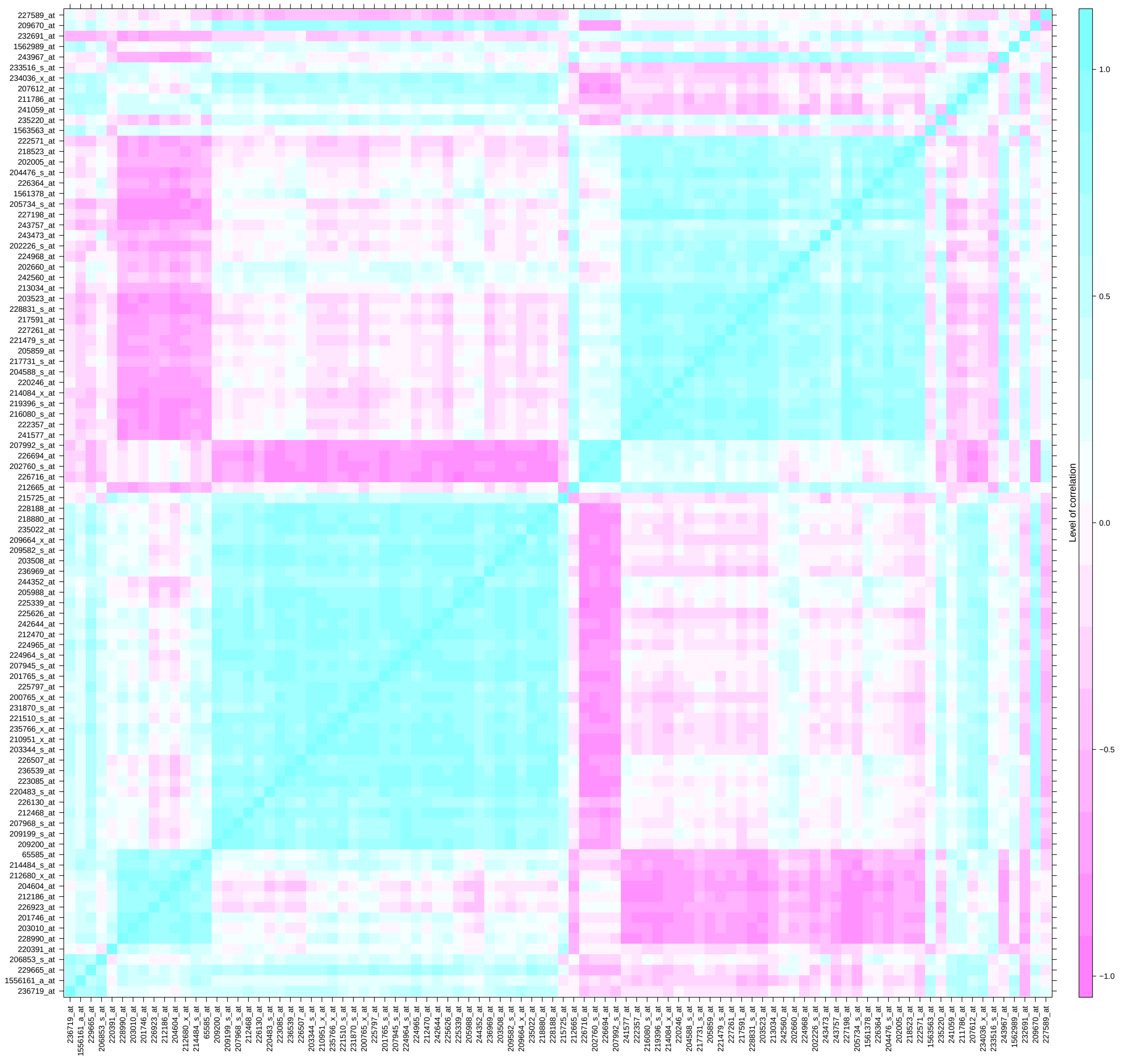

Figure 4: Correlation structure of the final selection

5 Gene regulatory network reverse-engineering

5.1 Theoretical background

Our gene regulatory network reverse-engineering method relies on a Lasso penalized estimation of a linear regression model (Tibshirani, 1996). Before describing our model, we make some general reminders of the Lasso estimator.

5.1.1 The Lasso estimate

Suppose that we have data $(\boldsymbol{x_{i.}}, y_i)_{i=1,\cdots,N}$ where the $\boldsymbol{x_{i.}} = (x_{i1}, \cdots, x_{ip})^T$ are the predictors while the y_i are the response. The linear regression model is:

$$y_i = \sum_{j=1}^{p} \beta_j x_{ij} + \eta_i, \tag{1}$$

where η_i is a noise following some probabilistic distribution.

Assume that the predictors are standardized and that the response is centered. The Lasso estimate is then given by:

$$\hat{\boldsymbol{\beta}}^L(\lambda) = \underset{\boldsymbol{\beta}\in\mathbb{R}^p}{\operatorname{argmin}} \left[\sum_{i=1}^{N} \left(y_i - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 + \lambda \|\boldsymbol{\beta}\|_1 \right], \tag{2}$$

with λ a non-negative scalar that determines the level of the constraints which is user-provided. We remark that:

- When $\lambda = 0$, $\hat{\boldsymbol{\beta}}^L$ is an ordinary least square estimation.
- When $\lambda = +\infty$, we get $\hat{\boldsymbol{\beta}}^L = \mathbf{0}_p$.

The Lasso estimate for linear regression has two main advantages:

1. it allows dealing with ill-posed problems where the number of observations is inferior to the number of variables,
2. it allows performing variable selection: for a proper choice of λ, $\hat{\boldsymbol{\beta}}^L(\lambda)$ will be parsimonious.

The Lasso estimate for linear regression can also be written in the following form:

$$\hat{\boldsymbol{\beta}}^L(\lambda) = \underset{\boldsymbol{\beta}\in\mathbb{R}^p \quad \|\boldsymbol{\beta}\|_1 \leqslant \tilde{\lambda}}{\operatorname{argmin}} \left[\sum_{i=1}^{N} \left(y_i - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 \right]. \tag{3}$$

These two formulations (equation (2) which is the penalized formulation and equation (3) which is the constrained formulation) are equivalent in the sense that for each non negative λ there is a non-negative $\tilde{\lambda}$ leading to the same solution.

5.1.2 Model for network reverse-engineering

Suppose that we have selected N genes across T time points and for P individuals; we note x_{npt} the expression of gene n for individual p at time-point t. Since each gene will be considered exactly once as a response variable, our model is composed of N linear regression models. As the action of a gene of another is not instantaneous, we define:

$$\tilde{\boldsymbol{x}}_{np.} = \begin{pmatrix} x_{npt_2} \\ \vdots \\ x_{npt_T} \end{pmatrix} \quad \text{and} \quad \check{\boldsymbol{x}}_{np.} = \begin{pmatrix} x_{npt_1} \\ \vdots \\ x_{npt_{T-1}} \end{pmatrix},$$

$$\tilde{\boldsymbol{x}}_{n..} = \begin{pmatrix} \tilde{\boldsymbol{x}}_{n1.} \\ \vdots \\ \tilde{\boldsymbol{x}}_{nP.} \end{pmatrix} \quad \text{and} \quad \check{\boldsymbol{x}}_{n..} = \begin{pmatrix} \check{\boldsymbol{x}}_{n1.} \\ \vdots \\ \check{\boldsymbol{x}}_{nP.} \end{pmatrix}.$$

We note that $\tilde{\boldsymbol{x}}_{np.}$ begins at time point t_2 and ends at time point t_T, while $\check{\boldsymbol{x}}_{np.}$ begins at time point t_1 and ends at time point t_{T-1}. In the following, when gene n is the response variable we will use $\tilde{\boldsymbol{x}}_{np.}$, and $\check{\boldsymbol{x}}_{np.}$ when gene n is a predictor variable.

We further assume that each gene has been assigned to one and only one of the T time-cluster (one cluster for each time).

We have previously proposed (Vallat *et al.*, 2013) the follwing linear regression model:

$$\tilde{\boldsymbol{x}}_{n..} = \sum_{n'=1}^{N} \boldsymbol{F}_{m(n')m(n)} \omega_{n'n} \check{\boldsymbol{x}}_{n'..} + \boldsymbol{\varepsilon}_n,$$

where:

- $m(\bullet)$ is the function that maps a gene to its time-cluster,
- $\boldsymbol{F}_{m(n')m(n)}$ is a $T-1$ square matrix that describes the action of genes,
- $\omega_{n'n}$ is the strength of the connection from gene i toward gene j,
- $\boldsymbol{\varepsilon}$ is a noise vector of length $T-1$ with $\mathbb{E}(\boldsymbol{\varepsilon}) = 0$ and $\text{var}(\boldsymbol{\varepsilon}) = \sigma^2$

We choose to use a Lasso estimate for our linear regression model:

$$(\hat{\boldsymbol{\omega}}, \hat{\boldsymbol{F}}) = \underset{\substack{\omega_{n'n} \in \mathbb{R},\ 1 \leqslant n', n \leqslant N \\ \boldsymbol{F}_{ab} \in \mathcal{M}_{T-1}(\mathbb{R}), 1 \leqslant a,b \leqslant T}}{\text{argmin}} \left[\sum_{n=1}^{N} \left(\tilde{\boldsymbol{x}}_{n..} - \sum_{n'=1}^{N} \boldsymbol{F}_{m(n')m(n)} \omega_{n'n} \check{\boldsymbol{x}}_{n'..} \right)^2 \right],$$

with the constraint:

$$\forall n = 1, ..., N, \quad \sum_{n'=1}^{N} \omega_{n'n} \leqslant \lambda_n.$$

So, $\tilde{\boldsymbol{x}}_{n..}$ is the regulated gene and $\boldsymbol{x}_{n'..}, n' = 1, \cdots, N$ are the regulators. Notice that matrix $F_{m(n')m(n)}$ permits to the link between genes n' and n to evolve across time. To enforce temporal causality we need the two following time constraints:

1. $m(n') \geqslant m(n) \Rightarrow \boldsymbol{F}_{m(n')m(n)} = 0$: this ensures that a gene with temporal cluster t_k can influence a gene with temporal cluster $t_{k'}$ if and only if $k < k'$,

2. the matrices $\boldsymbol{F}$ are lower triangular matrices: this ensures that the expression of a gene at time t_k can influence another gene at time $t_{k'}$ if and only if $k < k'$.

Sub-diagonals and the diagonal of matrices F are supposed to be invariant (Vallat *et al.*, 2013). Consequently, interactions depend only on time index differences rather than absolute time index.

We solve this problem with a coordinate ascent approach, by iteratively supposing the $\boldsymbol{F}$ matrices or the $\omega_{n'n}$ matrices known. The result of the optimization is a connectivity network described by the nonzero elements of $\hat{\omega}_{n'n}(obs)$ combined with a set of cluster-dependent interaction models described by the set $\hat{\boldsymbol{F}}_{m(n')m(n)}(obs)$.

However, if clusters are sufficiently homogeneous, inference of matrices $\boldsymbol{F}_{m(n')m(n)}$ doesn't depend on which genes are active (*i.e.* which $\omega_{n'n} \neq 0$). That's why a non iterative algorithm is proposed in which estimation of of matrices $F_{m(i)m(j)}$ precedes estimation of matrix $\boldsymbol{\Omega}$.

To get a more robust result, at each step, the estimation of matrices $\boldsymbol{F}_{m(n')m(n)}$ is done several times throughout cross-validation. Furthermore, to avoid computational issues, the new solution is chosen by a linear combination between the old and the new solution.

5.2 Performing the reverse-engineering algorithm

To perform this algorithm on our data:

```
> network<-inference(Selection)
```

```
We are at step :  1
The convergence of the network is (L1 norm) : 0.01096
We are at step :  2
The convergence of the network is (L1 norm) : 0.00302
We are at step :  3
The convergence of the network is (L1 norm) : 0.00217
We are at step :  4
The convergence of the network is (L1 norm) : 0.00177
We are at step :  5
The convergence of the network is (L1 norm) : 0.00146
We are at step :  6
The convergence of the network is (L1 norm) : 0.00111
We are at step :  7
The convergence of the network is (L1 norm) : 0.00089
```

We can plot a representation of F matrices (Figure 5) and the resulting network (Figure 6) by simply using the `plot` method:

```
> plot(network,choice="F")
> plot(network,choice="network",gr=Selection@group,label_v=Selection@name)
```

Note that all network plots are computed using the Igraph R package (Csardi and Nepusz, 2006).

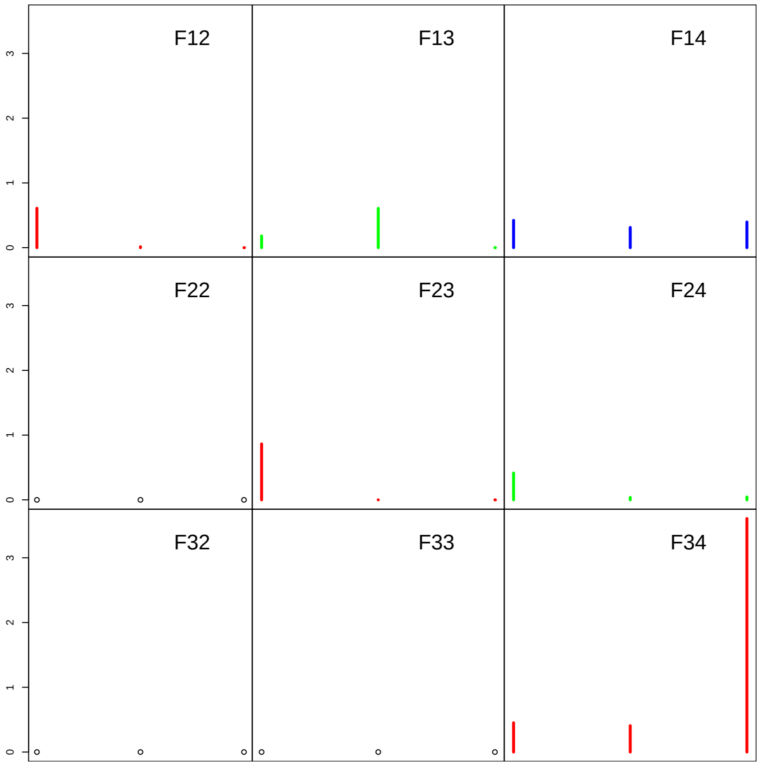

Figure 5: The F matrices ; for each matrix, the first bar plot corresponds to the coefficient of the diagonal, the second to the first sub-diagonal...

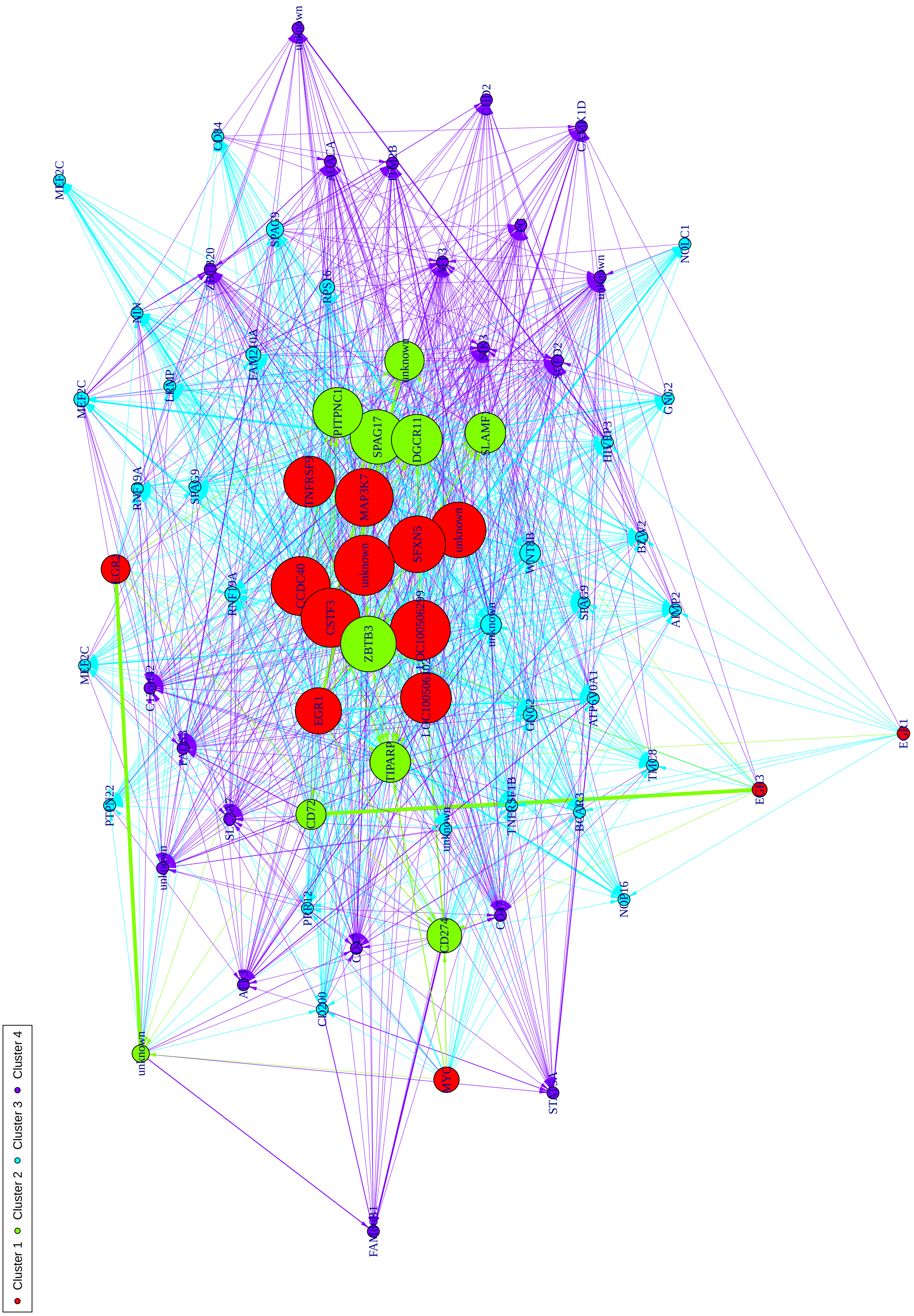

Figure 6: The resulting network with all edges

The number of edges in the network makes the message difficult to interpret ; and as we will see in the next section, results in term of predictive positive value and F-score can be improved when choosing a right cutoff level. Using the nv option, we will choose a cutoff under which the regression coefficient estimates ($\hat{\omega_{ij}}(obs)$) are set to 0. In Figure 7 a cutoff of 0.11 is chosen.

5.3 Choosing the best cutoff for edge minimal strength

The difficulty is now to choose the best cutoff. As a starting point, we propose method evolution, that allows the user to see, in a html page, the evolution of the network when the cutoff is growing up. When the fix option is set to FALSE, at each step the position of the genes are re-calculated.

```
> evolution(network,seq(0,0.4,by=0.01),gr=Selection@group,
   fix=TRUE,label_v=Selection@name)
> evolution(network,seq(0,0.4,by=0.01),gr=Selection@group,
   fix=FALSE,label_v=Selection@name)
```

To see the result of these functions, go to :

- http://www-irma.u-strasbg.fr/~njung/evolution_fix_true/evol.html : here the fix option is set to TRUE.
- http://www-irma.u-strasbg.fr/~njung/evolution_fix_false/evol.html: here the fix option is set to FALSE.

As it is mostly accepted, gene regulatory networks are supposed to be scale-free (Jeong *et al.*, 2000). The notion of scale freeness in networks relies on the probability distribution of the number of outgoing edges. A network is called scale free when this distribution is a power law distribution (Clauset *et al.*, 2009). As this family of law is large, it is difficult to test such an hypothesis. We used the test proposed by Clauset *et al.*(Clauset *et al.*, 2009):

```
> #To be computed:
> #evol_cutoff<-cutoff(network)
> nv<-0.15
```

We plot here the smooth interpolation rather than the exact values, as our interest relies mostly on the trend (Figure 8). We propose a choice of cutoff that relies on two criteria:

- the p-value should be greater than 0.10: in this case, the scale-freeness of the network is reliable (Clauset *et al.*, 2009).
- we determined by simulation the best area of choice (on the plot (Figure 8)).

Based on these two criteria, we choose a cutoff of $nv = 0.11$.

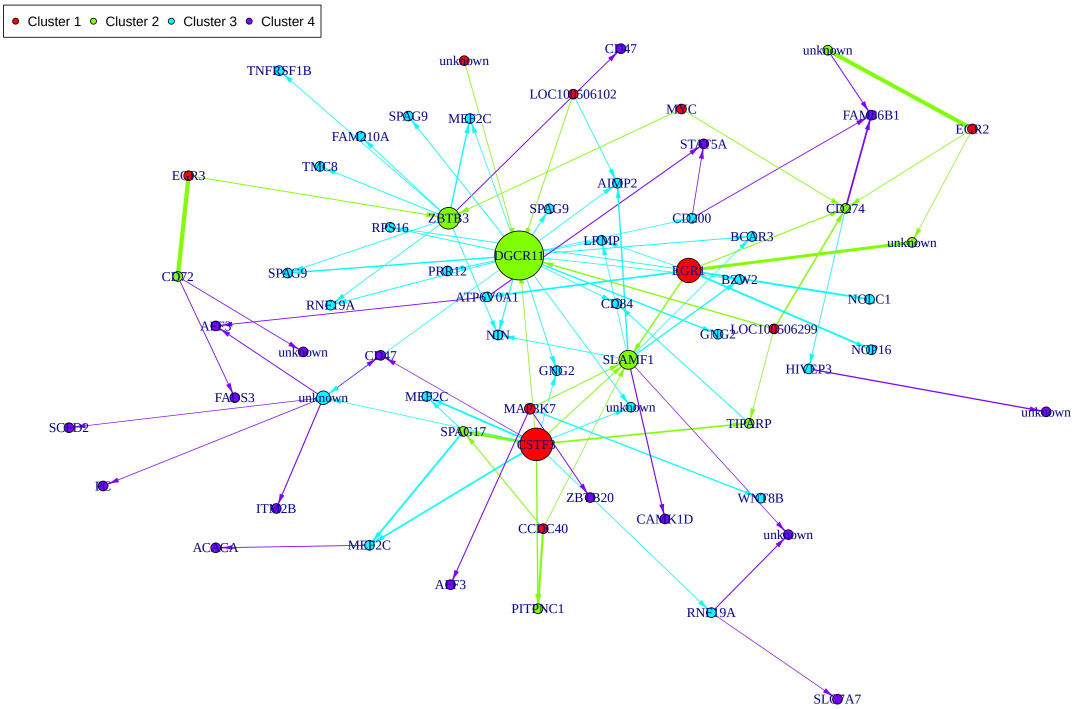

Figure 7: The resulting network with a cutoff of 0.15

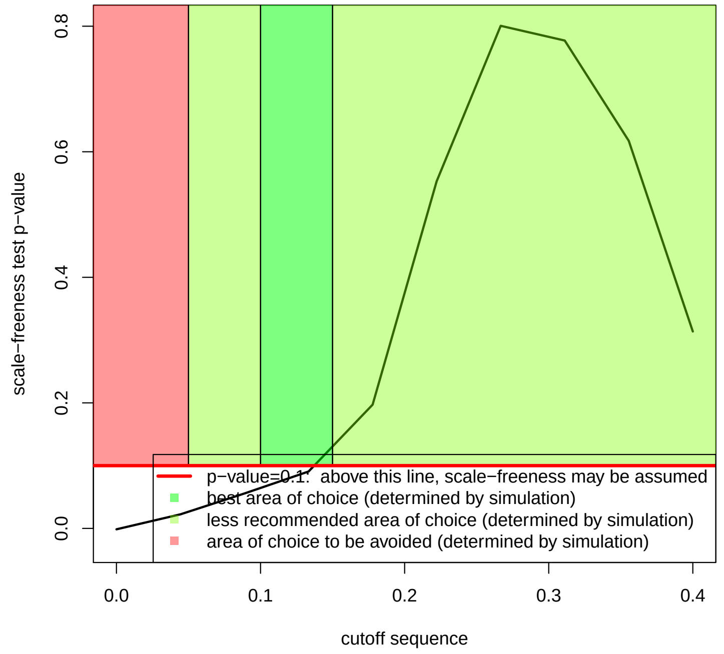

Figure 8: Evolution of scale-freeness of the network in function of the cutoff. The p-value corresponds to the adequacy of the data to a power law distribution.

5.4 Analyzing the network

One may want to know which genes are important in the network. In our representation, the bigger the vertex the larger the number of outgoing edges. Indeed, genes with many outgoing edges, the hubs, are important in the network. But genes controlling these hubs should be considered with attention. The `analyze_network` method allows computing different indicators:

- betweenness : it is a measure of the node centrality. It is calculated, for node n, by the following formula:

$$\sum_{s \neq t \neq n} \frac{\sigma_{st}(n)}{\sigma_{st}}$$

  where σ_{st} is the number of shortest ways between s and t, and $\sigma_{st}(n)$ is the number of shortest ways between s and t passing by n ;

- degree : the number of outgoing edges ;

- output : the sum of weights of outgoing genes ;

- closeness : it is a measure of the distance (in terms of shortest path) of a gene to others.

As our network is weighted we used specific measures developed by Opsahl (Opsahl, 2009).

```
> analyze<-analyze_network(network,nv,Selection@name)
> head(analyze)
          node betweenness degree    output closeness
1 LOC100506299           0      3 0.8133348 14.841838
2       CCDC40           0      3 0.8884602  7.305208
3      unknown           0      1 0.1749376  8.826222
4 LOC100506102           0      2 0.3661906  9.622533
5      TNFRSF9           0      0 0.0000000  0.000000
6        CSTF3           0     12 3.4345058 23.065564
```

Note that one can plot the network and modulate the size of the vertex following one of this measure, using the `weight.node` option.

Using again the package `animation`, we can see how the signal spreads in the network by turning to `TRUE` the option `ani`:

```
> plot(network,nv=nv,gr=Selection@group,ani=TRUE,label_v=Selection@name,
  edge.arrow.size=0.9,edge.thickness=1.5)
```

Result is available at `http://www-irma.u-strasbg.fr/~njung/network_spread/spread.html`.

The method `plot` has basically two steps:

1. it calculates the position of the vertex,
2. it plots the graph.

In some case, it is interesting to produce two plots of a same network without changing vertex positions. Here is a way to do that, using the `ini` option of method plot:

```
> P<-position(network,nv=nv)
> #plotting the network with the given position
> plot(network,nv=nv,gr=Selection@group,ini=P,label_v=Selection@name)
```

However, we didn't develop all possibilities of the `plot` option ; for more possibilities, please refer to the manual:

```
> vignette("Cascade-manual")
```

6 Prediction of gene expression modulations after a knock-out experiment

Once the network has been reverse-engineered, we want to know the impact of an experimental perturbation in this network. For example, what would happen if expression of EGR1 is knocked-out ?

```
> EGR1<-which(Selection@name %in% "EGR1")
```

First the `geneNeighborhood` method allows determining which are the neighborhood of EGR1 (see Figure 9).

```
> geneNeighborhood(network,targets=EGR1,nv=nv,ini=P,
   label_v=Selection@name)
> #label.hub: only hubs vertex should have a name
> #label_v: name of the vertex
```

We predict gene expression modulations within the network if EGR1 is experimentaly knocked-out.

```
> prediction_ko5<-predict(Selection,network,nv=nv,targets= EGR1)
```

Then we plot the results (Figure 10):

```
> #We plot the results.
> #Here for example we see changes at time point t2:
> plot(prediction_ko5,time=2,ini=P,label_v=Selection@name)
```

7 Simulation

To simulate gene expressions based on a gene regulatory network, we first have to generate the network. Here, we implemented an algorithm that is inspired by the *preferential attachment* from Barabási (Barabási, 2003; Jeong *et al.*, 2007). We adapted this algorithm in our case of temporal cascade networks.

We then use our linear model to make some simulations, using Laplace laws to initiate the algorithm.

```
> #We set the seed to make the results reproducible
> set.seed(1)
> #We create a random scale free network
> Net<-network_random(
          nb=100,
          time_label=rep(1:4,each=25),
          exp=1,
          init=1,
          regul=round(rexp(100,1))+1,
          min_expr=0.1,
          max_expr=2,
          casc.level=0.4
          )
> #We change F matrices
> T<-4
> F<-array(0,c(T-1,T-1,T*(T-1)/2))
```

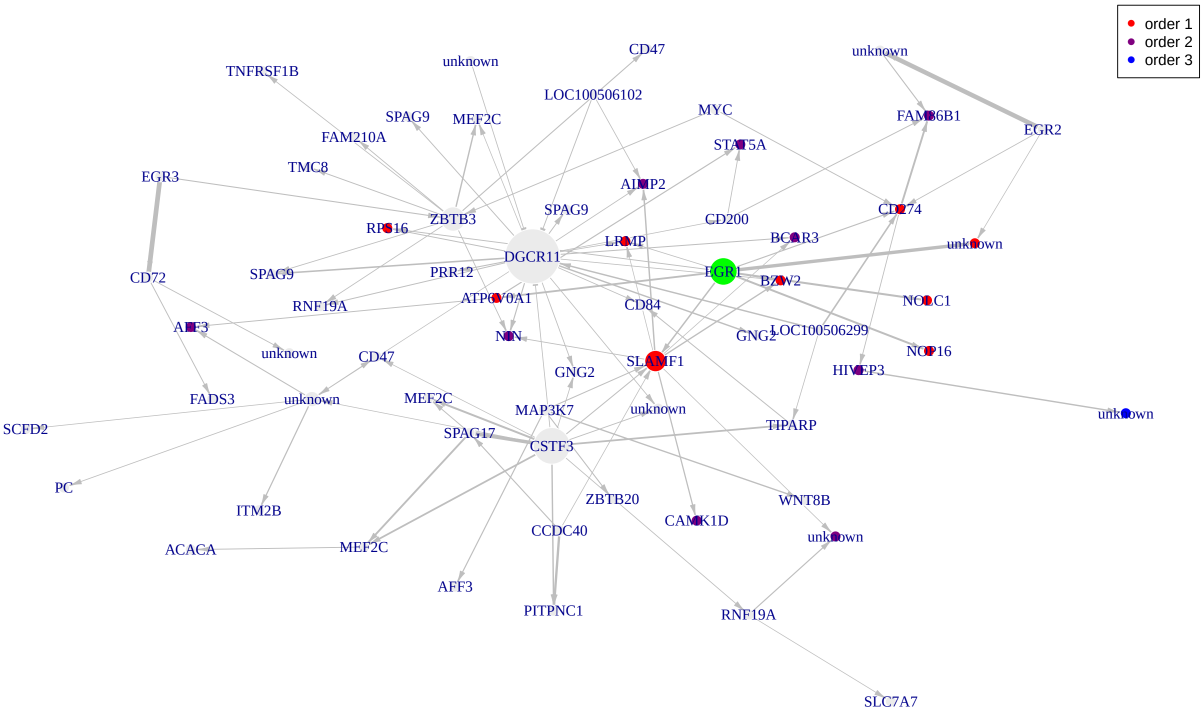

Figure 9: Neighborhood of gene EGR1

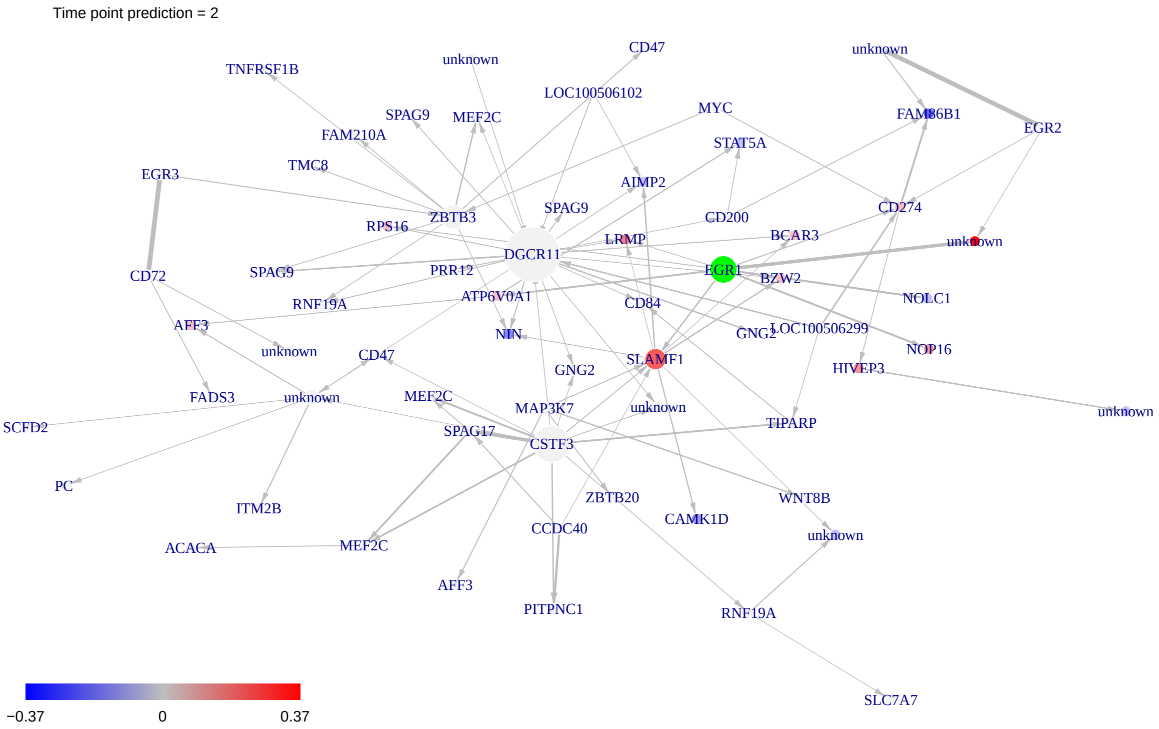

Figure 10: Perturbation modulation at time point 2 of the network consecutively to the knock-out of EGR1.

```
> for(i in 1:(T*(T-1)/2)){diag(F[,,i])<-1}
> F[,,2]<-F[,,2]*0.2
> F[2,1,2]<-1
> F[3,2,2]<-1
> F[,,4]<-F[,,2]*0.3
> F[3,1,4]<-1
> F[,,5]<-F[,,2]
> Net@F<-F
> #We simulate gene expression according to the network Net
> M<-gene_expr_simulation(
         network=Net,
         time_label=rep(1:4,each=25),
         subject=5,
         level_pic=200)


> #We infer the new network
> Net_inf<-inference(M)


> #Comparing true and inferred networks
> F_score<-rep(0,200)
> #Here are the cutoff level tested
> test.seq<-seq(0,max(abs(Net_inf@network*0.9)),length.out=200)
> u<-0
> for(i in test.seq){
         u<-u+1
         F_score[u]<-compare(Net,Net_inf,i)[3]
  }


> #Choosing the cutoff
> cut.seq<-cutoff(Net_inf)
> points(0.125,0.1199,col="red",pch=16,cex=2)
> #Corresponding Fscore evolution
> plot(test.seq,F_score,type="l",xlab="cutoff",ylab="Fscore")
> abline(v=0.125,col="red")
```

Figure 11 shows the evolution of the p-value of the scale-freeness test while Figure 12 shows the corresponding evolution of the F-score. As shown, choosing the best cut-off allows a dramatic increase of the cut-off.

Figure 13 show the evolution of the F-score when the number of individuals increase.

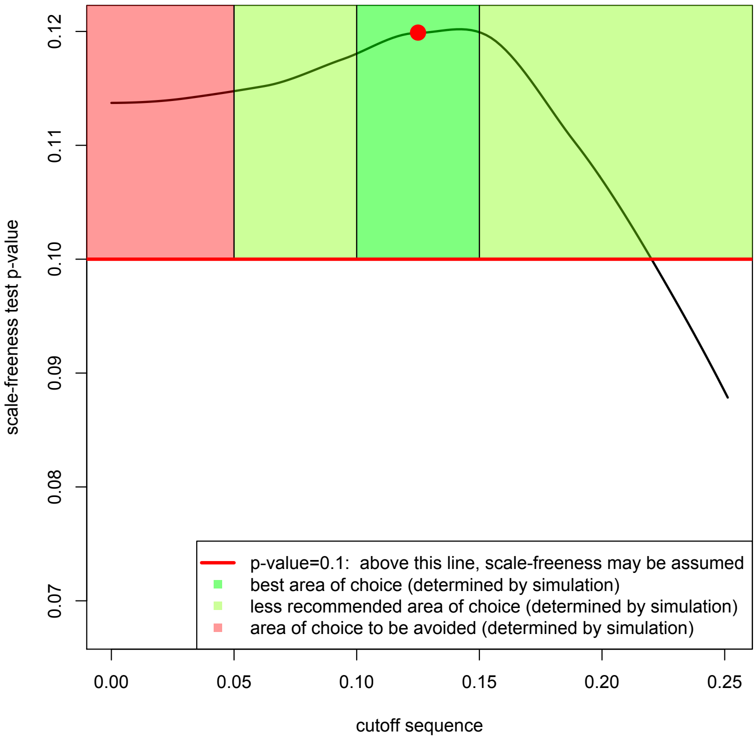

Figure 11: Evolution of the scale-freeness of the network in function of the cutoff

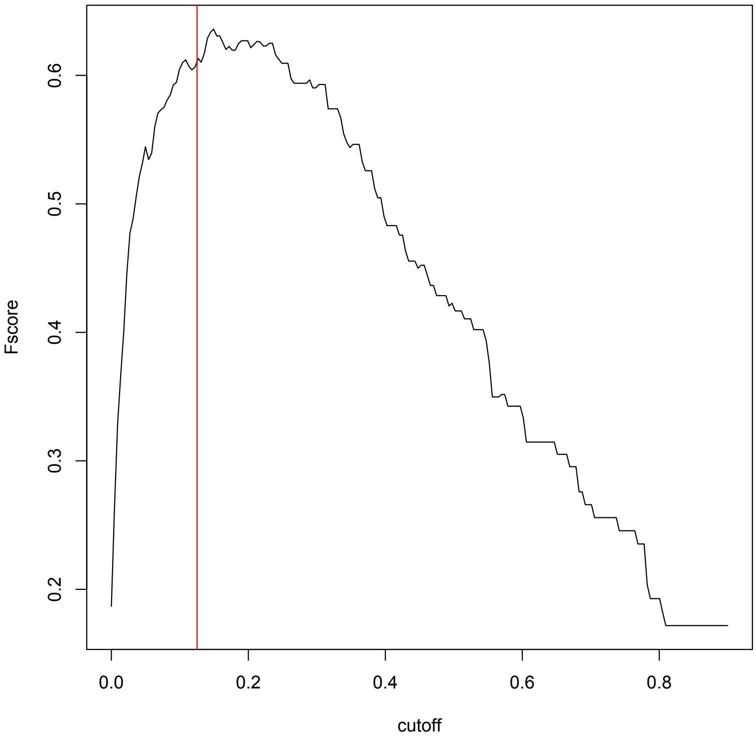

Figure 12: Evolution of F-score in function of the cutoff

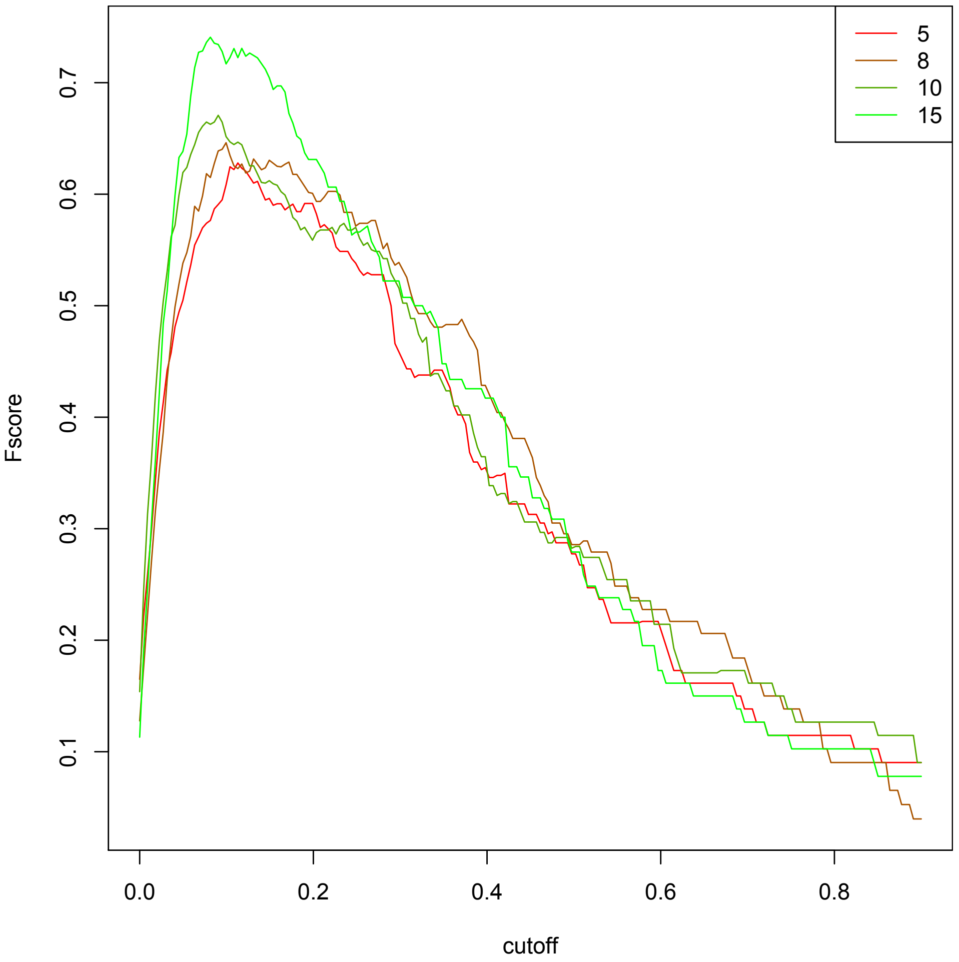

Figure 13: Evolution of F-score in function of the cutoff and the number of subject in the study

References

Bansal, M., Belcastro, V., Ambesi-Impiombato, A., and Di Bernardo, D. (2007). How to infer gene networks from expression profiles. *Molecular systems biology*, 3(1).

Bar-Joseph, Z., Gitter, A., and Simon, I. (2012). Studying and modelling dynamic biological processes using time-series gene expression data. *Nature Reviews Genetics*, 13(8):552–564.

Barabási, A.-L. (2003). Emergence of scaling in complex networks. In Bornholdt, S. and Schuster, H. G., editors, *Handbook of graphs and networks: from the genome to the internet*, pages 69–84. Wiley-VCH, Weinheim.

Barabási, A.-L. and Oltvai, Z. N. (2004). Network biology: understanding the cell's functional organization. *Nature Reviews Genetics*, 5(2):101–113.

Clauset, A., Shalizi, C. R., and Newman, M. E. (2009). Power-law distributions in empirical data. *SIAM review*, 51(4):661–703.

Crick, F. *et al.* (1970). Central dogma of molecular biology. *Nature*, 227(5258):561–563.

Csardi, G. and Nepusz, T. (2006). The igraph software package for complex network research. *InterJournal*, Complex Systems:1695.

Hecker, M., Lambeck, S., Toepfer, S., Van Someren, E., and Guthke, R. (2009). Gene regulatory network inference: data integration in dynamic models?a review. *Biosystems*, 96(1):86–103.

Jeong, H., Néda, Z., and Barabási, A.-L. (2007). Measuring preferential attachment in evolving networks. *EPL (Europhysics Letters)*, 61(4):567.

Jeong, H., Tombor, B., Albert, R., Oltvai, Z. N., and Barabási, A.-L. (2000). The large-scale organization of metabolic networks. *Nature*, 407(6804):651–654.

Luscombe, N. M., Babu, M. M., Yu, H., Snyder, M., Teichmann, S. A., and Gerstein, M. (2004). Genomic analysis of regulatory network dynamics reveals large topological changes. *Nature*, 431(7006):308–312.

Opsahl, T. (2009). *Structure and Evolution of Weighted Networks.* University of London (Queen Mary College), London, UK.

Smyth, G. K. (2005). Limma: linear models for microarray data. In Gentleman, R., Carey, V., Dudoit, S., Irizarry, R., and Huber, W., editors, *Bioinformatics and Computational Biology Solutions using R and Bioconductor*, pages 397–420. Springer, New York.

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

Vallat, L., Kemper, C. A., Jung, N., Maumy-Bertrand, M., Bertrand, F., Meyer, N., Pocheville, A., Fisher, J. W., Gribben, J. G., and Bahram, S. (2013). Reverse-engineering the genetic circuitry of a cancer cell with predicted intervention in chronic lymphocytic leukemia. *Proceedings of the National Academy of Sciences*, 110(2):459–464.

Vallat, L., Park, Y., Li, C., and Gribben, J. G. (2007). Temporal genetic program following b-cell receptor cross-linking: altered balance between proliferation and death in healthy and malignant b cells. *Blood*, 109(9):3989–3997.

Yosef, N. and Regev, A. (2011). Impulse control: temporal dynamics in gene transcription. *Cell*, 144(6):886–896.

Zhu, X., Gerstein, M., and Snyder, M. (2007). Getting connected: analysis and principles of biological networks. *Genes & development*, 21(9):1010–1024.