Networks II

Filipe Russo

August 18, 2019

Introduction

At this time we will use **only** the `R` package `igraph` as the mainframe to build networks based on Pearson Correlation and to detect modules in them.

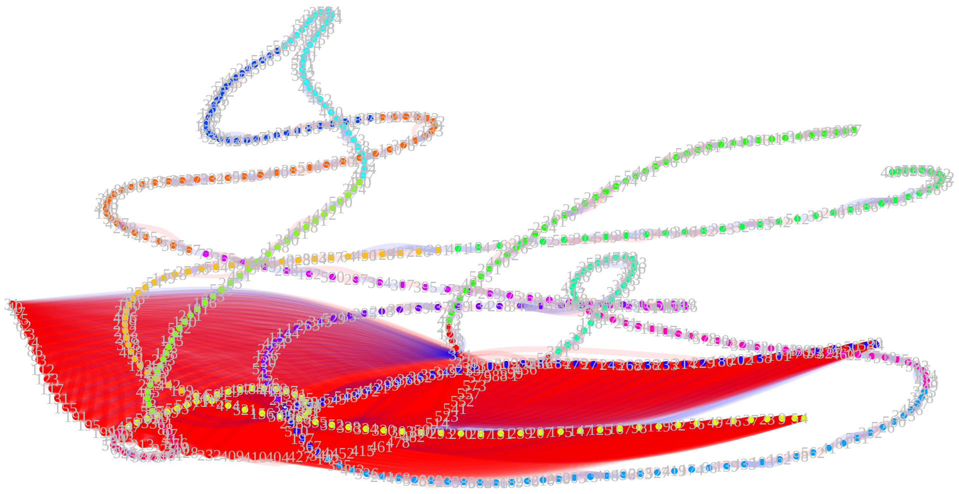
```r
library(igraph)
library(scales)
library(colorspace)
library(dplyr)
library(sda)

proteins <- read.csv("proteins.csv", sep = ",", header = TRUE)
datExprA <- t(proteins[, c("Hete", "Mixo", "Auto")])
colnames(datExprA) <- proteins$ID
```

Final Graph

I devised the `makeAdjMat` function, which guarantees the preservation of the highest absolute correlations (both lowest and highest correlations) in our `final_g` graph, but it doesn't necessarily produces a fully connected graph. The connectedness is achieved by combining this method with the Minimum Spanning Tree (MST).

```r
corMat <- cor(datExprA[, ], method = "pearson")

valid_g <- graph.adjacency(
  as.matrix(corMat),
  mode = "undirected",
  weighted = TRUE,
  diag = F,
  add.colnames = NULL,
  add.rownames = NULL
)

# A function built to make an adjacency matrix
# with the min and max values out of a correlation matrix
makeAdjMat <- function(corMat){
  diag(corMat) <- 0
  dimNum <- dim(corMat)[1]
  boolMat <- matrix(F, ncol = dimNum, nrow = dimNum)
  for(i in 1:dimNum){
    givenVec <- corMat[i, ]
    maxNum <- max(givenVec)
    minNum <- min(givenVec)
    boolVec <- givenVec == maxNum | givenVec == minNum
    corMat[which(boolVec), i] <- 0
    boolMat[i, ] <- boolVec
  }
  adjMat <- corMat * boolMat
  return(adjMat)
}

# Make a personalized adjacency matrix
weight_mat <- makeAdjMat(corMat)

# Keep only the upper diagonal values
# and turn it into a vector
weight_mat[lower.tri(weight_mat, diag = T)] <- -10
weight_v <- as.vector(t(weight_mat))
weight_v <- weight_v[weight_v != -10]

# Colour negative correlation edges as blue
E(valid_g)[which(E(valid_g)$weight < 0)]$color <- rgb(0, 0, 1, 0.1)

# Colour positive correlation edges as red
E(valid_g)[which(E(valid_g)$weight > 0)]$color <- rgb(1, 0, 0, 0.1)

# Define the appearance of the graph
V(valid_g)$size <- 1
V(valid_g)$label.cex <- 0.5
E(valid_g)$curved = TRUE
E(valid_g)$arrow.mode = 0

# Create my_g as a building block for the final_g
my_g <- valid_g

# Make a distance metric out of the absolute correlation
E(valid_g)$weight <- 1.1 - abs(E(valid_g)$weight)

# From the original graph we construct the MST (minimum spanning tree) subgraph
valid_mst <- mst(valid_g, algorithm = "prim")

# Fruchterman-Reingold is one of the most used force-directed layout algorithms out there
valid_l <- layout_with_fr(valid_mst)
valid_g$layout <- valid_l
valid_mst$layout <- valid_l

# Replace the weights with our personalized weigth vector
E(my_g)$weight <- weight_v

# Remove the 0 weighted edges
my_g <- delete_edges(my_g, E(my_g)[which(E(my_g)$weight == 0)])

# Make a distance metric out of the absolute correlation
E(my_g)$weight <- 1.1 - abs(E(my_g)$weight)

# Creates final graph by adding the edges from valid_mst
# to my_g, this way the final_g is fully connected
final_g <- add_edges(graph = my_g,
                     edges = as.vector(t(as_edgelist(valid_mst))),
                     attr = edge_attr(valid_mst))

# Remove redundant edges that came with the addition
final_g <- simplify(final_g,
                    remove.multiple = TRUE,
                    remove.loops = TRUE,
                    edge.attr.comb = list(weight = "first",
                                          color = "first",
                                          arrow.mode = "first",
                                          curved = "first",
                                          "ignore"))

# Create the modules based on edge betweenness
clusters <- cluster_edge_betweenness(graph = final_g)
members <- membership(clusters)

# Configuring the appearance of the final graph
V(final_g)$membership <- members
cols <- rainbow(length(unique(V(final_g)$membership)))[V(final_g)$membership]
V(final_g)$color <- cols
V(final_g)$frame.color <- cols
V(final_g)$label <- 1:600
V(final_g)$label.color = "grey"
E(final_g)$width <- rescale(abs(E(final_g)$weight - 1.1), to = c(1, 3))

# Define a metric to account for both degree and betweenness by multiplying both
new_intensity <- rescale(rescale(betweenness(final_g,
                                             directed = FALSE),
                                 to = c(0, 1)) *
                         rescale(degree(final_g),
                                 to = c(0, 1)),
                         to = c(0,1))

# Redefine the inner colors of the vertices according to the new connectivity metric
new_cols <- lighten(cols, new_intensity)
V(final_g)$color <- new_cols

# png(filename = "protein network.png",
#     width = 15000,
#     height = 15000,
#     units = "px",
#     res = 400)

plot(
  final_g,
  layout = valid_l,
  asp = FALSE,
  main = "Coexpression Network of 600 Proteins"
)
```

Coexpression Network of 600 Proteins



```
# dev.off()
```

Our 600 proteins are now connected to each other by a network of 3456 connections accros 16 modules.

Highlighted Proteins

We can see below how the combined metric `Highlight`, which accounts for both degree and betweenness, decreases rapidly. Proof it achieves its goal of highlighting a few interesting proteins.

```
# Save our new Partition and the Highlight of the proteins in the csv file
proteins <- proteins %>% mutate(Partition_D = members,
                                Highlight = new_intensity)
write.csv(proteins, file = "proteins.csv", row.names = FALSE)

proteins %>% arrange(desc(Highlight)) %>% select(ID, Highlight) %>% head(n = 5)
```

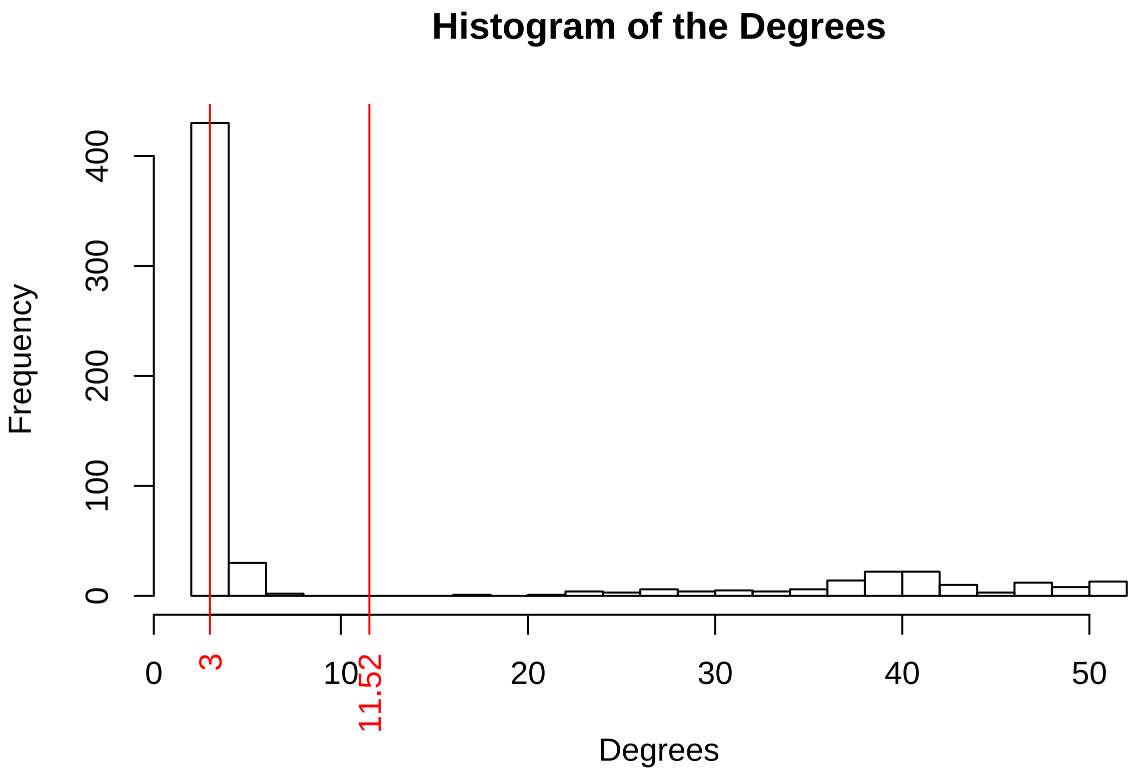
```
##                    ID Highlight
## 1 g5726.t1.g5726.t2 1.0000000
## 2          g7238.t1 0.8796268
## 3          g8268.t1 0.5305075
## 4          g9980.t1 0.2595285
## 5          g5818.t1 0.2229435
```

Degrees

The degree of a vertice in our graph is the number of connections of a protein in our network. The median degree was 3 and the mean degree was 11.52, we believe the general degree of the true network to be an integer number somewhere between these two values.

```
# png(filename = "degrees.png",
#     width = 5000,
#     height = 5000,
#     units = "px",
#     res = 400)
```

```r
degree_v <- degree(final_g)
hist(degree_v, breaks = 20, main = "Histogram of the Degrees", xlab = "Degrees")
abline(v = median(degree_v), col = "red")
abline(v = mean(degree_v), col = "red")
axis(1, at = c(3, 11.52), labels = c("3",  "11.52"),
     las = 2, col.ticks = "red", col.axis = "red")
```

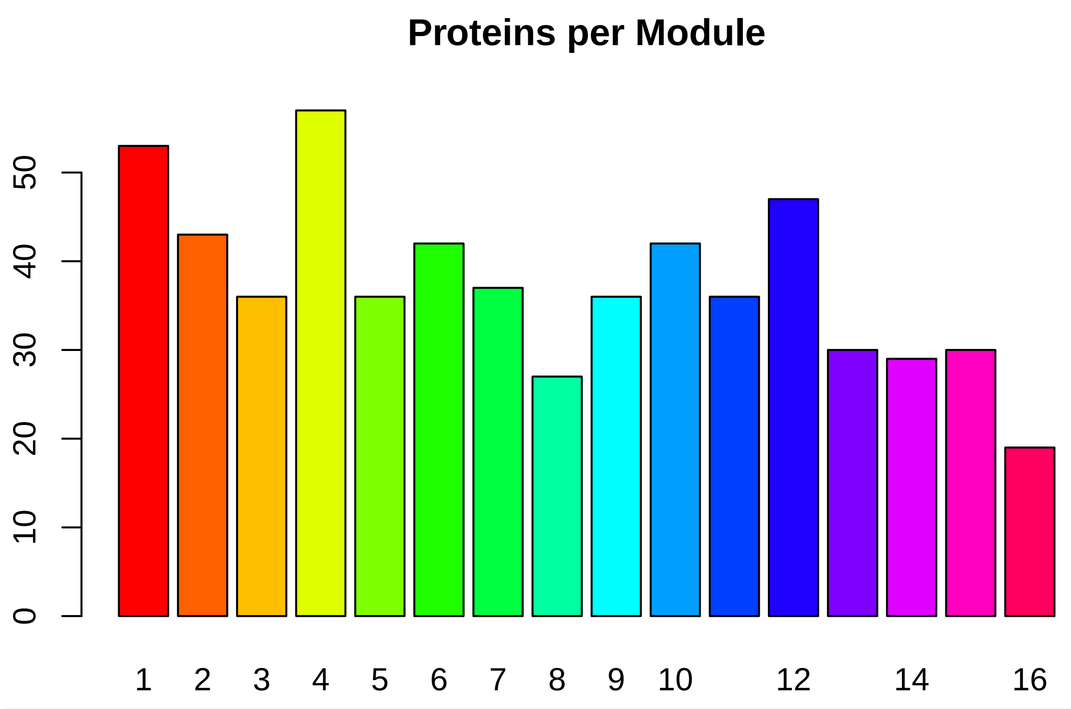


```r
# dev.off()
```

Modules

The yellow module has the highest amount of proteins, totalizing 57. On the other hand, the reddish-pink module has the lowest amount at 19. Some variance is good and to be expected, the proteins seem to be well distributed accross the modules.

```r
module_cols <- rainbow(16)

# png(filename = "modules.png",
#     width = 5000,
#     height = 5000,
#     units = "px",
#     res = 400)

barplot(table(V(final_g)$membership),
        col = module_cols,
        main = "Proteins per Module")
```



```
# dev.off()
```

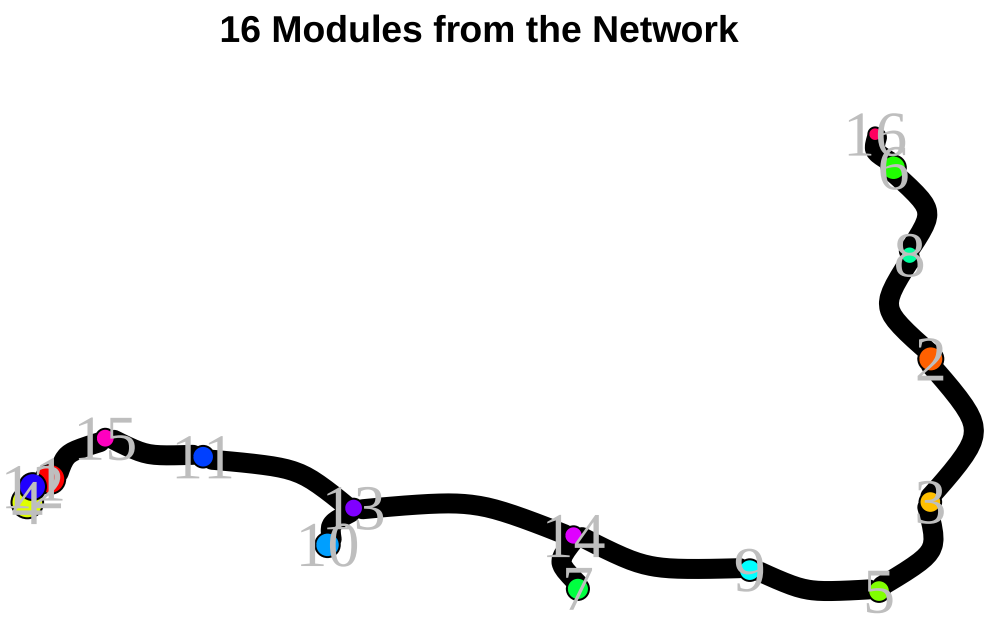
Centroids

To discover how the modules are related to each other, we decided to create a centroid for each module, a point that better represents the surrounding data points. Then we calculate the Euclidean distance between those centroids and construct a Minimum Spanning Tree (MST) based on said distances. The amount of proteins in each module was graphically translated to the size of the corresponding circle.

```
# Creates centroids out of the expression data based on the modules
centers <- centroids(x = t(datExprA), L = as.factor(members))
```

```
## Number of variables: 3
## Number of observations: 600
## Number of classes: 16
##
## Estimating optimal shrinkage intensity lambda.freq (frequencies): 0.3929
## Estimating variances (pooled across classes)
## Estimating optimal shrinkage intensity lambda.var (variance vector): 0.7154
```

```
# Creates centroids vector
centroid_v <- t(centers$means[, -c(17)])

# Creates distance object from the centroids
centroid_distances <- dist(centroid_v)

# Creates a graph for the centroids
centroid_g <- graph.adjacency(
  as.matrix(centroid_distances),
  mode = "undirected",
  weighted = TRUE,
  diag = F,
  add.colnames = NA,
  add.rownames = NA
)

# Creates a Minimum Spanning Tree out of the centroids graph
centroid_mst <- mst(centroid_g, algorithm = "prim")
centroid_l <-  layout_nicely(centroid_mst)

# Defining the appearance of centroid_mst
V(centroid_mst)$size <- rescale(as.vector(table(V(final_g)$membership)), to = c(3, 7))
V(centroid_mst)$label.cex <- 2
V(centroid_mst)$color <- module_cols
V(centroid_mst)$label.color = "grey"
E(centroid_mst)$curved = TRUE
E(centroid_mst)$arrow.mode = 0
E(centroid_mst)$width <- 10
E(centroid_mst)$color <- "black"

# png(filename = "centroids.png",
#     width = 15000,
#     height = 15000,
#     units = "px",
#     res = 400)

plot(
  main = "16 Modules from the Network",
  centroid_mst,
  layout = centroid_l,
  asp = FALSE
)
```



```
# dev.off()
```

Final Remarks

We were able to construct a Coexpression Network of Proteins using only the `R` package `igraph`. Then we highlighted interesting proteins based on their net connectivity. At last we studied the modules by constructing a new network based on centroids. We hope this work offers insights and tools on how to further develop the research field of biological networks.