Analysis of the protein-protein interaction (PPI) network on categories of the node (protein) connectivity.

The network (mito.csv) was taken from BioGrid (version 3.4.160), which contains physical interactions among the yeast (*S. cerevisiae*) mitochondrial proteins (21 proteins encoded in the mitochondrial genome) and their first-layer interacting partners (83 proteins encoded in the nucleus). This network has 104 nodes (proteins) in total, forming 674 edges (PPIs).

1. The first step is to calculate the node-connectivities, which is the total number of other nodes that a node connects. Below is the R code (connectivity.R):

```
###################################################################
##Read the network as my.data
my.data <- read.csv("mito.csv", header=T, stringsAsFactors = F)
##Read the list of all nodes as my.list
my.list <- read.csv("mito.list", header=T, stringsAsFactors=F)
gene <- my.list$gene
##Define a vector that stores both the gene names and their connectivities
pair <- c("Gene", "Conn")
##my.data has both “geneA” and “geneB”, the connectivity counts for both locations
##In preparing the network all self-interactions and redundant interactions are excluded
##Note that the interactions (A,B) and (B,A) are considered redundant.
for (i in 1:length(gene)) {
   gene.name <- as.character(gene[i])
   geneA.connection <- length(which(my.data$geneA %in% gene.name))
   geneB.connection <- length(which(my.data$geneB %in% gene.name))
   connectivity <- geneA.connection + geneB.connection
   pair <- rbind(pair,c(gene.name, connectivity))
}
##The connectivities are stored in “mito.conn.csv”
write.table(pair, file="mito.conn.csv",sep=",",row.names=F,col.names=F)
###################################################################
```

The next step will statistically analysis the architecture of the network in frame of the node-connectivities calculated in this step.

2. Connectivities of all nodes are appended to the network using the script “mito.conn.R”

```
###################################################################
# Read the connectivity of all genes
conn.info <- read.csv("mito.conn.csv",header=TRUE, stringsAsFactors=F)

# read the network
mito <- read.csv("mito.csv",header=TRUE, stringsAsFactors=F)
geneA <- mito$geneA
```

```
geneB <- mito$geneB

orf <- conn.info$Gene
conn <- conn.info$Conn

# the "which" funciton was utilized to distribute the info to the network
matcha <- which(geneA %in% orf)
matchb <- which(geneB %in% orf)
# the overlaps between {Ai} and {Bi} will gives the locations of the pairs, in which both
ORFs are included,
# i.e., matcha[match2] will give the locations of ORFs in geneA that has been included
in the set
match2 <- which (matcha %in% matchb)

pair <- c("geneA","geneB","connA","connB", "physical.conn")

for (i in 1:length(match2)) {
   a0 <- as.numeric(match2[i])
   a1 <- as.numeric(matcha[a0])
   linkA <- as.character(geneA[a1])
   linkB <- as.character(geneB[a1])
   t <- which(orf %in% geneA[a1])
   connA <- conn[t]
   s <- which(orf %in% geneB[a1])
   connB <- conn[s]
   phys.connectivity <- round(sqrt(connA * connB),3)
   pair = rbind(pair,c(linkA,linkB,connA,connB,phys.connectivity))
}

write.table(pair,file="mito.by.conn.csv",sep=",",row.names=F,col.names=F)
##################################################################
```

Here the term “physical connectivity” was defined by Ozier et al. (Nature Biotech. 2003) that may be used in further analysis (not here).

The above job can be submitted to run on Ts117 using a bash script “mito.pbs”:

```
##################################################################
#!/bin/bash -l
#$ -S /bin/bash
#$ -N mito.conn
#$ -cwd
. /etc/profile.d/modules.sh
module load shared
module load R/3.4.3
cmd=“R -f mito.conn.R”
$cmd
##################################################################
```

After that, submit the pbs script using terminal:
$ qsub mito.pbs

3. To analyze the architecture of the original network, it is better to compare with random “null” models. We use an algorithm originally proposed by Maslov and Sneppen (Science 2002), or the ms02 model here after and revised by Qin et al. (PNAS 2003). In this method, patterns of node-connections are kept while random permutations of all connecting nodes were performed. The current small test do 100 random permutations (higher number of random models are needed for statistical significance). The code used here is “ms02-old.R”

```
################################################################
#require(igraph)
rm(list=ls())
debug = 0
#set.seed(2014)
permute.pairs.wo.selfpairs = function( inpairs,  ncycles=10, debug=1 ) {
  if (ncycles >= 1 ) {
    if(debug) {
      print(paste('ncycles=', ncycles))
    }
    longids = c(as.character(inpairs[,1]), as.character(inpairs[,2]) )
    longids = sample(longids)
    len = length(inpairs[,1])
    newpairs = data.frame( cbind( longids[1:len], longids[(len+1): (2*len)]) )
    names(newpairs) = c('geneA', 'geneB')
    newpairs$geneA = as.character( newpairs$geneA)
    newpairs$geneB = as.character( newpairs$geneB)
    newpairs$selfpairs = ifelse( newpairs$geneA == newpairs$geneB, 1, 0 )
    self.tb = newpairs[ newpairs$selfpairs==1, ]
    nonself.tb = newpairs[newpairs$selfpairs==0, ]
    if(debug) {
      print(paste("===selfpairs===="),NULL)
      print(self.tb)
      print(paste("================="),NULL)
    }
    if( length(self.tb[,1])>=1 ) {
      if ( ncycles == 0) {
        #return (c(NA,NA, NA) );
        print(paste("ncycles reached zero, ncycles"),ncycles)
        print(paste("Abort!"),NULL)
        stop;
      } else {
        ncycles = ncycles - 1
        splitPos = round( length(self.tb[,1]) * sqrt(ncycles) ) + 5  #2014Jan31 change
        splitPos = min( splitPos, (length(nonself.tb[,1])-1 ) )
```

```
      selectedpairs = rbind(self.tb,  nonself.tb[1: splitPos, ] )
      restpairs = nonself.tb[ (splitPos + 1): length(nonself.tb[,1]), ]
      return( rbind(restpairs, permute.pairs.wo.selfpairs(selectedpairs, ncycles)))
    }
  } else {
    return (newpairs)
  }
 } else {
  return( c(NA,NA,NA ))
 }
}

#R -f file --args start end
options(echo=TRUE) # if you want see commands in output file
args <- commandArgs(trailingOnly = TRUE)
print(args)
start = as.integer(args[1])
end = as.integer(args[2])

debug = 9

#mydir = "/Users/yhw543/Documents/GitHub/simcenter_clusters_excises"
#setwd(mydir)
#list.files()
#source('network.r')

net = read.table( "mito.csv", header=T, sep=",", colClasses = c("character",
"character") )
head(net)

for( i in start:end) {
 net.ms02 = permute.pairs.wo.selfpairs( net )
 cmnd = paste( "mkdir ms02/", i, sep="")
 system( cmnd )
 outputname = paste( 'ms02/', i, '/', "ms02_",i,".csv", sep="")
 write.table(net.ms02, outputname, quote=F, row.names=F, sep=",")
}
#################################################################
```

To run this job, in a terminal, use
$ R -f ms02-old.R -- args 1 100

The arguments 1 and 100 are the first and the last serial numbers of the random models, respectively. All models will be saved as csv files in a folder “ms02”.

4. Repeat step 2 on all random models. The R-script for the ms02_1.csv (saved in ./ms02/1/) is “ms02.R.1”, the difference between this script and that used in step 2

(mito.conn.R) is the network file, which is

```
####
mito <- read.csv("route.to.ms02/ms02/1/ms02_1.csv", header = TRUE, stringsAsFactors
= F)
####
```

And the output

```
####
write.table(pair, file="ms02_1.by.conn.csv", sep=",", row.names=F, col.names.F)
####
```

A simple shell script can be used to generate all scripts for other ms02 models. Here, for ms02_2 to ms02_100, for example:

```
####
#!/bin/csh -f
set i = 1
while ($i < 100)
  @ i++
  cat ms02.R.1 | sed -e "s/ms02_1/ms02_$1/g" | sed -e "s/\/1\/\/$i\/g" > ms02.R.$i
end
####
```

After generating all script ms02.R.1, ms02.R.2, …, ms02.R.100, a pbs script could be written as

```
####
#!/bin/bash -l
#$ -S /bin/csh
#$ -t 1-100
#$ -N mito.ms02.100
#$ -cwd
. /etc/profile.d/modules.sh
module load shared
module load R/3.4.3
cmd="R -f ms02.R.$SGE_TASK_ID"
$cmd
####
```

Then
$ qsub ms02.pbs

All jobs will be submitted simultaneously, but run individually.

5. Now we have the file "mito.by.conn.csv" for the original network and

ms02_x.by.conn.csv for the random models (x = 1 to 100) and want to know the architectural differences between the original network and that expected for random models. To do this job, we use the frame work of connectivities of all the nodes that had been calculated in step 1 (“mito.conn.csv”). To do this, all nodes (proteins) are classified in ten categories based on the quantiles of their connectivities for the whole set of nodes. For example, of a pair (A, B), if the connectivity of A is in the m-th quantile and that of B is in the n-th quantile (m, n = 1, 2, … 10) of all connectivities, the pair (A, B) belongs to the category (m, n). Symmetrically, (A, B) and (B, A) are considered as the same interaction, such that the category (m, n) and (n, m) are equivalent to each other as we want to generate a frequency distribution matrix on the full 10*10 space. Here is the script to do generate the matrix (note that, when m == n, because this script will double write twice to the category [m,n] and [n,m], the diagonal elements of the final matrix should be decided by two. Here is the R-script (“matrix.R”)

```
#################################################################
# All genes are classified in N (= 10 here) categories based on quantiles of the connectivity values
quant.data <- read.csv("/home/hguo/GitHub/simcenter_clusters_excises/mito.conn.csv", header = TRUE)
quant <- quantile(quant.data$Conn, probs = seq(0, 1, by=0.1))

mydata <- read.csv("mito.by.conn.csv", header = TRUE)
connA <- mydata$connA
connB <- mydata$connB

# First making a 10*10 matrix with zero elements
A <- matrix(0, nrow=10, ncol=10)
B <- matrix(0, nrow=10, ncol=10)

# for each gene pair the elemments A_mn will be changed depending on the RLS values of the two genes;
# symmetricity has been considered from the original matrix; i.e., both gene pairs (i,j) and (j,i) are
# included in the csv file

for (i in 1:length(connA)){
   if (connA[i] > quant[10]) {
      m = 10
   }  else {
      if (connA[i] > quant[9]) {
         m = 9
      }  else {
         if (connA[i] > quant[8]) {
            m = 8
         }  else {
            if (connA[i] > quant[7]) {
               m = 7
```

```
        }   else {
            if (connA[i] > quant[6]) {
                m = 6
            }   else {
                if (connA[i] > quant[5]) {
                    m = 5
                }   else {
                    if (connA[i] > quant[4]) {
                        m = 4
                    }   else {
                        if (connA[i] > quant[3]) {
                            m = 3
                        }   else {
                            if (connA[i] > quant[2]) {
                                m = 2
                            } else {
                                m = 1
                            }
}}}}}}}

if (connB[i] > quant[10]) {
    n = 10
}   else {
    if (connB[i] > quant[9]) {
        n = 9
    }   else {
        if (connB[i] > quant[8]) {
            n = 8
        }   else {
            if (connB[i] > quant[7]) {
                n = 7
            }   else {
                if (connB[i] > quant[6]) {
                    n = 6
                }   else {
                    if (connB[i] > quant[5]) {
                        n = 5
                    }   else {
                        if (connB[i] > quant[4]) {
                            n = 4
                        }   else {
                            if (connB[i] > quant[3]) {
                                n = 3
                            }   else {
                                if (connB[i] > quant[2]) {
                                    n = 2
                                } else {
```

```
                                n = 1
                            }
    }}}}}}}}

    A[m,n] <- A[m,n] + 1
    A[n,m] <- A[n,m] + 1
}

## Considering the symmetricity of the matrix A, if an interaction is counted in A[m,n],
## this script will also add the interaction in A[n,m].
## However, for A[m,m], i.e., the diagonal elements of the matrix, it was double counted,
## and therefore need be corrected to matrix B.

for (m in 1:10) {
    for (n in 1:10){
        if (m == n) {
            B[m,n] = A[m,n]/2
        }  else {
            B[m,n] = A[m,n]
        }
    }
}


write.table(B, file="mito.matrix.csv",sep=",",row.names=F,col.names=F)

#################################################################
```

Note that, the quantile function is universal and can be applied to any other characters. You will find the 10*10 matrix saved in “mito.matrix.csv” as

```
0,0,4,1,0,0,0,0,1,17
0,0,2,1,1,3,1,0,2,17
4,2,2,5,2,3,1,1,6,42
1,1,5,5,8,3,9,6,7,63
0,1,2,8,0,3,2,8,5,25
0,3,3,3,3,0,12,22,8,41
0,1,1,9,2,12,8,17,19,74
0,0,1,6,8,22,17,19,34,75
1,2,6,7,5,8,19,34,10,43
17,17,42,63,25,41,74,75,43,36
```

Which means, the frequency of gene pairs in the [1,1] category is 0, and that in the [10,10] category is 36, etc. This is reasonable because the category of [10,10] are formed by nodes with higher connectivities (nodes the 10th quantile has connectivities >90% of all other nodes), whereas that in [1,1] are by nodes with lower connectivities (<90% of all other nodes).

This script will be finished very fast (seconds) and can be run directly in a terminal (do not use the head node though):

$ R -f matrix.R

For all (100) random models a similar script can be utilized; and to run it in terminal, a shell script (“matrix.csh”) can be again adopted:

```
####
#!/bin/csh -f
set i = 0
while ($i < 100)
   @ i++
   cat matrix.R | sed “s/mito.by/ms02_$i.by/g” | sed “s/mito.matrix/ms02_$i.matrix/g” >
tmp.R
   R -f tmp.R
   rm tmp.R
end
####
```

Simply run (in seconds) as:
$ ./matrix.csh

All matrices for random models are saved as ms02_x.matrix.csv (x = 1,2,…100).

To find out the difference between the original network and random models, the Z-scores is calculated for each of the matrix elements via

z = (freq_original_i,j - mean_i,j)/std_i,j

Here freq_original_i,j is the frequency in the [i,j] category calculated in the original network; mean_i.j is the mean value of the frequencies in the [i,j] category of all random models; std_i,j is the standard deviation of frequencies in the [i,j] category of all random models.

Another script (“zscore.R”) is used for this calculation:

```
#####################################################################
mito <- read.table("mito.matrix.csv", header = F, sep = ",")


for (m in 1:10) {
   for (n in 1:10) {
      e <- c()
      for (i in 1:100) {
         name <- paste("ms02_", i, ".matrix.csv", sep = "")
```

```
            mat <- read.table(name, header = F, sep = ",")
            e <- rbind(e,mat[m,n])
        }
        mito[m,n] <- round((mito[m,n] - mean(e))/sd(e),3)
    }
}

write.table(mito, file="mito.matrix.zscore.csv", sep = ",", row.names=F, col.names=F)
#################################################################
```

Below is the z-matrix based on 100 random models (different runs may lead to subtly different results, which is expected by the randomness of the tests:

```
-0.434,-0.669,2.731,-0.581,-0.985,-1.27,-1.837,-2.031,-0.993,3.793
-0.669,-0.421,0.402,-0.945,0,0.718,-1.168,-2.098,-0.506,2.796
2.731,0.402,0.265,-0.299,-0.463,-0.96,-2.69,-3.594,-0.786,4.788
-0.581,-0.945,-0.299,0.375,1.797,-1.829,-1.533,-2.704,-1.551,4.505
-0.985,0,-0.463,1.797,-1.017,-0.455,-1.805,-0.126,-0.191,1.66
-1.27,0.718,-0.96,-1.829,-0.455,-1.801,0.503,2.195,-0.72,1.237
-1.837,-1.168,-2.69,-1.533,-1.805,0.503,0.269,-1.769,0.774,3.518
-2.031,-2.098,-3.594,-2.704,-0.126,2.195,-1.769,1.387,3,0.683
-0.993,-0.506,-0.786,-1.551,-0.191,-0.72,0.774,3,1.411,-1.574
3.793,2.796,4.788,4.505,1.66,1.237,3.518,0.683,-1.574,-6.065
```

This matrix can be visualized using the plotly and webshot libraries in R (“zscore.plot.R”, please run it in R studio):

```
#################################################################
##
library(plotly)
library(webshot)

z <- read.csv2("mito.matrix.zscore.csv", header = FALSE, sep = ",", stringsAsFactors =
F)

conn_A <- c(0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0)
conn_B <- c(0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0)

freq <- c(z$V1,z$V2,z$V3,z$V4,z$V5,z$V6,z$V7,z$V8,z$V9,z$V10)

p <- plot_ly(type = 'contour',
    z = matrix(freq, nrow=10, ncol=10), color=~z,
    x=~conn_A, y=~conn_B, width=500, height=400,
    contours=list(start=-3,end=3,size=0.5), colorbar=list(title="Z-score"))
a <- list(title="Quantiles of Connectivity")
b <- list(title="Quantiles of Connectivity")
```

```
layout(p, xaxis=a, yaxis=b, title="Mitochondrial Set in Connectivity")
## the webshot package will be needed to export the plotly figure
export(layout(p,title="Mitochondrial Set in Connectivity",
xaxis=a,yaxis=b),file="Mito.z.conn.png")
#################################################################
```

A test of the final result is shown on GitHub:
https://github.com/QinLab/simcenter_clusters_excises/blob/master/Mito.z.conn.png

This plot clearly indicates that the hub-to-hub interactions (hubs are the code with high connectivities) are significantly decreased (in upper right corner, in blue color), whereas the hub-to-nonhub interactions are increased, in the original network. This feature had also been observed in other biological networks.

This Figure is based on the quantiles of connectivities but not the real connectivity levels. Because all quantiles have been calculated using the connectivities of all nodes, it is straight forward to use the real connectivities, too. Check the script “zscore.plot2.R” and final result “Mito.z.conn2.png” on GitHub.

Hao-Bo, 5-11-2018.