`CoSync`: an R package for Co-Synchronization network analysis of pseudo-temporally ordered single cell data

Ye Chen, Shouguo Gao, Nathaniel Wolanyk, Xujing Wang
Systems Biology Center
National Heart, Lung, and Blood Institute
National Institutes of Health
ye.chen@nih.gov

Feburary 2016
Version 1.1 updated: Feburary 2016

Contents

1 Background

The `CoSync` package is designed to construct co-synchronization networks. The metrics included in this package are: phase synchronization index, granger causality, and coherence.

Recently we have seen a rapid blossom of single cell proteomic and transcriptomic technologies. In particular, it has be observed that cells obtained during a biological process, such as differentiation, activation, development, represent a continuum of the intermediate cell states through the process trajectory, and algorithms have been developed to pseudo-temporally ordered along the virtual trajectory. Data of pseudo-temporally ordered single cells largely addressed the two aforementioned challenges: the sample size (and hence the number of pseudo time points) is much larger than typical bulk sample studies; more importantly, the single-cell level measurements preserve the critical variation information that is lost by bulk assays.

Here, we assumed the single cells are pseudo-temporally ordered with data properly normalized. Cosync takes the ordered data as input, and return the co-synchronization modules and GO enrichment analysis.

2 Summary of method

The CoSync calculates phase synchronization index, granger causality, and coherence and uses as the weigt of between each pair of genes.

Implementing the CoSync involves the following steps:

1. Pick the top n (suggested $n < 5000$ for computational timing issue) genes with highest variation to build the network.
2. Calculate the pair-wise co-synchronization index.
3. Define the modules by WGCNA using soft-thresh-hold.
4. Export and plot the results.

3 Detailed implementation

The complete analysis is performed in steps, you will need to pick the soft-thresh hold before you define the modules.

3.1 Discover the co-synchronization index

The library depends on

Loading the package

```
#library(Cosync)
library(RSEIS)
library(MSBVAR)
library(igraph)
library(WGCNA)
library(biomaRt)
```

The advantge of considering co-synchronization index other than correlation can be depicted as the following example. Using the `phaseLocking` function to estimate the phase locking index.

```
x=sin(1:100)
y=cos(1:100)
cor(x,y)
```

```
## [1] -0.002733781
```

```
phaseLocking(x,y)
```

```
## $`entropy rho`
## [1] -0.8491037
##
## $gamma
## [1] 0.999224
##
## $`strobo lmb`
## [1] 0.9941078
##
## $`strobo ang`
## [1] -1.572809
```

```
z=t(matrix(c(x,y),ncol=2))
grangerTest(z)
```

```
## $F_statistics_matrix
##          [,1]     [,2]
## [1,]     0.00 67615.53
## [2,] 67615.53     0.00
##
## $p_value_matrix
##      [,1] [,2]
## [1,]    0    0
## [2,]    0    0
```

```
coherenceTest(z)
```

```
## [1] 0.9999764
```

As we can see, the correlation is low while the co-synchronization indcies are high. Let's use the psudo ordered data from the library monocle as a display example.

```
load("./data/HSMM.RData")
```

```
## Warning in readChar(con, 5L, useBytes = TRUE): cannot open compressed file './data/HSMM.RData',
probable reason 'No such file or directory'
```

```
## Error in readChar(con, 5L, useBytes = TRUE): cannot open the connection
```

We will have to resstrict the number of genes for the computaional time issue and clear result (??WGCNA ref??). By monocle, the samples are splitted to 3 states. Here, we consider the state 1 and 2. We will build up the synchronization modules for each state, and find the intersection of them.

```
n_genes = 2000
sd_all = apply(expr_all,1,function(x) sd(x))
expr1 = as.matrix(expr_all[names(sort(sd_all,decreasing = T)[1:n_genes]),
                           rownames(subset(HSMM_pheno,State=="1"))])
rownames(expr1)=HSMM_feature[rownames(expr1),1]
expr2 = as.matrix(expr_all[names(sort(sd_all,decreasing = T)[1:n_genes]),
                           rownames(subset(HSMM_pheno,State=="2"))])
rownames(expr2)=HSMM_feature[rownames(expr2),1]
```

Using phase locking index as an example, we calculate the phase locking index matrix as the following. This step may take a long time. (Note that the functions phaseLockingMatrix and coherenceTest provide the progressing bar, but grangerTest does not.)

```
#!!!!!!!!!!correct here for the final version
#phase1=phaseLockingMatrix(expr1)
#phase2=phaseLockingMatrix(expr2)
#entropy rho, many 0, very few close to 1, not work
#gamma, best hit of x is 2, best hit of y is 1 and only 1.
#strobo_lmbda, none-hit 0.9 which is the scale free fit
#coherence is far away from scale free.
#so we use rho here for now.
#This step takes ~5 hours for phase locking and granger, ~2 hours for coherence. So we precompute the r
load("./data/phase_locking_HSMM.RData")
```

```
## Warning in readChar(con, 5L, useBytes = TRUE): cannot open compressed file './data/phase_locking_HSMM
probable reason 'No such file or directory'
```

```
## Error in readChar(con, 5L, useBytes = TRUE): cannot open the connection
```

Throughout this paper, we use strobo lambda for the co-synchronization index.

3.2 Buidling modules

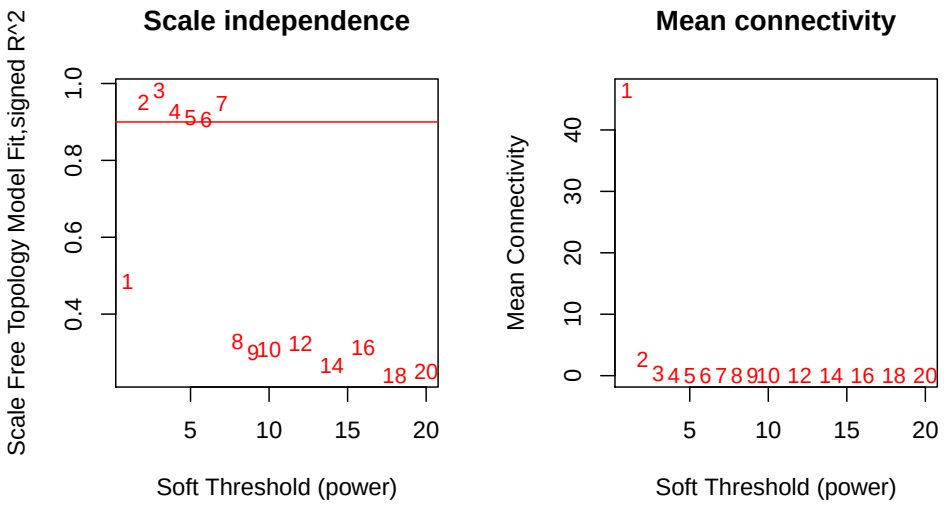
We assume that the whole network contains the genes are vertices, and co-synchronization index as the edge weight between vertices. To make the weighted network scale free, we turn the weighted adjacency matrix

to WGCNA and find a reasonable soft-threshold.

```
power1=wgcnaPower(phase1$gamma)
```

Check the result in the plot. Find the smallest number that passes 0.9. If none, then use 1 as the soft power for next step.

```
par(mfrow = c(1,2));
  cex1 = 0.9;
  # Scale-free topology fit index as a function of the soft-thresholding power
  plot(
    power1$fitIndices[,1],-sign(power1$fitIndices[,3]) * power1$fitIndices[,2],
    xlab = "Soft Threshold (power)",ylab = "Scale Free Topology Model Fit,signed R^2",type =
      "n",
    main = paste("Scale independence")
  );
  text(
    power1$fitIndices[,1],-sign(power1$fitIndices[,3]) * power1$fitIndices[,2],
    labels = c(c(1:10), seq(from = 12, to=20, by=2)),cex = cex1,col = "red"
  );
  # this line corresponds to using an R^2 cut-off of h
  abline(h = 0.90,col = "red")
  # Mean connectivity as a function of the soft-thresholding power
  plot(
    power1$fitIndices[,1], power1$fitIndices[,5],
    xlab = "Soft Threshold (power)",ylab = "Mean Connectivity", type =
      "n",
    main = paste("Mean connectivity")
  )
  text(
    power1$fitIndices[,1], power1$fitIndices[,5], labels = c(c(1:10), seq(from = 12, to=20, by=2)), cex
      "red"
  )
```



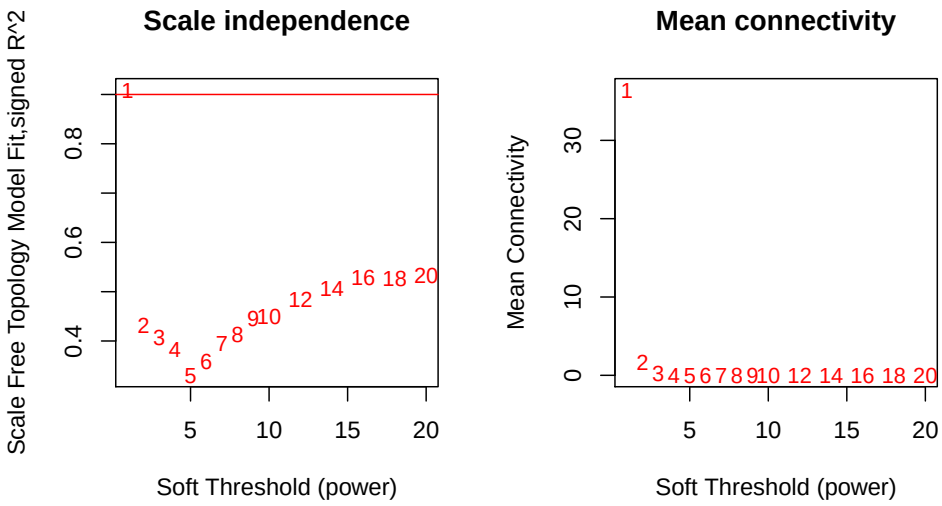
And we do the same for the 2nd state co-synchronization matrix.

```
power2=wgcnaPower(phase2$gamma)
```

Check the result in the plot.

```
par(mfrow = c(1,2));
  cex1 = 0.9;
  # Scale-free topology fit index as a function of the soft-thresholding power
  plot(
    power2$fitIndices[,1],-sign(power2$fitIndices[,3]) * power2$fitIndices[,2],
    xlab = "Soft Threshold (power)",ylab = "Scale Free Topology Model Fit,signed R^2",type =
      "n",
    main = paste("Scale independence")
  );
  text(
    power2$fitIndices[,1],-sign(power2$fitIndices[,3]) * power2$fitIndices[,2],
    labels = c(c(1:10), seq(from = 12, to = 20, by = 2)),cex = cex1,col = "red"
  );
  # this line corresponds to using an R^2 cut-off of h
  abline(h = 0.90,col = "red")
  # Mean connectivity as a function of the soft-thresholding power
  plot(
    power2$fitIndices[,1], power2$fitIndices[,5],
    xlab = "Soft Threshold (power)",ylab = "Mean Connectivity", type =
      "n",
    main = paste("Mean connectivity")
  )
  text(
    power2$fitIndices[,1], power2$fitIndices[,5], labels = c(c(1:10), seq(from = 12, to = 20, by = 2)),
```

```
      "red"
  )
```



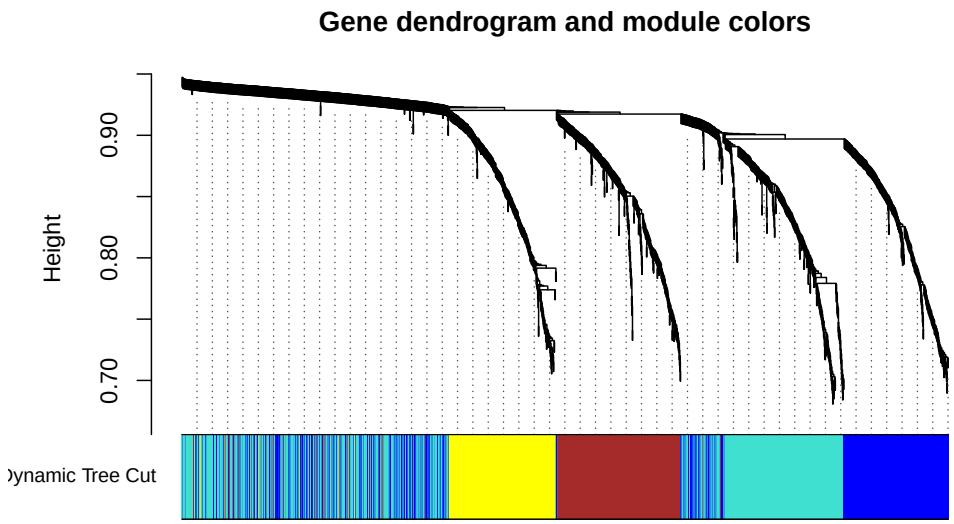
Base on the figures, we pick 1 as the soft-threshold power for the co-synchronization matrices, which is the original phase index. If none of the power gets the distribution scale free, we suggest to keep the original matrix, which the power is 1. We compile WGCNA functions into the one which returns the modules and the dendrogram. Note that we set the minimum module size we set is 30 as we are performing GO enrichment analysis later.

```
modules1=wgcnaAnalysis(phase1$gamma,softPower=1)


## ..connectivity..
## ..matrix multiplication..
## ..normalization..
## ..done.
##  ..cutHeight not given, setting it to 0.945  ===>  99% of the (truncated) height range in dendro.
##  ..done.
##  mergeCloseModules: Merging modules whose distance is less than 0.25
##    multiSetMEs: Calculating module MEs.
##      Working on set 1 ...
##      moduleEigengenes: Calculating 4 module eigengenes in given set.
##    Calculating new MEs...
##    multiSetMEs: Calculating module MEs.
##      Working on set 1 ...
##      moduleEigengenes: Calculating 4 module eigengenes in given set.
```

Check the tree and the modules in the plot.

```
WGCNA::plotDendroAndColors(
    modules1$geneTree, modules1$modulColors, "Dynamic Tree Cut", dendroLabels = FALSE, hang = 0.03, add
  )
```



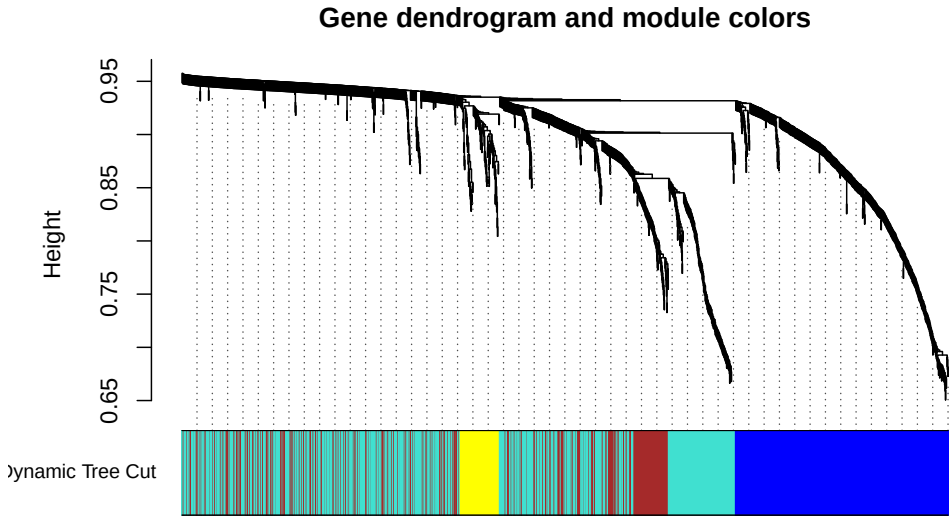
We do the same for the state 2 samples.

```
modules2=wgcnaAnalysis(phase2$gamma,1)


## ..connectivity..
## ..matrix multiplication..
## ..normalization..
## ..done.
##  ..cutHeight not given, setting it to 0.955  ===>  99% of the (truncated) height range in dendro.
##  ..done.
##  mergeCloseModules: Merging modules whose distance is less than 0.25
##    multiSetMEs: Calculating module MEs.
##      Working on set 1 ...
##      moduleEigengenes: Calculating 4 module eigengenes in given set.
##    Calculating new MEs...
##    multiSetMEs: Calculating module MEs.
##      Working on set 1 ...
##      moduleEigengenes: Calculating 4 module eigengenes in given set.
```

Check the tree and the modules in the plot.

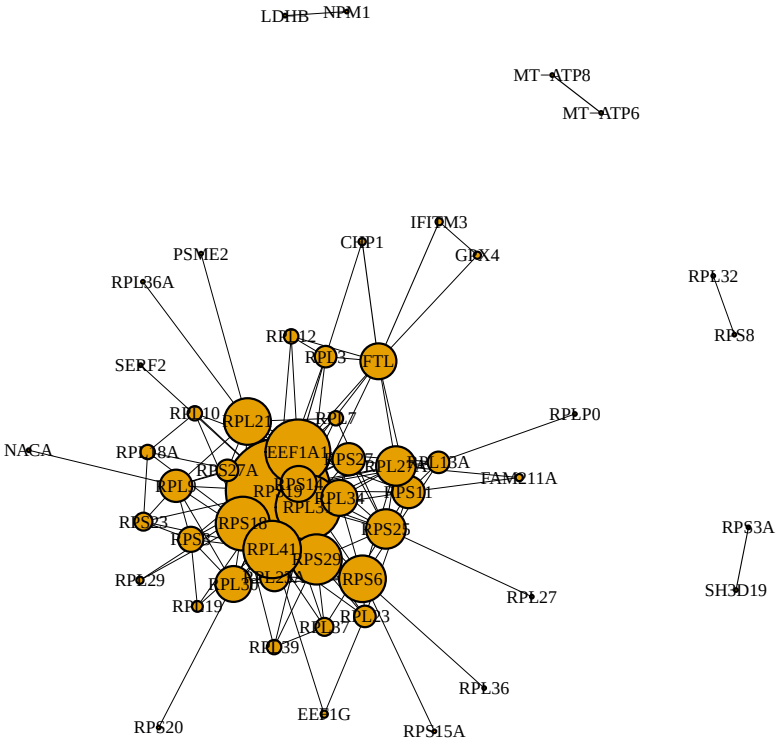
```
WGCNA::plotDendroAndColors(
    modules2$geneTree, modules2$modulColors, "Dynamic Tree Cut", dendroLabels = FALSE, hang = 0.03, add
  )
```

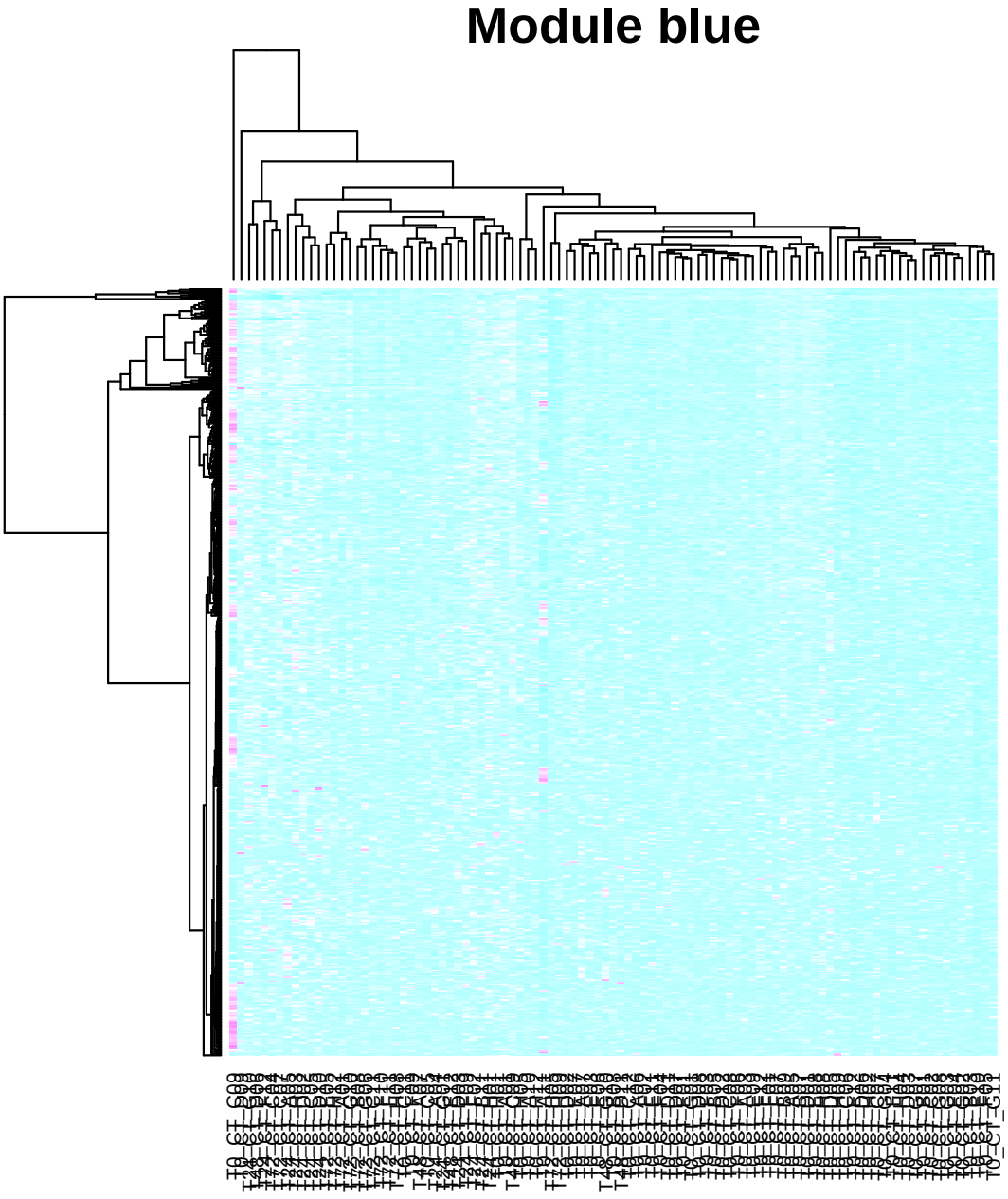


To look into each module, the ModulePlots funtion can plot out the graph for the indicated module and row-centered heatmap of each module. The edges are shown in the graph if their weights are over the threshold. Vertex size is proportional to its vertex degree.

```
ModulePlots(x=phase1$gamma,expr=expr1,mColors=modules1$modulColors,thisColor = "blue",cut = 0.3)
```

Module blue


LDHB
NPM1
MT-ATP8
MT-ATP6
IFITM3
CHP1
PSME2
GPX4
RPL36A
RPL32
RPS8
RPL12
RPL3
FTL
SERF2
RPL10
RPL21
RPLP0
NACA
RPL18A
EEF1A1
RPS27A
RPL13A
FAM211A
RPS11
RPL34
RPS23
RPS18
RPS25
RPS3A
RPS3
RPL41
RPS29
RPS29
RPS6
SH3D19
RPL27
RPL19
RPL23
RPL37
RPL39
RPL36
EEF1G
RPS20
RPS15A



With each module, we can perform GO enrichment analysis (WGCNA function) to investigate the dominent ontology. We also provide the GO annotation for each gene using biomaRt as well. These functions are compiled into `ModuleAnnotation` which returns a table.

```
state1_module=ModuleAnnotation(colors = modules1$modules)
```

Check the result.

```
state1_module[1:5,1:8]
```

```
##            module modSize bkgrModSize rank  enrichmentP   BonferoniP
## GO:0022626   blue     512         494    1 1.539876e-46 2.447633e-42
## GO:0000184   blue     512         494    2 1.796736e-40 2.855912e-36
## GO:0006614   blue     512         494    3 1.394256e-39 2.216169e-35
## GO:0006613   blue     512         494    4 2.378423e-39 3.780504e-35
## GO:0045047   blue     512         494    5 7.882524e-38 1.252927e-33
##            nModGenesInTerm fracOfBkgrModSize
## GO:0022626              83         0.1680162
## GO:0000184              79         0.1599190
## GO:0006614              87         0.1761134
## GO:0006613              88         0.1781377
## GO:0045047              88         0.1781377
```

So for the state 2 modules.

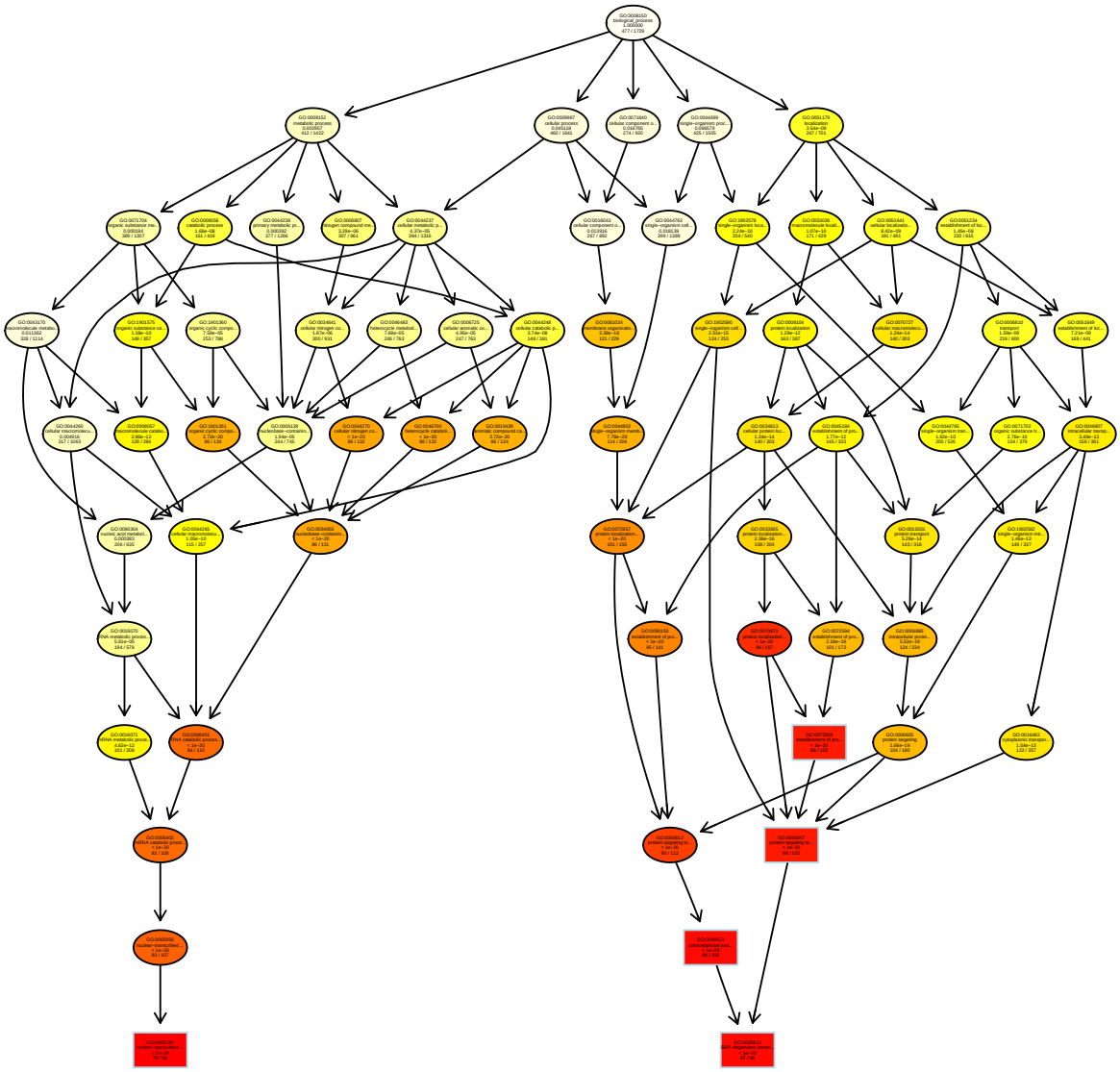
```
state2_module=ModuleAnnotation(colors = modules2$modules)
```

Another GO enrichment analysis option - topGO, is also provided for GO enrichment analysis.

```
topgo1=topGOanalysis(modules1$modules)
```

```
##
## Building most specific GOs ..... ( 4295 GO terms found. )
##
## Build GO DAG topology .......... ( 7825 GO terms and 18437 relations. )
##
## Annotating nodes ............... ( 1729 genes annotated to the GO terms. )
##
##   -- Classic Algorithm --
##
##   the algorithm is scoring 4551 nontrivial nodes
##   parameters:
##   test statistic:  fisher
```

```
## Error in pdf(file = paste(out.fileName, "pdf", sep = "."), width = 10, :  cannot open file
'TRUE_classic_5_all.pdf'
```



```
head(topgo1)
```

```
##      GOterm
## [1,] "GO:0000184"
## [2,] "GO:0006614"
## [3,] "GO:0006613"
## [4,] "GO:0045047"
## [5,] "GO:0072599"
## [6,] "GO:0070972"
##      GODesc
```

```
## [1,] "nuclear-transcribed mRNA catabolic process, nonsense-mediated decay"
## [2,] "SRP-dependent cotranslational protein targeting to membrane"
## [3,] "cotranslational protein targeting to membrane"
## [4,] "protein targeting to ER"
## [5,] "establishment of protein localization to endoplasmic reticulum"
## [6,] "protein localization to endoplasmic reticulum"
##      pValue
## [1,] "3.37249351885948e-40"
## [2,] "2.7067011985182e-39"
## [3,] "4.63408794691251e-39"
## [4,] "1.53093486122405e-37"
## [5,] "1.53093486122405e-37"
## [6,] "2.43217613294055e-35"
```

For the modules in two different states, it is of interest to see how the genes are overlapped between modules. Therefore, we calculate the number of genes in the intersections and the correspondence = (# of genes in the intersection/(# of genes in a module in state1 + # of genes in a module in state2). We also plot out the labeled heatmap. Numbers in the retangles are the number of genes in the intersections between two corresponding modules. The color is scaled by the correspondence (in the range [0,1]). Number of genes in each modules are attach after the color name.

```
overlap_matrix=wgcnaOverlap(modules1 = modules1$modules,modules2 = modules2$modules)
```

Intergrating the overlapping matrix and the GO enrichment analysis, we came to the conclusion that the module XXX is conserved in two different states. This is verified by the paper XXX.

4 Conclusions

It is hoped that this package will facilitate analysis of psudo temporal odering sigle cell expression data. Its strength is that it can distill co-synconization information into graph clusters that outline the underlying dynamical structure.

5 Acknowledgments

Thanks to colleagues that help in testing this tool.

6 References

Gao, S., J. Hartman, J.L. Carter, M.J. Hessner and X. Wang, Global analysis of phase locking in gene expression during cell cycle: the potential in network modeling. BMC Syst Biol, 2010. 4: p. 167.

Thurman W.N. & Fisher M.E. (1988), Chickens, Eggs, and Causality, or Which Came First?, American Journal of Agricultural Economics, 237-238.

Langfelder, P. & Horvath, S. WGCNA: an R package for weighted correlation network analysis. BMC Bioinformatics 9, 559 (2008)