CBNA: A control theory based method for identifying coding and non-coding cancer drivers

Vu VH Pham[1], Lin Liu[1], Cameron Bracken[2], Greg Goodall[2], Jiuyong Li[1] and Thuc D Le[1]

[1] School of Information Technology and Mathematical Sciences, University of South Australia, Mawson Lakes, 5095, Australia and

[2] Centre for Cancer Biology, SA Pathology, Adelaide, 5000, Australia.

This is the script to run the proposed method. Please remember to include the script of functions from the file ProposedMethod_Functions.R to run this script.

1. CBNA: Controllability based Biological Network Analysis & detecting BRCA drivers

This is the script of the proposed method and its application in detecting cancer drivers. The method is applied to the breast invasive carcinoma (BRCA) dataset of The Cancer Genome Atlas (TCGA) to discover BRCA drivers, including both coding and non-coding drivers.

To run the script, please prepare below input files and reset environment variables in the script.

The input files include the followings and are put in the folder "rootDir/Data" (rootDir is an environment variable):

- PPI.xls - Protein protein interaction network
- Browse Transcription Factors hg19 - resource_browser.csv - Transcription factors (TFs)
- hsa.tsv - TransmiR dataset for TF-miRNA interactions
- miRTarBase_v6.1+TarBase_v7.0+miRWalk_v2.0.csv - Datasets for miRNA-mRNA and miRNA-TF interactions
- TargetScan_7.0.csv - Dataset for miRNA-mRNA and miRNA-TF interactions
- BRCA_matchedData_full.RData - Tumour expression data
- mut.RData - Mutation data
- Census_allFri Sep 28 07_39_37 2018.tsv - Cancer Gene Census (CGC)

This script uses a library from the paper Liu, Y.-Y., et al. (2011). "Controllability of complex networks." Nature 473: 167. The code of the library can be downloaded from https://scholar.harvard.edu/yyl/code. You need to build the code before running this script.

```r
#==========================================================================
#==========================================================================
# CBNA: Controllability based Biological Network Analysis
#==========================================================================
#==========================================================================
# Clear the environment
rm(list = ls())

# Load necessary libraries if any
library(readxl)
library(miRLAB)
library(miRBaseConverter)
library(ggplot2)
library(varhandle)
library(scales)
library(reshape)
library(plyr)
library(RColorBrewer)
library(tidyverse)
library(xtable)

#----------------------------------------
# Set environment variables if any
# Please remember to create necessary folders
rootDir <- "C:/Users/phavy022/MyDoc/05CancerDriver" # And put the input files in "rootDir/Data"
controlDir <- "C:/MinGW/bin" # Put here the library from
  # the paper "Controllability of complex networks."
outDir <- "C:/Users/phavy022/MyDoc/05CancerDriver/Data/Output/CancerDriver/Cancer" # Output folder
#----------------------------------------

# Include the script of functions
source(paste(rootDir, "/Script/ProposedMethod_Functions.R", sep=""))

# Main script - Controllability based Biological Network Analysis (CBNA)
#=================================================================
# (1) Building the network for a specific condition
#=================================================================
# Load the tumor expression data
load(paste(rootDir, "/Data/BRCA_matchedData_full.RData", sep = ""))

# Get PPI network
edges <- read_excel(paste(rootDir, "/Data/PPI.xls",
                          sep = ""), sheet = 1)
interactions <- edges[, c(1, 3)]
colnames(interactions) <- c("cause", "effect")
interactions <- interactions[which(interactions$cause %in% colnames(BRCA_matchedData$mRNAs)),]
interactions <- interactions[which(interactions$effect %in% colnames(BRCA_matchedData$mRNAs)),]
nodes <- unique(union(interactions$cause, interactions$effect))

# TFs: Download the list from http://fantom.gsc.riken.jp/5/sstar/Browse_Transcription_Factors_hg19
tfs <- read.csv(paste(rootDir, "/Data/Browse Transcription Factors hg19 - resource_browser.csv",
                      sep = ""))
i <- which(levels(tfs$Symbol) %in% nodes)
tfData <- BRCA_matchedData$mRNAs[, levels(tfs$Symbol)[i]]

# Update cancer data of mRNAs
BRCA_matchedData$mRNAs <- BRCA_matchedData$mRNAs[,
  nodes[which(!(nodes %in% levels(tfs$Symbol)[i]))]]
mRNAsData_Cancer <-  BRCA_matchedData$mRNAs

# Get the cancer data of miRNAs
miRNAsData_Cancer <-  BRCA_matchedData$miRs

# Combine data
```

```r
nomiR <- ncol(BRCA_matchedData$miRs)
nomR <- ncol(BRCA_matchedData$mRNAs)
noTF <- ncol(tfData)
cancer_data <- cbind(miRNAsData_Cancer, mRNAsData_Cancer, tfData)

# Free the memory
gc()

# Build the network
cancer_network <- buildNetworkWithmiRs(interactions, nomiR, nomR, noTF, cancer_data, rootDir)

# Save the network
write.csv(cancer_network, paste(outDir, "/cancer_network.csv", sep = ""), row.names = FALSE)

# Analyse network
# cancer_network <- read.csv(paste(outDir, "/cancer_network.csv", sep = ""))
analyseNetwork(nomiR, nomR, noTF, cancer_network, cancer_data,
               paste(outDir, "/cancer_network_analysis.txt", sep = ""))

#==================================================================
# (2) Identifying driver profile
#==================================================================
# Load the mutation data
# Read file
load(paste(rootDir, "/Data/mut.RData", sep = ""))
mutData <- mut[, c("gene_name_WU", "Tumor_Sample_Barcode", "trv_type_WU")]
colnames(mutData) <- c("symbol", "Sample", "ct")
# Just get mutations which have effects on proteins
# Refer to the below link for more information
# https://sagebionetworks.jira.com/wiki/spaces/METAGENOMICS/pages/23396441/Curation+of+Mutation+Data
proteinAffectingMut <- mutData[!(mutData$ct %in% c("silent", "rna")),]
proteinAffectingMut$Sample <- substr(proteinAffectingMut$Sample, 1, 12)
# Evaluate the frequency for each mutated gene
proteinAffectingMut$weight <- 0
proteinAffectingMut <- proteinAffectingMut[order(proteinAffectingMut$symbol),]
l <- nrow(proteinAffectingMut)
curGene <- proteinAffectingMut[1,1]
count <- 1
for (i in 2:l) {
  if(i == l) {
    if (proteinAffectingMut[i,1] == curGene) {
      count <- count + 1
    } else {
      count <- 1
    }
    proteinAffectingMut[i,4] <- count
  } else {
    if (proteinAffectingMut[i,1] == curGene) {
      count <- count + 1
    } else {
      proteinAffectingMut[i-1,4] <- count
      curGene <- proteinAffectingMut[i,1]
      count <- 1
    }
  }
}
proteinAffectingMut <- proteinAffectingMut[which(proteinAffectingMut[,4] != 0),]
proteinAffectingMut <- proteinAffectingMut[, c(1,4)]

# Analyse controllability of the network
# Read the network with miRNAs
interactions <- read.csv(paste(outDir, "/cancer_network.csv",
                     sep = ""))
# Write the edges of the network for analysing controllability
write.table(interactions, paste(outDir, "/Controllability/edges.dat", sep = ""),
          row.names = FALSE, col.names=FALSE, quote=FALSE)
# Run the controllability analysis
cmd <- paste(controlDir, "/parse.exe ", outDir,
             "/Controllability/edges.dat", sep = "")
system(cmd)
cmd <- paste(controlDir, "/controllability_analysis.exe ", outDir,
             "/Controllability/edges.dat", sep = "")
system(cmd)
# Analyse controllability of the network and output in a file
analyseControllability(paste(outDir, "/Controllability/edges.dat.output", sep = ""),
  paste(outDir, "/analyseControllability.txt", sep = ""))

# Identify critical nodes in the network
# Read the result
nodetype <- read.table(paste(outDir, "/Controllability/edges.dat.nodetype", sep = ""))
colnames(nodetype) <- c("Name", "K", "Kin", "Kout", "TypeI", "TypeII")
# Critical nodes of the network
critical_nodes <- nodetype[which(nodetype$TypeI == 0),]
# Save file
write.csv(critical_nodes, paste(outDir, "/critical_nodes.csv", sep = ""),
          row.names = FALSE)

# Combine mutation data, rank candidate cancer drivers
# Merge with mutaion proteinAffectingMut
candidate_cancer_drivers <- merge(critical_nodes, proteinAffectingMut,
                        all.x = TRUE, by.x = "Name", by.y = "symbol")
# Classify miRNAs and TFs/genes
candidate_cancer_drivers$Type <- "coding"
candidate_cancer_drivers[which(candidate_cancer_drivers[, "Name"] %in%
                               colnames(cancer_data)[1:nomiR]), "Type"] <- "non-coding"
# Candidate coding cancer drivers
coding_candidate_cancer_drivers <- candidate_cancer_drivers[
  which(candidate_cancer_drivers[, "Type"] == "coding"),]
coding_candidate_cancer_drivers <-
  coding_candidate_cancer_drivers[order(coding_candidate_cancer_drivers$weight, decreasing = TRUE),]
coding_candidate_cancer_drivers_mutations <-
  coding_candidate_cancer_drivers[!(is.na(coding_candidate_cancer_drivers$weight)),]
coding_candidate_cancer_drivers_no_mutations <-
  coding_candidate_cancer_drivers[is.na(coding_candidate_cancer_drivers$weight),]
# Candidate non-coding cancer drivers
noncoding_candidate_cancer_drivers <- candidate_cancer_drivers[
  which(candidate_cancer_drivers[, "Type"] == "non-coding"),]
# Write files
write.csv(coding_candidate_cancer_drivers,
          paste(outDir, "/coding_candidate_cancer_drivers.csv", sep = ""), row.names = FALSE)
write.csv(coding_candidate_cancer_drivers_mutations,
          paste(outDir, "/coding_candidate_cancer_drivers_mutations.csv", sep = ""),
          row.names = FALSE)
write.csv(coding_candidate_cancer_drivers_no_mutations,
          paste(outDir, "/coding_candidate_cancer_drivers_no_mutations.csv", sep = ""),
          row.names = FALSE)
write.csv(noncoding_candidate_cancer_drivers,
          paste(outDir, "/noncoding_candidate_cancer_drivers.csv", sep = ""), row.names = FALSE)

#=================================================================
# Validation
#=================================================================
# Load the gold standard, CGC
gold_standard = read.table(
  file = paste(rootDir, "/Data/Census_allFri Sep 28 07_39_37 2018.tsv", sep = ""),
  sep = '\t', header = TRUE)
gold_standard <- gold_standard[, 1]
gold_standard <- as.character(gold_standard)

# DawnRank
DawnRankDriver = read.table(
  file = paste(rootDir, "/DawnRank/DawnRank-drivers.txt", sep = ""),
  sep = '\t', header = F)
DawnRankDriver_Sorted <- DawnRankDriver[order(DawnRankDriver$V2, decreasing = T),]
DawnRankDriver_Top50 <- DawnRankDriver_Sorted[1:50,]
DawnRankDriver_Top100 <- DawnRankDriver_Sorted[1:100,]
DawnRankDriver_Top150 <- DawnRankDriver_Sorted[1:150,]
DawnRankDriver_Top200 <- DawnRankDriver_Sorted[1:200,]
DawnRankDriver_Top50_Validated <- intersect(DawnRankDriver_Top50[, 1], gold_standard)
DawnRankDriver_Top100_Validated <- intersect(DawnRankDriver_Top100[, 1], gold_standard)
DawnRankDriver_Top150_Validated <- intersect(DawnRankDriver_Top150[, 1], gold_standard)
DawnRankDriver_Top200_Validated <- intersect(DawnRankDriver_Top200[, 1], gold_standard)

# IntOGen
# Load result file
IntOGenResult = read.table(
  file = paste(rootDir, "/OncodriveCLUST_OncodriveFM/2018-09-24_intogen/gene.tsv", sep = ""),
  sep = '\t', header = TRUE)

# OncodriveCLUST
OncodriveCLUST_Sorted <- IntOGenResult[order(IntOGenResult$QVALUE_ONCODRIVECLUST, decreasing = F),]
OncodriveCLUST_Top50 <- OncodriveCLUST_Sorted[1:50,]
OncodriveCLUST_Top100 <- OncodriveCLUST_Sorted[1:100,]
OncodriveCLUST_Top150 <- OncodriveCLUST_Sorted[1:150,]
OncodriveCLUST_Top200 <- OncodriveCLUST_Sorted[1:200,]
OncodriveCLUST_Top50_Validated <- intersect(OncodriveCLUST_Top50[, "SYMBOL"], gold_standard)
OncodriveCLUST_Top100_Validated <- intersect(OncodriveCLUST_Top100[, "SYMBOL"], gold_standard)
OncodriveCLUST_Top150_Validated <- intersect(OncodriveCLUST_Top150[, "SYMBOL"], gold_standard)
OncodriveCLUST_Top200_Validated <- intersect(OncodriveCLUST_Top200[, "SYMBOL"], gold_standard)
```

```
# OncodriveFM
OncodriveFM_Sorted <- IntOGenResult[order(IntOGenResult$QVALUE_ONCODRIVEFM, decreasing = F),]
OncodriveFM_Top50 <- OncodriveFM_Sorted[1:50,]
OncodriveFM_Top100 <- OncodriveFM_Sorted[1:100,]
OncodriveFM_Top150 <- OncodriveFM_Sorted[1:150,]
OncodriveFM_Top200 <- OncodriveFM_Sorted[1:200,]
OncodriveFM_Top50_Validated <- intersect(OncodriveFM_Top50[, "SYMBOL"], gold_standard)
OncodriveFM_Top100_Validated <- intersect(OncodriveFM_Top100[, "SYMBOL"], gold_standard)
OncodriveFM_Top150_Validated <- intersect(OncodriveFM_Top150[, "SYMBOL"], gold_standard)
OncodriveFM_Top200_Validated <- intersect(OncodriveFM_Top200[, "SYMBOL"], gold_standard)

# New method - CBNA
coding_candidate_cancer_drivers = read.csv(
  file = paste(outDir, "/coding_candidate_cancer_drivers_mutations.csv", sep = ""))
CBNA_Top50 <- coding_candidate_cancer_drivers[1:50,]
CBNA_Top100 <- coding_candidate_cancer_drivers[1:100,]
CBNA_Top150 <- coding_candidate_cancer_drivers[1:150,]
CBNA_Top200 <- coding_candidate_cancer_drivers[1:200,]
CBNA_Top50_Validated <- intersect(CBNA_Top50[, "Name"], gold_standard)
CBNA_Top100_Validated <- intersect(CBNA_Top100[, "Name"], gold_standard)
CBNA_Top150_Validated <- intersect(CBNA_Top150[, "Name"], gold_standard)
CBNA_Top200_Validated <- intersect(CBNA_Top200[, "Name"], gold_standard)

# ActiveDriver
active_drivers = read.csv(
  file = paste(rootDir, "/ActiveDriver/brca_results_pvals.csv", sep = ""))
active_Sorted <- active_drivers[order(active_drivers$p, decreasing = F),]
active_Top50 <- active_Sorted[1:50,]
active_Top100 <- active_Sorted[1:100,]
active_Top150 <- active_Sorted[1:150,]
active_Top200 <- active_Sorted[1:200,]
active_Top50_Validated <- intersect(active_Top50[, "gene"], gold_standard)
active_Top100_Validated <- intersect(active_Top100[, "gene"], gold_standard)
active_Top150_Validated <- intersect(active_Top150[, "gene"], gold_standard)
active_Top200_Validated <- intersect(active_Top200[, "gene"], gold_standard)

# DriverNet
net_drivers = read.csv(
  file = paste(rootDir, "/DriverNet/res.csv", sep = ""))
net_Sorted <- net_drivers[order(net_drivers$p.value, decreasing = F),]
net_Top50 <- net_Sorted[1:50,]
net_Top100 <- net_Sorted[1:100,]
net_Top150 <- net_Sorted[1:150,]
net_Top200 <- net_Sorted[1:200,]
net_Top50_Validated <- intersect(net_Top50[, "gene"], gold_standard)
net_Top100_Validated <- intersect(net_Top100[, "gene"], gold_standard)
net_Top150_Validated <- intersect(net_Top150[, "gene"], gold_standard)
net_Top200_Validated <- intersect(net_Top200[, "gene"], gold_standard)

# # 1. Graph to compare methods using CGC
# tops <-rep(c("Top 50", "Top 100", "Top 150", "Top 200"), 4)
# OncodriveCLUST <- c(length(OncodriveCLUST_Top50_Validated),
#                     length(OncodriveCLUST_Top100_Validated),
#                      length(OncodriveCLUST_Top150_Validated),
#                      length(OncodriveCLUST_Top200_Validated))
# OncodriveFM <- c(length(OncodriveFM_Top50_Validated),
#                   length(OncodriveFM_Top100_Validated),
#                   length(OncodriveFM_Top150_Validated),
#                   length(OncodriveFM_Top200_Validated))
# DawnRank <- c(length(DawnRankDriver_Top50_Validated),
#               length(DawnRankDriver_Top100_Validated),
#               length(DawnRankDriver_Top150_Validated),
#               length(DawnRankDriver_Top200_Validated))
# CBNA <- c(length(CBNA_Top50_Validated),
#                      length(CBNA_Top100_Validated),
#                      length(CBNA_Top150_Validated),
#                      length(CBNA_Top200_Validated))
# values <-c(OncodriveCLUST, OncodriveFM, DawnRank, CBNA)
# type <-c(rep("OncodriveCLUST", 4), rep("OncodriveFM", 4), rep("DawnRank", 4), rep("CBNA", 4))
# mydata <-data.frame(tops, values)
# mydata$tops <- factor(mydata$tops,
#                       levels = c("Top 50", "Top 100", "Top 150", "Top 200"))
# type <-factor(type, levels = c("OncodriveCLUST", "OncodriveFM", "DawnRank", "CBNA"))
# p <- ggplot(mydata, aes(tops, values))
# p + geom_bar(stat = "identity", aes(fill = type), position = "dodge") + scale_fill_manual("",
#   values = c("OncodriveCLUST" = "#9d5257", "OncodriveFM" = "#0079bf",
#              "DawnRank" = "#44d9e6", "CBNA" = "#9e93b8")) +
#   labs(title= "", y= "", x = "") + theme(text = element_text(size=24))

# 1. Graph to compare methods using CGC (6 methods)
tops <-rep(c("Top 50", "Top 100", "Top 150", "Top 200"), 6)
OncodriveCLUST <- c(length(OncodriveCLUST_Top50_Validated),
                    length(OncodriveCLUST_Top100_Validated),
                    length(OncodriveCLUST_Top150_Validated),
                    length(OncodriveCLUST_Top200_Validated))
OncodriveFM <- c(length(OncodriveFM_Top50_Validated),
                 length(OncodriveFM_Top100_Validated),
                 length(OncodriveFM_Top150_Validated),
                 length(OncodriveFM_Top200_Validated))
DawnRank <- c(length(DawnRankDriver_Top50_Validated),
              length(DawnRankDriver_Top100_Validated),
              length(DawnRankDriver_Top150_Validated),
              length(DawnRankDriver_Top200_Validated))
CBNA <- c(length(CBNA_Top50_Validated),
                    length(CBNA_Top100_Validated),
                    length(CBNA_Top150_Validated),
                    length(CBNA_Top200_Validated))
ActiveDriver <- c(length(active_Top50_Validated),
                    length(active_Top100_Validated),
                    length(active_Top150_Validated),
                    length(active_Top200_Validated))
DriverNet <- c(length(net_Top50_Validated),
                    length(net_Top100_Validated),
                    length(net_Top150_Validated),
                    length(net_Top200_Validated))
values <-c(OncodriveCLUST, ActiveDriver, OncodriveFM, DriverNet, DawnRank, CBNA)
```

```r
type <-c(rep("OncodriveCLUST", 4), rep("ActiveDriver", 4), rep("OncodriveFM", 4),
         rep("DriverNet", 4), rep("DawnRank", 4), rep("CBNA", 4))
mydata <-data.frame(tops, values)
mydata$tops <- factor(mydata$tops,
                      levels = c("Top 50", "Top 100", "Top 150", "Top 200"))
type <-factor(type, levels = c("OncodriveCLUST", "ActiveDriver", "OncodriveFM",
                               "DriverNet", "DawnRank", "CBNA"))
p <- ggplot(mydata, aes(tops, values))
p + geom_bar(stat = "identity", aes(fill = type), position = "dodge") + scale_fill_manual("",
  values = c("OncodriveCLUST" = "#7efaff", "ActiveDriver" = "#13abc4", "OncodriveFM" = "#3161a3",
             "DriverNet" = "#cdffeb", "DawnRank" = "#009f9d", "CBNA" = "#07456f")) +
  labs(title= "", y= "", x = "") + theme(text = element_text(size=24))

# # 2. Graph to find the overlaps between different methods
# # Prepare the data for the graph
# print("Top 50")
# printGeneList(OncodriveCLUST_Top50_Validated, OncodriveFM_Top50_Validated,
#               DawnRankDriver_Top50_Validated, CBNA_Top50_Validated)
# print("Top 100")
# printGeneList(OncodriveCLUST_Top100_Validated, OncodriveFM_Top100_Validated,
#               DawnRankDriver_Top100_Validated, CBNA_Top100_Validated)
# print("Top 150")
# printGeneList(OncodriveCLUST_Top150_Validated, OncodriveFM_Top150_Validated,
#               DawnRankDriver_Top150_Validated, CBNA_Top150_Validated)
# print("Top 200")
# printGeneList(OncodriveCLUST_Top200_Validated, OncodriveFM_Top200_Validated,
#               DawnRankDriver_Top200_Validated, CBNA_Top200_Validated)

# 2. Graph to find the overlaps between different methods (6 methods)
# Prepare the data for the graph
print("Top 50")
printGeneList_6methods(OncodriveCLUST_Top50_Validated,
                       active_Top50_Validated, OncodriveFM_Top50_Validated,
              net_Top50_Validated, DawnRankDriver_Top50_Validated, CBNA_Top50_Validated)
print("Top 100")
printGeneList_6methods(OncodriveCLUST_Top100_Validated,
                       active_Top100_Validated, OncodriveFM_Top100_Validated,
              net_Top100_Validated, DawnRankDriver_Top100_Validated, CBNA_Top100_Validated)
print("Top 150")
printGeneList_6methods(OncodriveCLUST_Top150_Validated,
                       active_Top150_Validated, OncodriveFM_Top150_Validated,
              net_Top150_Validated, DawnRankDriver_Top150_Validated, CBNA_Top150_Validated)
print("Top 200")
printGeneList_6methods(OncodriveCLUST_Top200_Validated,
                       active_Top200_Validated, OncodriveFM_Top200_Validated,
              net_Top200_Validated, DawnRankDriver_Top200_Validated, CBNA_Top200_Validated)

# # 3. Graph for Precision, Recall, and F1 Score
# # OncodriveCLUST
# OncodriveCLUSTData <- prepareDataForComparison(OncodriveCLUST_Top200_Validated,
#                                           unfactor(OncodriveCLUST_Top200$SYMBOL),
#                                           gold_standard)
# # OncodriveFM
# OncodriveFMData <- prepareDataForComparison(OncodriveFM_Top200_Validated,
#                                           unfactor(OncodriveFM_Top200$SYMBOL),
#                                           gold_standard)
# # DawnRank
# DawnRankData <- prepareDataForComparison(DawnRankDriver_Top200_Validated,
#                                           unfactor(DawnRankDriver_Top200$V1),
#                                           gold_standard)
# # CBNA
# CBNAData <- prepareDataForComparison(CBNA_Top200_Validated,
#                                           unfactor(CBNA_Top200$Name),
#                                           gold_standard)
# # Graph
# # Create Line Chart
# # Precision
# drawLineChart(OncodriveCLUSTData[,3], OncodriveFMData[,3], DawnRankData[,3],
#               CBNAData[,3], Type="Precision")
# # Recall
# drawLineChart(OncodriveCLUSTData[,4], OncodriveFMData[,4], DawnRankData[,4],
#               CBNAData[,4], Type="Recall")
# # F1 Score
# drawLineChart(OncodriveCLUSTData[,5], OncodriveFMData[,5], DawnRankData[,5],
#               CBNAData[,5], Type="F1Score")

# 3. Graph for Precision, Recall, and F1 Score (6 methods)
# OncodriveCLUST
OncodriveCLUSTData <- prepareDataForComparison(OncodriveCLUST_Top200_Validated,
                                        unfactor(OncodriveCLUST_Top200$SYMBOL),
                                        gold_standard)
# OncodriveFM
OncodriveFMData <- prepareDataForComparison(OncodriveFM_Top200_Validated,
                                        unfactor(OncodriveFM_Top200$SYMBOL),
                                        gold_standard)
# DawnRank
DawnRankData <- prepareDataForComparison(DawnRankDriver_Top200_Validated,
                                        unfactor(DawnRankDriver_Top200$V1),
                                        gold_standard)
# CBNA
CBNAData <- prepareDataForComparison(CBNA_Top200_Validated,
                                        unfactor(CBNA_Top200$Name),
                                        gold_standard)

# ActiveDriver
ActiveDriverData <- prepareDataForComparison(active_Top200_Validated,
                                        unfactor(active_Top200$gene),
                                        gold_standard)

# DriverNet
DriverNetData <- prepareDataForComparison(net_Top200_Validated,
                                        unfactor(net_Top200$gene),
                                        gold_standard)
# Graph
# Create Line Chart
# Precision
```

```r
drawLineChart6(OncodriveCLUSTData[,3], ActiveDriverData[,3], OncodriveFMData[,3],
               DriverNetData[,3], DawnRankData[,3],
              CBNAData[,3], Type="Precision")
# Recall
drawLineChart6(OncodriveCLUSTData[,4], ActiveDriverData[,4], OncodriveFMData[,4],
               DriverNetData[,4], DawnRankData[,4],
              CBNAData[,4], Type="Recall")
# F1 Score
drawLineChart6(OncodriveCLUSTData[,5], ActiveDriverData[,5], OncodriveFMData[,5],
               DriverNetData[,5], DawnRankData[,5],
              CBNAData[,5], Type="F1Score")

# 4. Pie chart for coding drivers with/without mutations and non-coding drivers
# Pie Chart with Percentages
coding_candidate_cancer_drivers_mutations = read.csv(
  file = paste(outDir, "/coding_candidate_cancer_drivers_mutations.csv", sep = ""))
coding_candidate_cancer_drivers_no_mutations = read.csv(
  file = paste(outDir, "/coding_candidate_cancer_drivers_no_mutations.csv", sep = ""))
noncoding_candidate_cancer_drivers = read.csv(
  file = paste(outDir, "/noncoding_candidate_cancer_drivers.csv", sep = ""))
slices <- c(nrow(coding_candidate_cancer_drivers_mutations),
            nrow(coding_candidate_cancer_drivers_no_mutations),
            nrow(noncoding_candidate_cancer_drivers))
pct <- round(slices/sum(slices)*100, digits = 1)
lbls <- c("Coding drivers with mutations", "Coding drivers without mutations",
          "Non-coding drivers")
print(lbls[1])
print(pct[1])
print(lbls[2])
print(pct[2])
print(lbls[3])
print(pct[3])

# 5. Enrichment analysis
# Biological process
GO_process <- read.table(paste(outDir, "/GO_Biological_Process_2018_table.txt", sep=""),
                         as.is = TRUE, sep = "\t", header = TRUE)
GO_process <- GO_process[, c(1,2,3,9)]
colnames(GO_process) <- c("Term", "Overlap", "P-value", "Genes")
GO_process <- GO_process[which(GO_process[,"P-value"] < 0.05),]
GO_process <- GO_process[order(GO_process$`P-value`, decreasing = FALSE),]
write.csv(GO_process, file = paste(outDir, "/GO_Biological_Process_2018_table_Cutoff.csv",
                                   sep=""), row.names = FALSE, quote=TRUE)
r <- prepareDataForClustergram(GO_process, termTop = 10, geneTop  = 20)
write.csv(r, file = paste(outDir, "/GO_Biological_Process_2018_table_Heatmap.csv",
                                   sep=""), row.names = FALSE, quote=TRUE)
dat <- read.csv(file = paste(outDir, "/GO_Biological_Process_2018_table_Heatmap.csv", sep=""))
colnames(dat) <- gsub("[.]",":",colnames(dat))
drawClustergram(dat, "GO Biological Process")

# Molecular function
GO_function <- read.table(paste(outDir, "/GO_Molecular_Function_2018_table.txt", sep=""),
                         as.is = TRUE, sep = "\t", header = TRUE)
```

```r
GO_function <- GO_function[, c(1,2,3,9)]
colnames(GO_function) <- c("Term", "Overlap", "P-value", "Genes")
GO_function <- GO_function[which(GO_function[,"P-value"] < 0.05),]
GO_function <- GO_function[order(GO_function$`P-value`, decreasing = FALSE),]
write.csv(GO_function, file = paste(outDir, "/GO_Molecular_Function_2018_table_Cutoff.csv",
                                    sep=""), row.names = FALSE, quote=TRUE)
# As GO:0005158 has no gene in the top 20 genes, remove it
r <- prepareDataForClustergram(GO_function, termTop = 11, geneTop  = 20)
r <- r[,-which(colnames(r) == "GO:0005158")]
write.csv(r, file = paste(outDir, "/GO_Molecular_Function_2018_table_Heatmap.csv",
                                    sep=""), row.names = FALSE, quote=TRUE)
dat <- read.csv(file = paste(outDir, "/GO_Molecular_Function_2018_table_Heatmap.csv", sep=""))
colnames(dat) <- gsub("[.]",":",colnames(dat))
drawClustergram(dat, "GO Molecular Function")

# 6. Demo heatmaps
# miRNA
set.seed(12343)
# create matrix with 12 columns and 9 rows
# with numbers between 0 - 16
AS <- runif(n = 12*9, min = 0, max = 16) %>%
  matrix(., nrow = 9)

colnames(AS) <- LETTERS[1:12]

AS <- as.data.frame(AS)
AS$Samples <- 1:9

AS <- gather(AS, A:L, key = "miRNA", value="value")

base_size <- 15

ggplot(AS, aes(x = miRNA, y=Samples)) +
  geom_tile(aes(fill = value), colour = "white", size=2) +
  scale_fill_distiller(palette = "Reds", limits = c(0,10), na.value = "#CCCCCC") +
  theme(legend.position="none") +
  scale_x_discrete(labels=rep("", 12)) +
  scale_y_discrete(labels=rep("", 9)) +
  theme(axis.title = element_text(size=base_size*2),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.border = element_blank(),
        panel.background = element_blank(),
        axis.text = element_blank(),
        axis.ticks = element_blank(),
        axis.title.x = element_blank(),
        axis.title.y = element_blank())

# TF/mRNA
set.seed(1)
# create matrix with 12 columns and 9 rows
# with numbers between 0 - 16
AS <- runif(n = 12*9, min = 0, max = 16) %>%
  matrix(., nrow = 9)

colnames(AS) <- LETTERS[1:12]

AS <- as.data.frame(AS)
AS$Samples <- 1:9

AS <- gather(AS, A:L, key = "mRNA", value="value")

base_size <- 15

ggplot(AS, aes(x = mRNA, y=Samples)) +
  geom_tile(aes(fill = value), colour = "white", size=2) +
  scale_fill_distiller(palette = "Blues", limits = c(0,10), na.value = "#CCCCCC") +
  theme(legend.position="none") +
  scale_x_discrete(labels=rep("", 12)) +
  scale_y_discrete(labels=rep("", 9)) +
  theme(axis.title = element_text(size=base_size*2),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.border = element_blank(),
        panel.background = element_blank(),
        axis.text = element_blank(),
        axis.ticks = element_blank(),
        axis.title.x = element_blank(),
        axis.title.y = element_blank())

# 7. Print table
# coding drivers with mutations
xyz <- data.frame(lapply(CBNA_Top200, as.character), stringsAsFactors=FALSE)
xyz <- xyz[,1:2]
colnames(xyz) <- c("Predicted driver", "In CGC")
xyz$`In CGC` <- "FALSE"
for (i in 1:200) {
  if(xyz[i,1] %in% gold_standard) {
    xyz[i,2] <- "TRUE"
  }
}
xyz <- data.frame(xyz)
print(xtable(xyz, display = c("d","s","s"), digits = 0), format.args = list(big.mark = " ",
                                                                        decimal.mark = "."))

# coding drivers without mutations
coding_candidate_cancer_drivers_no_mutations = read.csv(
  file = paste(outDir, "/coding_candidate_cancer_drivers_no_mutations.csv", sep = ""))
xyz <- coding_candidate_cancer_drivers_no_mutations
xyz <- data.frame(lapply(xyz, as.character), stringsAsFactors=FALSE)
xyz <- matrix(xyz[,1], ncol = 1)
colnames(xyz) <- c("Predicted driver")
xyz <- data.frame(xyz)
print(xtable(xyz, display = c("d","s"), digits = 0), format.args = list(big.mark = " ",
                                                                    decimal.mark = "."))
```

```r
# non-coding drivers
noncoding_candidate_cancer_drivers = read.csv(
  file = paste(outDir, "/noncoding_candidate_cancer_drivers.csv", sep = ""))
xyz <- noncoding_candidate_cancer_drivers
xyz <- data.frame(lapply(xyz, as.character), stringsAsFactors=FALSE)
xyz <- matrix(xyz[,1], ncol = 1)
colnames(xyz) <- c("Predicted driver")
xyz <- data.frame(xyz)
print(xtable(xyz, display = c("d","s"), digits = 0), format.args = list(big.mark = " ",
                                                                          decimal.mark = "."))

# 8. Number of predicted drivers, Fraction of predicted drivers in CGC

daw <- DawnRankDriver_Sorted
clu <- OncodriveCLUST_Sorted[which(OncodriveCLUST_Sorted$QVALUE_ONCODRIVECLUST <= 0.1),]
fm <- OncodriveFM_Sorted[which(OncodriveFM_Sorted$QVALUE_ONCODRIVEFM <= 0.1),]
cbna <- coding_candidate_cancer_drivers
act <- active_Sorted[which(active_Sorted$p <= 0.05),]
net <- net_Sorted[which(net_Sorted$p.value <= 0.05),]

n_daw <- nrow(daw)
n_clu <- nrow(clu)
n_fm <- nrow(fm)
n_cbna <- nrow(cbna)
n_act <- nrow(act)
n_net <- nrow(net)

daw_Validated <- intersect(daw$V1, gold_standard)
clu_Validated <- intersect(clu$SYMBOL, gold_standard)
fm_Validated <- intersect(fm$SYMBOL, gold_standard)
cbna_Validated <- intersect(cbna$Name, gold_standard)
act_Validated <- intersect(act$gene, gold_standard)
net_Validated <- intersect(net$gene, gold_standard)

n_daw_validated <- length(daw_Validated)
n_clu_validated <- length(clu_Validated)
n_fm_validated <- length(fm_Validated)
n_cbna_validated <- length(cbna_Validated)
n_act_validated <- length(act_Validated)
n_net_validated <- length(net_Validated)

f_daw <- n_daw_validated/n_daw
f_clu <- n_clu_validated/n_clu
f_fm <- n_fm_validated/n_fm
f_cbna <- n_cbna_validated/n_cbna
f_act <- n_act_validated/n_act
f_net <- n_net_validated/n_net

# Chart of Number of predicted drivers
par(mar = c(9, 6, 3, 3) + 0.2) # add room for the rotated labels
dat <- data.frame("no" = c(n_clu, n_act, n_fm, n_net, n_daw, n_cbna))
rownames(dat) <- c("OncodriveCLUST  ", "ActiveDriver  ", "OncodriveFM  ",
                   "DriverNet  ", "DawnRank  ", "CBNA  ")
```

```r
end_point = 0.5 + nrow(dat) + nrow(dat)/2 - 1 # this is the line
  # which does the trick (together with barplot "space = 0.5" parameter)
barplot(dat$no, col="grey50",
        main="",
        ylab="Number of predicted driver genes", ylim=c(0,100+max(dat$no)),
        xlab = "",
        space=0.5, cex.axis = 1.5, cex.lab = 1.5)
# rotate 60 degrees, srt=60
text(seq(1,end_point,by=1.5), par("usr")[3]-0.25,
     srt = 60, adj= 1, xpd = TRUE,
     labels = paste(rownames(dat)), cex=1.5)

# Chart of Fraction of predicted drivers in CGC
par(mar = c(9, 6, 3, 3) + 0.2) # add room for the rotated labels
dat <- data.frame("no" = c(f_clu, f_act, f_fm, f_net, f_daw, f_cbna))
rownames(dat) <- c("OncodriveCLUST  ", "ActiveDriver  ", "OncodriveFM  ",
                   "DriverNet  ", "DawnRank  ", "CBNA  ")
end_point = 0.5 + nrow(dat) + nrow(dat)/2 - 1 # this is the line
  # which does the trick (together with barplot "space = 0.5" parameter)
barplot(dat$no, col="grey50",
        main="",
        ylab="Fraction of predicted driver genes in CGC", ylim=c(0,0.1+max(dat$no)),
        xlab = "",
        space=0.5, cex.axis = 1.5, cex.lab = 1.5)
# rotate 60 degrees, srt=60
text(seq(1,end_point,by=1.5), par("usr")[3],
     srt = 60, adj= 1, xpd = TRUE,
     labels = paste(rownames(dat)), cex=1.5)
```

2. Identifying drivers in different conditions

This is the script to detect candidate cancer drivers which are in the network built from the cancer data but not in the network built from the normal data. The method is applied to the BRCA dataset.

To run the script, please prepare below input files and reset environment variables in the script.

The input files include the followings and are put in the folder "rootDir/Data" (rootDir is an environment variable):

- PPI.xls - Protein protein interaction network
- Browse Transcription Factors hg19 - resource_browser.csv - Transcription factors (TFs)
- hsa.tsv - TransmiR dataset for TF-miRNA interactions
- miRTarBase_v6.1+TarBase_v7.0+miRWalk_v2.0.csv - Datasets for miRNA-mRNA and miRNA-TF interactions
- TargetScan_7.0.csv - Dataset for miRNA-mRNA and miRNA-TF interactions
- BRCA_matchedData_normal_samples_full.RData - Normal expression data
- Census_allFri Sep 28 07_39_37 2018.tsv - Cancer Gene Census (CGC)

You need to run the script of "1. CBNA: Controllability based Biological Network Analysis & detecting BRCA drivers" before running this script.

```r
#==========================================================================
#==========================================================================
# Identifying drivers in different conditions
#==========================================================================
#==========================================================================
# Clear the environment
rm(list = ls())

# Load necessary libraries if any
library(readxl)
library(miRLAB)
library(miRBaseConverter)

#---------------------------------------
# Set environment variables if any
# Please remember to create necessary folders
rootDir <- "C:/Users/phavy022/MyDoc/05CancerDriver" # And put the input files in "rootDir/Data"
controlDir <- "C:/MinGW/bin" # Put here the library from
  # the paper "Controllability of complex networks."
outDir <- "C:/Users/phavy022/MyDoc/05CancerDriver/Data/Output/CancerDriver/Normal" # Output folder
outDirForCancer <-
  "C:/Users/phavy022/MyDoc/05CancerDriver/Data/Output/CancerDriver/Cancer" # Output folder of cancer
#---------------------------------------

# Include the script of functions
source(paste(rootDir, "/Script/ProposedMethod_Functions.R", sep=""))

# Main script - Identifying drivers in different conditions
#=================================================================
# (1) Building the network for a specific condition
#=================================================================
# Load the normal expression data
load(paste(rootDir, "/Data/BRCA_matchedData_normal_samples_full.RData", sep = ""))
BRCA_matchedData <- BRCA_matchedData_normal_samples

# Get PPI network
edges <- read_excel(paste(rootDir, "/Data/PPI.xls",
                          sep = ""), sheet = 1)
interactions <- edges[, c(1, 3)]
colnames(interactions) <- c("cause", "effect")
interactions <- interactions[which(interactions$cause %in% colnames(BRCA_matchedData$mRNAs)),]
interactions <- interactions[which(interactions$effect %in% colnames(BRCA_matchedData$mRNAs)),]
nodes <- unique(union(interactions$cause, interactions$effect))

# TFs: Download the list from http://fantom.gsc.riken.jp/5/sstar/Browse_Transcription_Factors_hg19
tfs <- read.csv(paste(rootDir, "/Data/Browse Transcription Factors hg19 - resource_browser.csv",
                      sep = ""))
i <- which(levels(tfs$Symbol) %in% nodes)
tfData <- BRCA_matchedData$mRNAs[, levels(tfs$Symbol)[i]]

# Update data of mRNAs
BRCA_matchedData$mRNAs <- BRCA_matchedData$mRNAs[,
  nodes[which(!(nodes %in% levels(tfs$Symbol)[i]))]]
```

```r
mRNAsData <-  BRCA_matchedData$mRNAs

# Get the data of miRNAs
miRNAsData <-  BRCA_matchedData$miRs

# Combine data
nomiR <- ncol(BRCA_matchedData$miRs)
nomR <- ncol(BRCA_matchedData$mRNAs)
noTF <- ncol(tfData)
data <- cbind(miRNAsData, mRNAsData, tfData)

# Free the memory
gc()

# Build the network
network <- buildNetworkWithmiRs(interactions, nomiR, nomR, noTF, data, rootDir)

# Save the network
write.csv(network, paste(outDir, "/normal_network.csv", sep = ""), row.names = FALSE)

# Analyse network
# network <- read.csv(paste(outDir, "/normal_network.csv", sep = ""))
analyseNetwork(nomiR, nomR, noTF, network, data,
              paste(outDir, "/normal_network_analysis.txt", sep = ""))

#=================================================================
# (2) Identifying driver profile
#=================================================================
# Load the mutation data
# Read file
load(paste(rootDir, "/Data/mut.RData", sep = ""))
mutData <- mut[, c("gene_name_WU", "Tumor_Sample_Barcode", "trv_type_WU")]
colnames(mutData) <- c("symbol", "Sample", "ct")
# Just get mutations which have effects on proteins
# Refer to the below link for more information
# https://sagebionetworks.jira.com/wiki/spaces/METAGENOMICS/pages/23396441/Curation+of+Mutation+Data
proteinAffectingMut <- mutData[!(mutData$ct %in% c("silent", "rna")),]
proteinAffectingMut$Sample <- substr(proteinAffectingMut$Sample, 1, 12)
# Evaluate the frequency for each mutated gene
proteinAffectingMut$weight <- 0
proteinAffectingMut <- proteinAffectingMut[order(proteinAffectingMut$symbol),]
l <- nrow(proteinAffectingMut)
curGene <- proteinAffectingMut[1,1]
count <- 1
for (i in 2:l) {
  if(i == l) {
    if (proteinAffectingMut[i,1] == curGene) {
      count <- count + 1
    } else {
      count <- 1
    }
    proteinAffectingMut[i,4] <- count
  } else {
    if (proteinAffectingMut[i,1] == curGene) {
      count <- count + 1
    } else {
      proteinAffectingMut[i-1,4] <- count
      curGene <- proteinAffectingMut[i,1]
      count <- 1
    }
  }
}
proteinAffectingMut <- proteinAffectingMut[which(proteinAffectingMut[,4] != 0),]
proteinAffectingMut <- proteinAffectingMut[, c(1,4)]

# Analyse controllability of the network
# Read the network with miRNAs
interactions <- read.csv(paste(outDir, "/normal_network.csv",
                      sep = ""))
# Write the edges of the network for analysing controllability
write.table(interactions, paste(outDir, "/Controllability/edges.dat", sep = ""),
          row.names = FALSE, col.names=FALSE, quote=FALSE)
# Run the controllability analysis
cmd <- paste(controlDir, "/parse.exe ", outDir,
             "/Controllability/edges.dat", sep = "")
system(cmd)
cmd <- paste(controlDir, "/controllability_analysis.exe ", outDir,
             "/Controllability/edges.dat", sep = "")
system(cmd)
# Analyse controllability of the network and output in a file
analyseControllability(paste(outDir, "/Controllability/edges.dat.output", sep = ""),
  paste(outDir, "/analyseControllability.txt", sep = ""))

# Identify critical nodes in the network
# Read the result
nodetype <- read.table(paste(outDir, "/Controllability/edges.dat.nodetype", sep = ""))
colnames(nodetype) <- c("Name", "K", "Kin", "Kout", "TypeI", "TypeII")
# Critical nodes of the network
critical_nodes <- nodetype[which(nodetype$TypeI == 0),]
# Save file
write.csv(critical_nodes, paste(outDir, "/critical_nodes.csv", sep = ""),
          row.names = FALSE)

# Compare the results of cancer and normal, identify candidate cancer drivers
# Read the result
critical_nodes_cancer <- read.table(paste(outDirForCancer, "/critical_nodes.csv", sep = ""),
                                    header = TRUE, sep = ",")
# Critical nodes of the network of cancer samples and not in the network of normal samples
cancer_only_critical_nodes <-
  critical_nodes_cancer[which(!(critical_nodes_cancer[, 1] %in% critical_nodes$Name)),]

# Combine mutation data, rank candidate cancer drivers
# Merge with mutaion proteinAffectingMut
candidate_cancer_drivers <- merge(cancer_only_critical_nodes, proteinAffectingMut,
                                  all.x = TRUE, by.x = "Name", by.y = "symbol")
# Classify miRNAs and TFs/genes
```

```r
candidate_cancer_drivers$Type <- "coding"
candidate_cancer_drivers[which(candidate_cancer_drivers[, "Name"] %in%
                                 colnames(data)[1:nomiR]), "Type"] <- "non-coding"
# Candidate coding cancer drivers
coding_candidate_cancer_drivers <- candidate_cancer_drivers[
  which(candidate_cancer_drivers[, "Type"] == "coding"),]
coding_candidate_cancer_drivers <-
  coding_candidate_cancer_drivers[order(coding_candidate_cancer_drivers$weight, decreasing = TRUE),]
coding_candidate_cancer_drivers_mutations <-
  coding_candidate_cancer_drivers[!(is.na(coding_candidate_cancer_drivers$weight)),]
coding_candidate_cancer_drivers_no_mutations <-
  coding_candidate_cancer_drivers[is.na(coding_candidate_cancer_drivers$weight),]
# Candidate non-coding cancer drivers
noncoding_candidate_cancer_drivers <- candidate_cancer_drivers[
  which(candidate_cancer_drivers[, "Type"] == "non-coding"),]
# Write files
write.csv(coding_candidate_cancer_drivers,
          paste(outDir, "/coding_candidate_cancer_drivers.csv", sep = ""), row.names = FALSE)
write.csv(coding_candidate_cancer_drivers_mutations,
          paste(outDir, "/coding_candidate_cancer_drivers_mutations.csv", sep = ""),
          row.names = FALSE)
write.csv(coding_candidate_cancer_drivers_no_mutations,
          paste(outDir, "/coding_candidate_cancer_drivers_no_mutations.csv", sep = ""),
          row.names = FALSE)
write.csv(noncoding_candidate_cancer_drivers,
          paste(outDir, "/noncoding_candidate_cancer_drivers.csv", sep = ""), row.names = FALSE)

#=================================================================
# Validation
#=================================================================
# Load the gold standard, CGC
gold_standard = read.table(
  file = paste(rootDir, "/Data/Census_allFri Sep 28 07_39_37 2018.tsv", sep = ""),
  sep = '\t', header = TRUE)
gold_standard <- gold_standard[, 1]
gold_standard <- as.character(gold_standard)

# New method - CBNA
coding_candidate_cancer_drivers_mutations = read.csv(
  file = paste(outDir, "/coding_candidate_cancer_drivers_mutations.csv", sep = ""))
CBNA_Validated <- intersect(coding_candidate_cancer_drivers_mutations[, "Name"], gold_standard)

# Calculate p_value
n_A = nrow(coding_candidate_cancer_drivers_mutations) # number of estimated cancer drivers
n_B = length(gold_standard) # number of confirmed cancer drivers
n_C = 839+5168 # number of genes
n_A_B = length(CBNA_Validated) # number of cancer drivers validated
p_value <- 1 - phyper(n_A_B, n_B, n_C-n_B, n_A)
print(paste("p_value: ", p_value, sep = ""))
```

3. Detecting drivers of cancer subtypes

This is the script to detect drivers for cancer subtypes. The method is applied to the BRCA dataset.

To run the script, please prepare below input files and reset environment variables in the script.

The input files include the followings and are put in the folder "rootDir/Data" (rootDir is an environment variable):

- PPI.xls - Protein protein interaction network
- Browse Transcription Factors hg19 - resource_browser.csv - Transcription factors (TFs)
- hsa.tsv - TransmiR dataset for TF-miRNA interactions
- miRTarBase_v6.1+TarBase_v7.0+miRWalk_v2.0.csv - Datasets for miRNA-mRNA and miRNA-TF interactions
- TargetScan_7.0.csv - Dataset for miRNA-mRNA and miRNA-TF interactions
- BRCA_matchedData_full.RData - Tumour expression data

Before identifying drivers for cancer subtypes, we need to run the below script first to classify the BRCA dataset into different subtypes. We need to put the library folder bioclassifier_R in "rootDir/Script".

```r
# Reset the environment
rm(list = ls())

# Load libraries
library(AER)
library(CancerSubtypes)
library(miRBaseConverter)
library(miRLAB)
library(genefu)
library(ctc)
library(heatmap.plus)

#---------------------------------------
# Set environment variables here
directoryPath <- "C:/Users/phavy022/MyDoc/05CancerDriver"
dataDir = "bioclassifier_data" # Create this folder in "directoryPath/Data/Output"
inputFileName = "inputFile.txt" # Just a file created during the process
#---------------------------------------

source(paste(directoryPath, "/Script/ProposedMethod_Functions.R", sep=""))

# 1. Get data set
load(paste(directoryPath, "/Data/BRCA_matchedData_full.RData", sep=""))

# 2. Classify samples based on cancer subtypes using Pam50
# Get 50 mRNAs from Pam50
data(pam50)
str(pam50)
fiftymRNAs <- pam50$centroids.map$probe.centroids
# In BRCA_matchedData$mRNAs, there are not CDCA1, KNTC2, ORC6L which are in Pam50
# However, CDCA1 ~ NUF2, KNTC2 ~ NDC80, ORC6L ~ ORC6
# In BRCA_matchedData$mRNAs, replace NUF2 by CDCA1, NDC80 by KNTC2, ORC6 by ORC6L
matchedData <- BRCA_matchedData
colnames(matchedData$mRNAs)[which(colnames(matchedData$mRNAs) == "NUF2")] <- "CDCA1"
colnames(matchedData$mRNAs)[which(colnames(matchedData$mRNAs) == "NDC80")] <- "KNTC2"
colnames(matchedData$mRNAs)[which(colnames(matchedData$mRNAs) == "ORC6")] <- "ORC6L"
# Get data of 50 mRNAs of Pam50 from matchedData
fiftymRNAsData <- matchedData$mRNAs[, fiftymRNAs]
prepareData(fiftymRNAsData, paste(directoryPath, "/Data/Output", sep = ""), dataDir, inputFileName)

# Input variables for the subtype prediction script
paramDir <- paste(directoryPath, "/Script/bioclassifier_R", sep = "") # the location of
# unchanging files such as the function library and main program
inputDir <- paste(directoryPath, "/Data/Output/", dataDir, sep = "") # the location of
# the data matrix, and where output will be located
inputFile <- inputFileName # the input data matrix as a tab delimited text file
short <- "outputFile" # short name that will be used for output files
calibrationParameters <- NA     # the column of the "mediansPerDataset.txt" file to use for
# calibration;
# NA will force centering within the test set & -1 will not do any
# adjustment (when adjustment performed by used)
hasClinical <- FALSE    # may include tumor size as second row, with 'T' as the gene name,
# and encoded as binary (0 for size <= 2cm or 1 for size > 2cm)
# set this variable to FALSE if tumor size is not available
collapseMethod <- "mean" # can be mean or iqr (probe with max iqr is selected)
# typically, mean is preferred for long oligo and
# iqr is preferred for short oligo platforms

# Run the assignment algorithm
source(paste(paramDir, "subtypePrediction_functions.R", sep="/"))
suppressWarnings(source(paste(paramDir, "subtypePrediction_distributed.R", sep="/")))
```

Please copy the file outputFile_pam50scores.txt in “rootDir/Data/Output/bioclassifier_data” which is created from the above script and put in “rootDir/Data”.

```r
#=========================================================================
#=========================================================================
# Detecting drivers of cancer subtypes
#=========================================================================
#=========================================================================
# Clear the environment
rm(list = ls())

# Load necessary libraries if any
library(readxl)
library(miRLAB)
library(miRBaseConverter)
library(GSVA)

#----------------------------------------
# Set environment variables if any
# Please remember to create necessary folders
rootDir <- "C:/Users/phavy022/MyDoc/05CancerDriver" # And put the input files in "rootDir/Data"
controlDir <- "C:/MinGW/bin" # Put here the library from
  # the paper "Controllability of complex networks."

# For Basal drivers
# outDir <- "C:/Users/phavy022/MyDoc/05CancerDriver/Data/Output/CancerSubtype/Basal" # Output folder
# type <- "Basal"

# For Her2 drivers
# outDir <- "C:/Users/phavy022/MyDoc/05CancerDriver/Data/Output/CancerSubtype/Her2" # Output folder
# type <- "Her2"

# For LumA drivers
# outDir <- "C:/Users/phavy022/MyDoc/05CancerDriver/Data/Output/CancerSubtype/LumA" # Output folder
# type <- "LumA"

# For LumB drivers
# outDir <- "C:/Users/phavy022/MyDoc/05CancerDriver/Data/Output/CancerSubtype/LumB" # Output folder
# type <- "LumB"

# For Normal-like drivers
outDir <- "C:/Users/phavy022/MyDoc/05CancerDriver/
  Data/Output/CancerSubtype/NormalLike" # Output folder
type <- "Normal"

#----------------------------------------

# Include the script of functions
source(paste(rootDir, "/Script/ProposedMethod_Functions.R", sep=""))

# Main script - Detecting drivers of cancer subtypes
#=================================================================
# (1) Building the network for a specific condition
#=================================================================
# Load the tumor expression data
load(paste(rootDir, "/Data/BRCA_matchedData_full.RData", sep = ""))

# Get samples
result <- read.table(paste(rootDir, "/Data/outputFile_pam50scores.txt", sep = ""))
result <- result[-1, -1]
# No. of Basal: 158 samples
# No. of Her2: 108 samples
# No. of LumA: 221 samples
# No. of LumB: 165 samples
# No. of Normal-like: 95 samples
# Total: 747 samples
BRCA_matchedData$miRs <- BRCA_matchedData$miRs[
  which(result[, 6] == type),]
BRCA_matchedData$mRNAs <- BRCA_matchedData$mRNAs[
  which(result[, 6] == type),]

# Get PPI network
edges <- read_excel(paste(rootDir, "/Data/PPI.xls",
                          sep = ""), sheet = 1)
interactions <- edges[, c(1, 3)]
colnames(interactions) <- c("cause", "effect")
interactions <- interactions[which(interactions$cause %in% colnames(BRCA_matchedData$mRNAs)),]
interactions <- interactions[which(interactions$effect %in% colnames(BRCA_matchedData$mRNAs)),]
nodes <- unique(union(interactions$cause, interactions$effect))
```

```r
# TFs: Download the list from http://fantom.gsc.riken.jp/5/sstar/Browse_Transcription_Factors_hg19
tfs <- read.csv(paste(rootDir, "/Data/Browse Transcription Factors hg19 - resource_browser.csv",
                      sep = ""))
i <- which(levels(tfs$Symbol) %in% nodes)
tfData <- BRCA_matchedData$mRNAs[, levels(tfs$Symbol)[i]]

# Update cancer data of mRNAs
BRCA_matchedData$mRNAs <- BRCA_matchedData$mRNAs[,
  nodes[which(!(nodes %in% levels(tfs$Symbol)[i]))]]
mRNAsData_Cancer <-  BRCA_matchedData$mRNAs

# Get the cancer data of miRNAs
miRNAsData_Cancer <-  BRCA_matchedData$miRs

# Combine data
nomiR <- ncol(BRCA_matchedData$miRs)
nomR <- ncol(BRCA_matchedData$mRNAs)
noTF <- ncol(tfData)
cancer_data <- cbind(miRNAsData_Cancer, mRNAsData_Cancer, tfData)

# Free the memory
gc()

# Build the network
cancer_network <- buildNetworkWithmiRs(interactions, nomiR, nomR, noTF, cancer_data, rootDir)

# Save the network
write.csv(cancer_network, paste(outDir, "/cancer_network.csv", sep = ""), row.names = FALSE)

# Analyse network
# cancer_network <- read.csv(paste(outDir, "/cancer_network.csv", sep = ""))
analyseNetwork(nomiR, nomR, noTF, cancer_network, cancer_data,
              paste(outDir, "/cancer_network_analysis.txt", sep = ""))

#=================================================================
# (2) Identifying driver profile
#=================================================================
# Load the mutation data
# Read file
load(paste(rootDir, "/Data/mut.RData", sep = ""))
mutData <- mut[, c("gene_name_WU", "Tumor_Sample_Barcode", "trv_type_WU")]
colnames(mutData) <- c("symbol", "Sample", "ct")
# Just get mutations which have effects on proteins
# Refer to the below link for more information
# https://sagebionetworks.jira.com/wiki/spaces/METAGENOMICS/pages/23396441/Curation+of+Mutation+Data
proteinAffectingMut <- mutData[!(mutData$ct %in% c("silent", "rna")),]
proteinAffectingMut$Sample <- substr(proteinAffectingMut$Sample, 1, 12)
# Evaluate the frequency for each mutated gene
proteinAffectingMut$weight <- 0
proteinAffectingMut <- proteinAffectingMut[order(proteinAffectingMut$symbol),]
l <- nrow(proteinAffectingMut)
curGene <- proteinAffectingMut[1,1]
count <- 1
```

```r
for (i in 2:l) {
  if(i == l) {
    if (proteinAffectingMut[i,1] == curGene) {
      count <- count + 1
    } else {
      count <- 1
    }
    proteinAffectingMut[i,4] <- count
  } else {
    if (proteinAffectingMut[i,1] == curGene) {
      count <- count + 1
    } else {
      proteinAffectingMut[i-1,4] <- count
      curGene <- proteinAffectingMut[i,1]
      count <- 1
    }
  }
}
proteinAffectingMut <- proteinAffectingMut[which(proteinAffectingMut[,4] != 0),]
proteinAffectingMut <- proteinAffectingMut[, c(1,4)]

# Analyse controllability of the network
# Read the network with miRNAs
interactions <- read.csv(paste(outDir, "/cancer_network.csv",
                         sep = ""))
# Write the edges of the network for analysing controllability
write.table(interactions, paste(outDir, "/Controllability/edges.dat", sep = ""),
          row.names = FALSE, col.names=FALSE, quote=FALSE)
# Run the controllability analysis
cmd <- paste(controlDir, "/parse.exe ", outDir,
             "/Controllability/edges.dat", sep = "")
system(cmd)
cmd <- paste(controlDir, "/controllability_analysis.exe ", outDir,
             "/Controllability/edges.dat", sep = "")
system(cmd)
# Analyse controllability of the network and output in a file
analyseControllability(paste(outDir, "/Controllability/edges.dat.output", sep = ""),
  paste(outDir, "/analyseControllability.txt", sep = ""))

# Identify critical nodes in the network
# Read the result
nodetype <- read.table(paste(outDir, "/Controllability/edges.dat.nodetype", sep = ""))
colnames(nodetype) <- c("Name", "K", "Kin", "Kout", "TypeI", "TypeII")
# Critical nodes of the network
critical_nodes <- nodetype[which(nodetype$TypeI == 0),]
# Save file
write.csv(critical_nodes, paste(outDir, "/critical_nodes.csv", sep = ""),
          row.names = FALSE)

# Combine mutation data, rank candidate cancer drivers
# Merge with mutaion proteinAffectingMut
candidate_cancer_drivers <- merge(critical_nodes, proteinAffectingMut,
                        all.x = TRUE, by.x = "Name", by.y = "symbol")
```

```r
# Classify miRNAs and TFs/genes
candidate_cancer_drivers$Type <- "coding"
candidate_cancer_drivers[which(candidate_cancer_drivers[, "Name"] %in%
                                 colnames(cancer_data)[1:nomiR]), "Type"] <- "non-coding"
# Candidate coding cancer drivers
coding_candidate_cancer_drivers <- candidate_cancer_drivers[
  which(candidate_cancer_drivers[, "Type"] == "coding"),]
coding_candidate_cancer_drivers <-
  coding_candidate_cancer_drivers[order(coding_candidate_cancer_drivers$weight, decreasing = TRUE),]
coding_candidate_cancer_drivers_mutations <-
  coding_candidate_cancer_drivers[!(is.na(coding_candidate_cancer_drivers$weight)),]
coding_candidate_cancer_drivers_no_mutations <-
  coding_candidate_cancer_drivers[is.na(coding_candidate_cancer_drivers$weight),]
# Candidate non-coding cancer drivers
noncoding_candidate_cancer_drivers <- candidate_cancer_drivers[
  which(candidate_cancer_drivers[, "Type"] == "non-coding"),]
# Write files
write.csv(coding_candidate_cancer_drivers,
          paste(outDir, "/coding_candidate_cancer_drivers.csv", sep = ""), row.names = FALSE)
write.csv(coding_candidate_cancer_drivers_mutations,
          paste(outDir, "/coding_candidate_cancer_drivers_mutations.csv", sep = ""),
          row.names = FALSE)
write.csv(coding_candidate_cancer_drivers_no_mutations,
          paste(outDir, "/coding_candidate_cancer_drivers_no_mutations.csv", sep = ""),
          row.names = FALSE)
write.csv(noncoding_candidate_cancer_drivers,
          paste(outDir, "/noncoding_candidate_cancer_drivers.csv", sep = ""), row.names = FALSE)

#=================================================================
# Evaluation after running the above script for all subtypes
#=================================================================
# Set constants
resultDir <- paste(rootDir, "/Data/Output/CancerSubtype", sep = "") # Result folder
Basal <- "Basal"
Her2 <- "Her2"
LumA <- "LumA"
LumB <- "LumB"
NormalLike <- "NormalLike"

# Load the gold standard, CGC
gold_standard = read.table(
  file = paste(rootDir, "/Data/Census_allFri Sep 28 07_39_37 2018.tsv", sep = ""),
  sep = '\t', header = TRUE)
gold_standard <- gold_standard[, 1]
gold_standard <- as.character(gold_standard)

# Get results
basalList <- readResult(resultDir, Basal)
her2List <- readResult(resultDir, Her2)
lumAList <- readResult(resultDir, LumA)
lumBList <- readResult(resultDir, LumB)
normalLikeList <- readResult(resultDir, NormalLike)
```

```r
# Coding with mutations
print("Basal")
print(getList(basalList$coding_mutations$Name[1:100]))
print("Her2")
print(getList(her2List$coding_mutations$Name[1:100]))
print("LumA")
print(getList(lumAList$coding_mutations$Name[1:100]))
print("LumB")
print(getList(lumBList$coding_mutations$Name[1:100]))
print("Normal-like")
print(getList(normalLikeList$coding_mutations$Name[1:100]))

# Coding without mutations
print("Basal")
print(getList(basalList$coding_no_mutations$Name))
print("Her2")
print(getList(her2List$coding_no_mutations$Name))
print("LumA")
print(getList(lumAList$coding_no_mutations$Name))
print("LumB")
print(getList(lumBList$coding_no_mutations$Name))
print("Normal-like")
print(getList(normalLikeList$coding_no_mutations$Name))

# Non-coding
print("Basal")
print(getList(basalList$noncoding$Name))
print("Her2")
print(getList(her2List$noncoding$Name))
print("LumA")
print(getList(lumAList$noncoding$Name))
print("LumB")
print(getList(lumBList$noncoding$Name))
print("Normal-like")
print(getList(normalLikeList$noncoding$Name))

# Coding with mutations, validated
Basal_Validated <- validateCGC(basalList, gold_standard)
Her2_Validated <- validateCGC(her2List, gold_standard)
LumA_Validated <- validateCGC(lumAList, gold_standard)
LumB_Validated <- validateCGC(lumBList, gold_standard)
NormalLike_Validated <- validateCGC(normalLikeList, gold_standard)

# Get different drivers
diff_coding_mutations <- findDiff(basalList$coding_mutations[1:100,],
                                  her2List$coding_mutations[1:100,],
                                  lumAList$coding_mutations[1:100,],
                                  lumBList$coding_mutations[1:100,],
                                  normalLikeList$coding_mutations[1:100,])
diff_coding_no_mutations <- findDiff(basalList$coding_no_mutations, her2List$coding_no_mutations,
                                  lumAList$coding_no_mutations, lumBList$coding_no_mutations,
                                  normalLikeList$coding_no_mutations)
diff_noncoding <- findDiff(basalList$noncoding, her2List$noncoding,
                                    lumAList$noncoding, lumBList$noncoding,
                                    normalLikeList$noncoding)

# Print different drivers
# Coding with mutations
print("diff_coding_mutations")
print("Basal")
print(getList(diff_coding_mutations$basalDiff))
print("Her2")
print(getList(diff_coding_mutations$her2Diff))
print("LumA")
print(getList(diff_coding_mutations$lumADiff))
print("LumB")
print(getList(diff_coding_mutations$lumBDiff))
print("Normal-like")
print(getList(diff_coding_mutations$normalLikeDiff))
# Coding without mutations
print("diff_coding_no_mutations")
print("Basal")
print(getList(diff_coding_no_mutations$basalDiff))
print("Her2")
print(getList(diff_coding_no_mutations$her2Diff))
print("LumA")
print(getList(diff_coding_no_mutations$lumADiff))
print("LumB")
print(getList(diff_coding_no_mutations$lumBDiff))
print("Normal-like")
print(getList(diff_coding_no_mutations$normalLikeDiff))
# Noncoding
print("diff_noncoding")
print("Basal")
print(getList(diff_noncoding$basalDiff))
print("Her2")
print(getList(diff_noncoding$her2Diff))
print("LumA")
print(getList(diff_noncoding$lumADiff))
print("LumB")
print(getList(diff_noncoding$lumBDiff))
print("Normal-like")
print(getList(diff_noncoding$normalLikeDiff))
```

4. Detecting drivers of epithelial-mesenchymal transition (EMT)

This is the script to detect candidate drivers which are in the network built from the mesenchymal data. The method is applied to the BRCA dataset.

To run the script, please prepare below input files and reset environment variables in the script.

The input files include the followings and are put in the folder "rootDir/Data" (rootDir is an environment variable):

- PPI.xls - Protein protein interaction network
- Browse Transcription Factors hg19 - resource_browser.csv - Transcription factors (TFs)

- hsa.tsv - TransmiR dataset for TF-miRNA interactions
- miRTarBase_v6.1+TarBase_v7.0+miRWalk_v2.0.csv - Datasets for miRNA-mRNA and miRNA-TF interactions
- TargetScan_7.0.csv - Dataset for miRNA-mRNA and miRNA-TF interactions
- BRCA_matchedData_full.RData - Tumour expression data
- Generic_EMT_signature.csv - EMT signatures (From paper: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4287932/)
- EMT_miRNAs.csv - EMT miRNAs (From paper: https://www.sciencedirect.com/science/article/pii/S2405471218302370?via%3Dihub)

```r
#==============================================================================
#==============================================================================
# Detecting drivers of EMT
#==============================================================================
#==============================================================================
# Clear the environment
rm(list = ls())

# Load necessary libraries if any
library(readxl)
library(miRLAB)
library(miRBaseConverter)
library(GSVA)

#-----------------------------------------
# Set environment variables if any
# Please remember to create necessary folders
rootDir <- "C:/Users/phavy022/MyDoc/05CancerDriver" # And put the input files in "rootDir/Data"
controlDir <- "C:/MinGW/bin" # Put here the library from
  # the paper "Controllability of complex networks."

# For Mes drivers
outDir <- "C:/Users/phavy022/MyDoc/05CancerDriver/Data/Output/EMT/Mes" # Output folder for Mes status
type <- "Mes"

# For Epi drivers
# outDir <- "C:/Users/phavy022/MyDoc/05CancerDriver/
#  Data/Output/EMT/Epi" # Output folder for Epi status
# type <- "Epi"
#-----------------------------------------

# Include the script of functions
source(paste(rootDir, "/Script/ProposedMethod_Functions.R", sep=""))

# Main script - Detecting drivers of EMT
#===================================================================
# (1) Building the network for a specific condition
#===================================================================
# Load the tumor expression data
load(paste(rootDir, "/Data/BRCA_matchedData_full.RData", sep = ""))

# Get samples
# Import generic EMT signatures from the reference: Tan TZ, Miow QH, Miki Y, et al.
# Epithelial-mesenchymal transition spectrum quantification and its efficacy in deciphering
# survival and drug responses of cancer patients. EMBO Mol Med. 2014;6(10):1279-93.
Generic_EMT_signature = read.csv(paste(rootDir, "/Data/Generic_EMT_signature.csv", sep = ""),
                                header=FALSE)
Exp <- processData(BRCA_matchedData)
Sample_EMT_Score = scoreEMT(Generic_EMT_signature, Exp)
filename <- paste(outDir, "/", BRCA_matchedData$cancerType, "_Sample_EMT_Score.csv", sep="")
write.csv(Sample_EMT_Score, filename)
BRCA_matchedData$miRs <- BRCA_matchedData$miRs[
  which(Sample_EMT_Score[,3] == type),]
BRCA_matchedData$mRNAs <- BRCA_matchedData$mRNAs[
  which(Sample_EMT_Score[,3] == type),]

# Get PPI network
edges <- read_excel(paste(rootDir, "/Data/PPI.xls",
                          sep = ""), sheet = 1)
interactions <- edges[, c(1, 3)]
colnames(interactions) <- c("cause", "effect")
interactions <- interactions[which(interactions$cause %in% colnames(BRCA_matchedData$mRNAs)),]
interactions <- interactions[which(interactions$effect %in% colnames(BRCA_matchedData$mRNAs)),]
nodes <- unique(union(interactions$cause, interactions$effect))

# TFs: Download the list from http://fantom.gsc.riken.jp/5/sstar/Browse_Transcription_Factors_hg19
tfs <- read.csv(paste(rootDir, "/Data/Browse Transcription Factors hg19 - resource_browser.csv",
                      sep = ""))
i <- which(levels(tfs$Symbol) %in% nodes)
tfData <- BRCA_matchedData$mRNAs[, levels(tfs$Symbol)[i]]

# Update cancer data of mRNAs
BRCA_matchedData$mRNAs <- BRCA_matchedData$mRNAs[,
  nodes[which(!(nodes %in% levels(tfs$Symbol)[i]))]]
mRNAsData_Cancer <-  BRCA_matchedData$mRNAs

# Get the cancer data of miRNAs
miRNAsData_Cancer <-  BRCA_matchedData$miRs

# Combine data
nomiR <- ncol(BRCA_matchedData$miRs)
nomR <- ncol(BRCA_matchedData$mRNAs)
noTF <- ncol(tfData)
cancer_data <- cbind(miRNAsData_Cancer, mRNAsData_Cancer, tfData)

# Free the memory
gc()

# Build the network
cancer_network <- buildNetworkWithmiRs(interactions, nomiR, nomR, noTF, cancer_data, rootDir)

# Save the network
write.csv(cancer_network, paste(outDir, "/cancer_network.csv", sep = ""), row.names = FALSE)

# Analyse network
# cancer_network <- read.csv(paste(outDir, "/cancer_network.csv", sep = ""))
analyseNetwork(nomiR, nomR, noTF, cancer_network, cancer_data,
              paste(outDir, "/cancer_network_analysis.txt", sep = ""))

#=================================================================
# (2) Identifying driver profile
#=================================================================
# Analyse controllability of the network
# Read the network with miRNAs
interactions <- read.csv(paste(outDir, "/cancer_network.csv",
                      sep = ""))
# Write the edges of the network for analysing controllability
write.table(interactions, paste(outDir, "/Controllability/edges.dat", sep = ""),
          row.names = FALSE, col.names=FALSE, quote=FALSE)
# Run the controllability analysis
cmd <- paste(controlDir, "/parse.exe ", outDir,
             "/Controllability/edges.dat", sep = "")
system(cmd)
cmd <- paste(controlDir, "/controllability_analysis.exe ", outDir,
             "/Controllability/edges.dat", sep = "")
system(cmd)
# Analyse controllability of the network and output in a file
analyseControllability(paste(outDir, "/Controllability/edges.dat.output", sep = ""),
  paste(outDir, "/analyseControllability.txt", sep = ""))

# Identify critical nodes in the network
# Read the result
nodetype <- read.table(paste(outDir, "/Controllability/edges.dat.nodetype", sep = ""))
colnames(nodetype) <- c("Name", "K", "Kin", "Kout", "TypeI", "TypeII")
# Critical nodes of the network
critical_nodes <- nodetype[which(nodetype$TypeI == 0),]
# Save file
write.csv(critical_nodes, paste(outDir, "/critical_nodes.csv", sep = ""),
          row.names = FALSE)

# Identify candidate drivers
candidate_drivers <- critical_nodes
# Classify miRNAs and TFs/genes
candidate_drivers$Type <- "coding"
candidate_drivers[which(candidate_drivers[, "Name"] %in%
                                 colnames(cancer_data)[1:nomiR]), "Type"] <- "non-coding"
# Candidate coding drivers
coding_candidate_drivers <- candidate_drivers[
  which(candidate_drivers[, "Type"] == "coding"),]
# Candidate non-coding drivers
noncoding_candidate_drivers <- candidate_drivers[
  which(candidate_drivers[, "Type"] == "non-coding"),]
# Write files
write.csv(coding_candidate_drivers,
          paste(outDir, "/coding_candidate_drivers.csv", sep = ""), row.names = FALSE)
write.csv(noncoding_candidate_drivers,
          paste(outDir, "/noncoding_candidate_drivers.csv", sep = ""), row.names = FALSE)
```

```r
#=================================================================
# Validation after running the above script for EMT
#=================================================================
# Load the gold standard
# Coding
coding_gold_standard = read.table(
  file = paste(rootDir, "/Data/Generic_EMT_signature.csv", sep = ""),
  sep = ',', header = FALSE, as.is = TRUE)
coding_gold_standard_Mes <- coding_gold_standard[which(coding_gold_standard[,2]=="Mes"),]
# coding_gold_standard_Epi <- coding_gold_standard[which(coding_gold_standard[,2]=="Epi"),]
coding_gold_standard_Mes <- coding_gold_standard_Mes[,1]
# coding_gold_standard_Epi <- coding_gold_standard_Epi[,1]
# Noncoding
noncoding_gold_standard = read.table(
  file = paste(rootDir, "/Data/EMT_miRNAs.csv", sep = ""),
  sep = ',', header = FALSE, as.is = TRUE)
noncoding_gold_standard_Mes <-
  noncoding_gold_standard[which(noncoding_gold_standard[,2] %in% c("Pro-mesenchymal", "Both")),]
# noncoding_gold_standard_Epi <-
#   noncoding_gold_standard[which(noncoding_gold_standard[,2] %in% c("Pro-epithelial", "Both")),]
noncoding_gold_standard_Mes <- noncoding_gold_standard_Mes[,1]
# noncoding_gold_standard_Epi <- noncoding_gold_standard_Epi[,1]
noncoding_gold_standard_Mes <- processmiR(noncoding_gold_standard_Mes)
# noncoding_gold_standard_Epi <- processmiR(noncoding_gold_standard_Epi)

# New method - CBNA
outDirMes <- paste(rootDir, "/Data/Output/EMT/Mes", sep = "")
# outDirEpi <- paste(rootDir, "/Data/Output/EMT/Epi", sep = "")
# Coding
coding_candidate_drivers_Mes = read.csv(
  file = paste(outDirMes, "/coding_candidate_drivers.csv", sep = ""))
coding_candidate_drivers_Mes <- coding_candidate_drivers_Mes[
  order(coding_candidate_drivers_Mes$K, decreasing = TRUE),]
coding_candidate_drivers_Mes <- coding_candidate_drivers_Mes[1:100,]
coding_candidate_drivers_Validated_Mes <- intersect(coding_candidate_drivers_Mes[, "Name"],
                                                    coding_gold_standard_Mes)
# coding_candidate_drivers_Epi = read.csv(
#   file = paste(outDirEpi, "/coding_candidate_drivers.csv", sep = ""))
# coding_candidate_drivers_Epi <- coding_candidate_drivers_Epi[
#   order(coding_candidate_drivers_Epi$K, decreasing = TRUE),]
# coding_candidate_drivers_Epi <- coding_candidate_drivers_Epi[1:100,]
# coding_candidate_drivers_Validated_Epi <- intersect(coding_candidate_drivers_Epi[, "Name"],
#                                                     coding_gold_standard_Epi)
# Noncoding
noncoding_candidate_drivers_Mes = read.csv(
  file = paste(outDirMes, "/noncoding_candidate_drivers.csv", sep = ""))
noncoding_candidate_drivers_Validated_Mes <- intersect(noncoding_candidate_drivers_Mes[, "Name"],
                                                       noncoding_gold_standard_Mes)
# noncoding_candidate_drivers_Epi = read.csv(
#   file = paste(outDirEpi, "/noncoding_candidate_drivers.csv", sep = ""))
# noncoding_candidate_drivers_Validated_Epi <- intersect(noncoding_candidate_drivers_Epi[, "Name"],
#                                                        noncoding_gold_standard_Epi)
# Calculate p_value
# Coding
# Mes
n_A = nrow(coding_candidate_drivers_Mes) # number of estimated drivers
n_B = length(coding_gold_standard_Mes) # number of signatures
n_C = 839+5168 # number of genes
n_A_B = length(coding_candidate_drivers_Validated_Mes) # number of drivers validated
p_value <- 1 - phyper(n_A_B, n_B, n_C-n_B, n_A)
print(paste("p_value for coding drivers of Mes: ", p_value, sep = ""))
# Epi
# n_A = nrow(coding_candidate_drivers_Epi) # number of estimated drivers
# n_B = length(coding_gold_standard_Epi) # number of signatures
# n_C = 839+5168 # number of genes
# n_A_B = length(coding_candidate_drivers_Validated_Epi) # number of drivers validated
# p_value <- 1 - phyper(n_A_B, n_B, n_C-n_B, n_A)
# print(paste("p_value for coding drivers of Epi: ", p_value, sep = ""))
# Noncoding
# Mes
n_A = nrow(noncoding_candidate_drivers_Mes) # number of estimated drivers
n_B = length(noncoding_gold_standard_Mes) # number of signatures
n_C = 1719 # number of miRNA
n_A_B = length(noncoding_candidate_drivers_Validated_Mes) # number of drivers validated
p_value <- 1 - phyper(n_A_B, n_B, n_C-n_B, n_A)
print(paste("p_value for noncoding drivers of Mes: ", p_value, sep = ""))
# Epi
# n_A = nrow(noncoding_candidate_drivers_Epi) # number of estimated drivers
# n_B = length(noncoding_gold_standard_Epi) # number of signatures
# n_C = 1719 # number of miRNA
# n_A_B = length(noncoding_candidate_drivers_Validated_Epi) # number of drivers validated
# p_value <- 1 - phyper(n_A_B, n_B, n_C-n_B, n_A)
# print(paste("p_value for noncoding drivers of Epi: ", p_value, sep = ""))
```

5. Detecting personalised cancer drivers

This is the script of the proposed method and its application in detecting personalised cancer drivers. The method is applied to the single cell dataset to discover personalised drivers.

To run the script, please prepare below input files and reset environment variables in the script.

The input files include the followings and are put in the folder "rootDir/Data" (rootDir is an environment variable):

- PPI.xls - Protein protein interaction network
- Browse Transcription Factors hg19 - resource_browser.csv - Transcription factors (TFs)
- GSE118390_epith_exprs.rda - Single cell data
- mut.RData - Mutation data
- Census_allFri Sep 28 07_39_37 2018.tsv - Cancer Gene Census (CGC)

This script uses a library from the paper Liu, Y.-Y., et al. (2011). "Controllability of complex networks." Nature 473: 167. The code of the library can be downloaded from https://scholar.harvard.edu/yyl/code. You need to build the code before running this script.

```r
#==============================================================================
#==============================================================================
# Detecting personalised cancer drivers
#==============================================================================
#==============================================================================
# Clear the environment
rm(list = ls())

# Load necessary libraries if any
library(readxl)
library(miRLAB)
library(miRBaseConverter)
library(ggplot2)
library(varhandle)
library(scales)
library(reshape)
library(plyr)

#----------------------------------------
# Set environment variables if any
# Please remember to create necessary folders
rootDir <- "C:/Users/phavy022/MyDoc/05CancerDriver" # And put the input files in "rootDir/Data"
controlDir <- "C:/MinGW/bin" # Put here the library from
  # the paper "Controllability of complex networks."
outDir <- "C:/Users/phavy022/MyDoc/05CancerDriver/Data/Output/PersonalisedDriver" # Output folder
generalOutDir <- "C:/Users/phavy022/MyDoc/05CancerDriver/Data/Output/CancerDriver/Cancer"
#----------------------------------------

# Include the script of functions
source(paste(rootDir, "/Script/ProposedMethod_Functions.R", sep=""))

# Main script - Detecting personalised cancer drivers
#=================================================================
# (1) Building the network for a specific condition
#=================================================================
# Load the single cell data
load(paste(rootDir, "/Data/GSE118390_epith_exprs.rda", sep = ""))
cellData <- t(epith_exprs)

# Get PPI network
edges <- read_excel(paste(rootDir, "/Data/PPI.xls",
                          sep = ""), sheet = 1)
interactions <- edges[, c(1, 3)]
colnames(interactions) <- c("cause", "effect")
interactions <- interactions[which(interactions$cause %in% colnames(cellData)),]
interactions <- interactions[which(interactions$effect %in% colnames(cellData)),]
nodes <- unique(union(interactions$cause, interactions$effect))

# TFs: Download the list from http://fantom.gsc.riken.jp/5/sstar/Browse_Transcription_Factors_hg19
tfs <- read.csv(paste(rootDir, "/Data/Browse Transcription Factors hg19 - resource_browser.csv",
                      sep = ""))
i <- which(levels(tfs$Symbol) %in% nodes)
tfData <- cellData[, levels(tfs$Symbol)[i]]
```

```r
# Update cancer data of mRNAs
cellData <- cellData[,
  nodes[which(!(nodes %in% levels(tfs$Symbol)[i]))]]
mRNAsData_Cancer <-  cellData

# Combine data
nomR <- ncol(cellData)
noTF <- ncol(tfData)
cancer_data <- cbind(mRNAsData_Cancer, tfData)

# Free the memory
gc()

# Build the network
cancer_network <- buildNetworkForPersonalised(interactions, nomR, noTF, cancer_data, rootDir)

# Save the network
write.csv(cancer_network, paste(outDir, "/cancer_network.csv", sep = ""), row.names = FALSE)

# Analyse network
# cancer_network <- read.csv(paste(outDir, "/cancer_network.csv", sep = ""))
analyseNetworkForPersonalised(nomR, noTF, cancer_network, cancer_data,
              paste(outDir, "/cancer_network_analysis.txt", sep = ""))

#=================================================================
# (2) Identifying driver profile
#=================================================================
# Load the mutation data
# Read file
load(paste(rootDir, "/Data/mut.RData", sep = ""))
mutData <- mut[, c("gene_name_WU", "Tumor_Sample_Barcode", "trv_type_WU")]
colnames(mutData) <- c("symbol", "Sample", "ct")
# Just get mutations which have effects on proteins
# Refer to the below link for more information
# https://sagebionetworks.jira.com/wiki/spaces/METAGENOMICS/pages/23396441/Curation+of+Mutation+Data
proteinAffectingMut <- mutData[!(mutData$ct %in% c("silent", "rna")),]
proteinAffectingMut$Sample <- substr(proteinAffectingMut$Sample, 1, 12)
# Evaluate the frequency for each mutated gene
proteinAffectingMut$weight <- 0
proteinAffectingMut <- proteinAffectingMut[order(proteinAffectingMut$symbol),]
l <- nrow(proteinAffectingMut)
curGene <- proteinAffectingMut[1,1]
count <- 1
for (i in 2:l) {
  if(i == l) {
    if (proteinAffectingMut[i,1] == curGene) {
      count <- count + 1
    } else {
      count <- 1
    }
    proteinAffectingMut[i,4] <- count
  } else {
    if (proteinAffectingMut[i,1] == curGene) {
      count <- count + 1
    } else {
      proteinAffectingMut[i-1,4] <- count
      curGene <- proteinAffectingMut[i,1]
      count <- 1
    }
  }
}
proteinAffectingMut <- proteinAffectingMut[which(proteinAffectingMut[,4] != 0),]
proteinAffectingMut <- proteinAffectingMut[, c(1,4)]

# Analyse controllability of the network
# Read the network
interactions <- read.csv(paste(outDir, "/cancer_network.csv",
                      sep = ""))
# Write the edges of the network for analysing controllability
write.table(interactions, paste(outDir, "/Controllability/edges.dat", sep = ""),
          row.names = FALSE, col.names=FALSE, quote=FALSE)
# Run the controllability analysis
cmd <- paste(controlDir, "/parse.exe ", outDir,
             "/Controllability/edges.dat", sep = "")
system(cmd)
cmd <- paste(controlDir, "/controllability_analysis.exe ", outDir,
             "/Controllability/edges.dat", sep = "")
system(cmd)
# Analyse controllability of the network and output in a file
analyseControllability(paste(outDir, "/Controllability/edges.dat.output", sep = ""),
  paste(outDir, "/analyseControllability.txt", sep = ""))

# Identify critical nodes in the network
# Read the result
nodetype <- read.table(paste(outDir, "/Controllability/edges.dat.nodetype", sep = ""))
colnames(nodetype) <- c("Name", "K", "Kin", "Kout", "TypeI", "TypeII")
# Critical nodes of the network
critical_nodes <- nodetype[which(nodetype$TypeI == 0),]
# Save file
write.csv(critical_nodes, paste(outDir, "/critical_nodes.csv", sep = ""),
          row.names = FALSE)

# Combine mutation data, rank candidate cancer drivers
# Merge with mutaion proteinAffectingMut
candidate_cancer_drivers <- merge(critical_nodes, proteinAffectingMut,
                        all.x = TRUE, by.x = "Name", by.y = "symbol")
# Set type
candidate_cancer_drivers$Type <- "coding"
# Candidate coding cancer drivers
coding_candidate_cancer_drivers <- candidate_cancer_drivers
coding_candidate_cancer_drivers <-
  coding_candidate_cancer_drivers[order(coding_candidate_cancer_drivers$weight, decreasing = TRUE),]
coding_candidate_cancer_drivers_mutations <-
  coding_candidate_cancer_drivers[!(is.na(coding_candidate_cancer_drivers$weight)),]
coding_candidate_cancer_drivers_no_mutations <-
  coding_candidate_cancer_drivers[is.na(coding_candidate_cancer_drivers$weight),]
```

```r
# Write files
write.csv(coding_candidate_cancer_drivers,
          paste(outDir, "/coding_candidate_cancer_drivers.csv", sep = ""), row.names = FALSE)
write.csv(coding_candidate_cancer_drivers_mutations,
          paste(outDir, "/coding_candidate_cancer_drivers_mutations.csv", sep = ""),
          row.names = FALSE)
write.csv(coding_candidate_cancer_drivers_no_mutations,
          paste(outDir, "/coding_candidate_cancer_drivers_no_mutations.csv", sep = ""),
          row.names = FALSE)

#=================================================================
# Validation
#=================================================================
# Load the gold standard, CGC
gold_standard = read.table(
  file = paste(rootDir, "/Data/Census_allFri Sep 28 07_39_37 2018.tsv", sep = ""),
  sep = '\t', header = TRUE)
gold_standard <- gold_standard[, 1]
gold_standard <- as.character(gold_standard)

# Personalised
personalised_drivers_mutations = read.csv(
  file = paste(outDir, "/coding_candidate_cancer_drivers_mutations.csv", sep = ""))
personalised_drivers_no_mutations = read.csv(
  file = paste(outDir, "/coding_candidate_cancer_drivers_no_mutations.csv", sep = ""))

# General
drivers_mutations = read.csv(
  file = paste(generalOutDir, "/coding_candidate_cancer_drivers_mutations.csv", sep = ""))
drivers_no_mutations = read.csv(
  file = paste(generalOutDir, "/coding_candidate_cancer_drivers_no_mutations.csv", sep = ""))

# 1. Validate with CGC
personalised_Top100 <- personalised_drivers_mutations[1:100,]
personalised_Top100_Validated <- intersect(personalised_Top100[, "Name"], gold_standard)

# 2. Compare to general
general_Top100 <- drivers_mutations[1:100,]
common_drivers_mutations <- intersect(personalised_Top100[, "Name"],general_Top100[, "Name"])
common_drivers_no_mutations <- intersect(personalised_drivers_no_mutations[, "Name"],
                                         drivers_no_mutations[, "Name"])
```