PSB 2017: Exploring the Reproducibility of Probabilistic Causal Molecular Network Models

R Code used in analyses and generating figures and tables

Ariella Cohain & Aparna Divaraniya

September 23, 2016

Contents

Create Edges_all object: **1**

Analysis Set-up: . . . 4

Load Functions . . . 4

summarySE Function . . . 4

get_ABoverA_directed (Figure 2A) . . . 5

Load in Data . . . 5

Analyses for Figures . . . 6

Figure 2A: Precision/Recall Plots for edges . . . 6

Figure 2B: Evaluation of Path Length . . . 6

Figure 2C: Clique Associates for Various Correlation Cutoffs for Edges . . . 10

Figure 3: Simulation Data Precision / Recall . . . 15

Figure 4: Reproducibility of Key Driver Nodes (KDs) . . . 18

Figure 5: Hub Nodes Reproducibility Rate . . . 22

Table 2: Compare 5 Replicates of 100% Networks . . . 26

Table 3: Jaccard Index . . . 27

Table 4: Edge Counts . . . 29

All files required to reproduce the results presented in this paper are located in the following directory:

```
gitPath <- c("https://raw.githubusercontent.com/divara01/PSB2017_ReproducibilityOfBNs/")
```

Create Edges_all object:

This code breaks up the results from the RIMBAnet algorithm to save the resulting edges into a dataframe. Singletons are removed. Each edge is associated with a resulting posterior probability which is output from the algorithm as well. The resulting edges_all file is saved in the current working directory. This is where all downstream output will be saved.

```
options(stringsAsFactors = F)
rm(list = ls())

#### Load all files into R####
gitPath <- c("https://raw.githubusercontent.com/divara01/PSB2017_ReproducibilityOfBNs/")
dataset <- c("GTEx", "STARNET", "Simulation")
subSets <- c(100, 90, 80, 50, 10)
replicates <- c(1:5)

FullLabels <- as.vector(sapply(subSets, function(x) sapply(replicates,
    function(y) paste(x, y, sep = "_"))))

for (datasetIndex in 1:length(dataset)) {
    if (datasetIndex < 3) {
        labels <- FullLabels[-grep("10_", FullLabels)]
    } else {
        labels <- FullLabels
    }

    for (index in 1:length(labels)) {
        filename <- paste(gitPath, "data/", dataset[datasetIndex],
            "_", labels[index], "_EdgeFile", sep = "")
        edgeFile_i <- read.csv(filename, quote = "\"", sep = "\t",
            comment.char = "", stringsAsFactors = FALSE)
        edgeFile_i <- as.data.frame.matrix(t(apply(edgeFile_i,
            1, function(x) {
                strsplit(x, "->", fixed = T)[[1]]
            })))
        colnames(edgeFile_i) <- c("FromNode", "ToNode")
        PostProbFile_i <- read.csv(paste(gitPath, "data/", dataset[datasetIndex],
            "_", labels[index], "_labeledEdges", sep = ""), quote = "\"",
            comment.char = "", stringsAsFactors = FALSE, header = F)
        PostProbFile_i$Edge <- unlist(lapply(strsplit(PostProbFile_i$V1,
            " "), function(x) x[1]))
        PostProbFile_i$PostProb <- as.numeric(unlist(lapply(strsplit(PostProbFile_i$V1,
            " []label="), function(x) lapply(strsplit(x[2],
            "[]];"), function(y) y[1]))))

        assign(paste(dataset[datasetIndex], labels[index], "EdgeFile",
            sep = "_"), edgeFile_i)
        assign(paste(dataset[datasetIndex], "PostProbFile", labels[index],
            sep = "_"), PostProbFile_i)
    }
}

rm(index, edgeFile_i, PostProbFile_i)

for (datasetIndex in 1:length(dataset)) {
    message(paste(dataset[datasetIndex], "running...", sep = " "))
    edges_all <- data.frame(matrix(NA, nrow = 1, ncol = 5))
    colnames(edges_all) <- c("node1", "node2", "postProb", "repRun",
        "data_type")
```

```
if (datasetIndex < 3) {
    labels <- FullLabels[-grep("10_", FullLabels)]
} else {
    labels <- FullLabels
}

for (index in 1:length(labels)) {
    edgeFile_i <- get(paste(dataset[datasetIndex], labels[index],
        "EdgeFile", sep = "_"))
    edge_i <- paste(edgeFile_i$FromNode, edgeFile_i$ToNode,
        sep = "->")
    PostProbFile_i <- get(paste(dataset[datasetIndex], "PostProbFile",
        labels[index], sep = "_"))
    postProb <- as.character(substr(PostProbFile_i$PostProb[match(edge_i,
        PostProbFile_i$Edge)], 1, 3))
    node1 <- as.vector(unlist(lapply(strsplit(edge_i, "->"),
        function(x) x[1])))
    node2 <- as.vector(unlist(lapply(strsplit(edge_i, "->"),
        function(x) x[2])))
    repRun <- rep(as.vector(strsplit(labels[index], "_")[[1]][2]),
        length(edge_i))
    data_type <- rep(as.vector(strsplit(labels[index], "_")[[1]][1]),
        length(edge_i))
    vect <- cbind(node1, node2, postProb, repRun, data_type)
    edges_all <- rbind.data.frame(edges_all, vect)
}
edges_all <- edges_all[-1, ]

# Add the additional rows for all posterior probabilities
# below the actual value (so for postProb = 0.5, add row for
# 0.4, 0.3, 0.2, and 0.1)
postProbOptions <- seq(0.1, 1, 0.1)

for (index in 1:nrow(edges_all)) {
    postProb_i <- edges_all$postProb[index]
    pos <- which(postProbOptions == postProb_i)

    tempEdgeFile <- c()
    if (pos != 1) {
        for (i in 1:(pos - 1)) {
            tempEdgeFile <- rbind(tempEdgeFile, edges_all[index,
              ])
            tempEdgeFile$postProb[nrow(tempEdgeFile)] <- postProbOptions[i]
        }
        edges_all <- rbind.data.frame(edges_all, tempEdgeFile)
    }
}
edges_all <- edges_all[order(as.numeric(edges_all$data_type),
    decreasing = TRUE), ]
edgesAll_filename <- paste(dataset[datasetIndex], "_edges_all_RIMBAnet.RData",
    sep = "")
save(edges_all, file = edgesAll_filename)
message(paste(dataset[datasetIndex], "complete!", sep = " "))
```

```
}
```

Analysis Set-up:

```
options(stringsAsFactors = F)
rm(list = ls())
library(ggplot2)
library(reshape)
library(RCurl)
```

Load Functions

summarySE Function

This function provides the count, mean, standard deviation, standard error of the mean, and confidence interval (default 95%). The inputs for the functions are the following:

- data: a data frame.
- measurevar: the name of a column that contains the variable to be summariezed
- groupvars: a vector containing names of columns that contain grouping variables
- na.rm: a boolean that indicates whether to ignore NA's
- conf.interval: the percent range of the confidence interval (default is 95%)

```
summarySE <- function(data = NULL, measurevar, groupvars = NULL,
    na.rm = FALSE, conf.interval = 0.95, .drop = TRUE) {
    library(plyr)

    # New version of length which can handle NA's: if na.rm==T,
    # don't count them
    length2 <- function(x, na.rm = FALSE) {
        if (na.rm)
            sum(!is.na(x)) else length(x)
    }

    # This does the summary. For each group's data frame, return
    # a vector with N, mean, and sd
    datac <- ddply(data, groupvars, .drop = .drop, .fun = function(xx,
        col) {
        c(N = length2(xx[[col]], na.rm = na.rm), mean = mean(xx[[col]],
            na.rm = na.rm), sd = sd(xx[[col]], na.rm = na.rm))
    }, measurevar)

    # Rename the 'mean' column
    datac <- rename(datac, c(mean = measurevar))

    datac$se <- datac$sd/sqrt(datac$N)  # Calculate standard error of the mean

    # Confidence interval multiplier for standard error Calculate
    # t-statistic for confidence interval: e.g., if conf.interval
    # is .95, use .975 (above/below), and use df=N-1
```

```
    ciMult <- qt(conf.interval/2 + 0.5, datac$N - 1)
    datac$ci <- datac$se * ciMult

    return(datac)
}
```

get_ABoverA_directed (Figure 2A)

Given two networks A and B, obtain the fraction of edges in both A and B over the number of edges in A assuming fixed probability for edges in A and any edge in B.

```
get_ABoverA_directed <- function(edges_all, A_name, B_name, prob,
    sameProb = F) {
    if (A_name == B_name) {
        if (sameProb) {
            aANDb = length(intersect(edges_all$edge_name[edges_all$postProb ==
                prob & edges_all$type_name == A_name], edges_all$edge_name[edges_all$postProb ==
                prob & edges_all$type_name != B_name]))
        } else {
            aANDb = length(intersect(edges_all$edge_name[edges_all$postProb ==
                prob & edges_all$type_name == A_name], edges_all$edge_name[edges_all$type_name !=
                B_name]))
        }
    } else {
        if (sameProb) {
            aANDb = length(intersect(edges_all$edge_name[edges_all$postProb ==
                prob & edges_all$type_name == A_name], edges_all$edge_name[edges_all$postProb ==
                prob & edges_all$type_name == B_name]))
        } else {
            aANDb = length(intersect(edges_all$edge_name[edges_all$postProb ==
                prob & edges_all$type_name == A_name], edges_all$edge_name[edges_all$type_name ==
                B_name]))
        }
    }
    a = sum(edges_all$postProb == prob & edges_all$type_name ==
        A_name)
    return(aANDb/a)
}
```

Load in Data

```
# Load in the data:
load("STARNET_edges_all_RIMBAnet.RData")
edges_all_starnet = edges_all
rm(edges_all)
load("GTEx_edges_all_RIMBAnet.RData")
edges_all_gtex = edges_all
rm(edges_all)

## adding edge name:
edges_all_starnet$edge_name = paste(edges_all_starnet$node1,
```

```
    edges_all_starnet$node2, sep = "_")
edges_all_gtex$edge_name = paste(edges_all_gtex$node1, edges_all_gtex$node2,
    sep = "_")
## adding type_name:
edges_all_starnet$type_name = paste(edges_all_starnet$data_type,
    edges_all_starnet$repRun, sep = "_")
edges_all_gtex$type_name = paste(edges_all_gtex$data_type, edges_all_gtex$repRun,
    sep = "_")
```

Analyses for Figures

Figure 2A: Precision/Recall Plots for edges

Network A: Subsampling Network
Network B: Complete Network (100% Replicate 1)
Precision = A ∩ B / A
Recall = A ∩ B / B

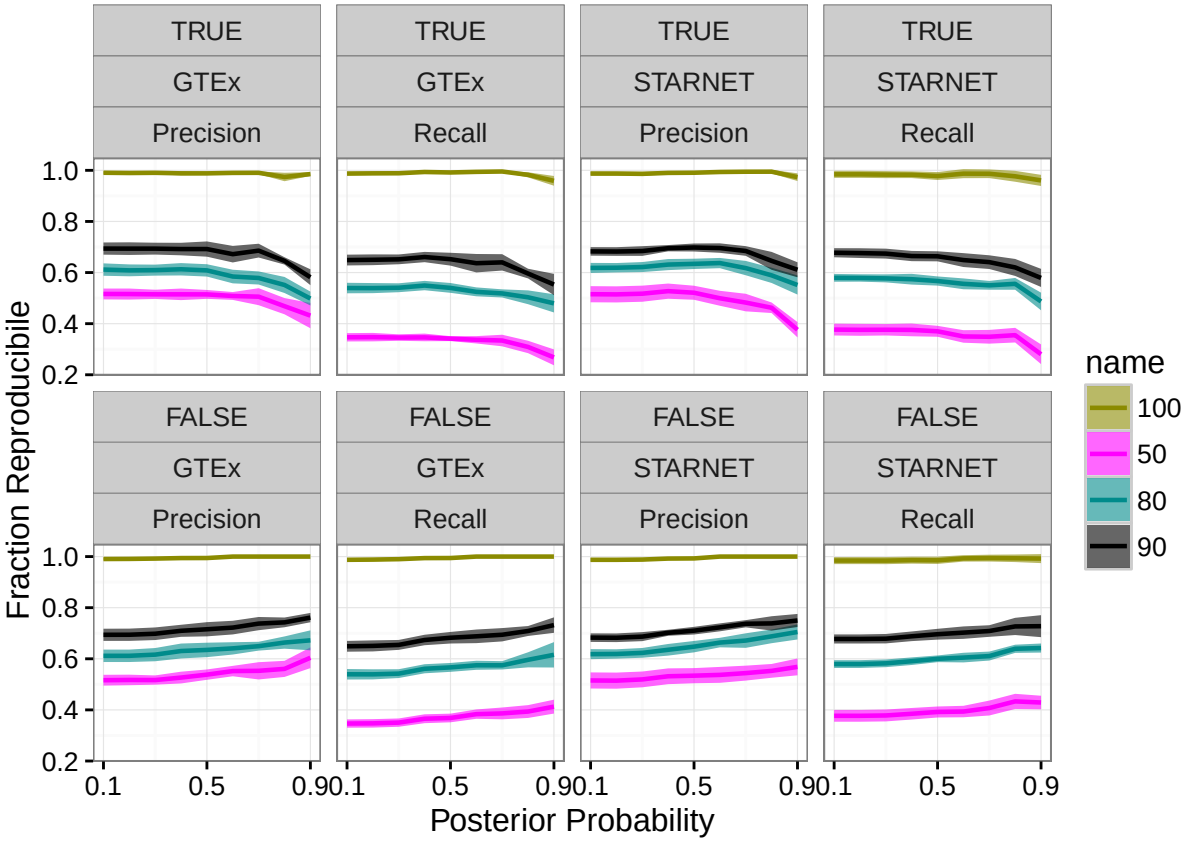

Figure 2B: Evaluation of Path Length

This code will prepare the data. It is highly advised that the code is run and resulting object is saved as it takes some time to run.

```
gtex_fn = "GTEx_edges_all_RIMBAnet.RData"
```

```
load(gtex_fn)
head(edges_all)

real_network_all = edges_all[edges_all$repRun == 1 & edges_all$data_type ==
    100, ]
dim(real_network_all)

threads = 18
library(foreach)

library(parallel)
library(doMC)
options(cores = detectCores())
registerDoMC(threads)

path <- getwd()
all_out = c()
tests = cbind(rep(1:9, 15), c(rep(1, 5 * 9), rep(2, 5 * 9), rep(3,
    5 * 9)), rep(1:5))
colnames(tests) = c("prob", "subsampling_idx", "repRun")

edgesInPath <- function(edgeList1, g2, path_length, directed = TRUE) {

    if (!directed) {
        g2 = as.undirected(g2)
    }

    InPath = apply(edgeList1, 1, function(x) {
        n1 = x[1]
        n2 = x[2]

        if (sum(is.element(c(n1, n2), V(g2)$name)) == 2) {
            p = get.shortest.paths(g2, n1, n2)
            l = min(unlist(lapply(p$vpath, length)))

            if (l <= (path_length + 1) & l != 0) {
                out = 1
            } else {
                out = 0
            }
        } else {
            out = 0
        }
        out
    })

    return(InPath)
}

compare_twoBN <- function(rand1, rand2, maxK = 10, text = TRUE) {
    g1 <- graph.edgelist(as.matrix(rand1))
    g2 <- graph.edgelist(as.matrix(rand2))
```

```
    rownames(rand1) = paste(rand1[, 1], rand1[, 2], sep = "_")
    rownames(rand2) = paste(rand2[, 1], rand2[, 2], sep = "_")
    out = c()

    for (i in 1:maxK) {
        if (text) {
            cat("on run path of", i, "\n")
        }
        e1_in_g2 = edgesInPath(as.matrix(rand1), g2, i)
        e2_in_g1 = edgesInPath(as.matrix(rand2), g1, i)

        fracE1_in_G2 = sum(e1_in_g2/length(e1_in_g2))
        fracE2_in_G1 = sum(e2_in_g1/length(e2_in_g1))

        e1_in_g2_undir = edgesInPath(as.matrix(rand1), g2, i,
            FALSE)
        e2_in_g1_undir = edgesInPath(as.matrix(rand2), g1, i,
            FALSE)

        fracE1_in_G2_undir = sum(e1_in_g2_undir/length(e1_in_g2_undir))
        fracE2_in_G1_undir = sum(e2_in_g1_undir/length(e2_in_g1_undir))

        out = rbind(out, cbind(fracE1_in_G2, fracE2_in_G1, fracE1_in_G2_undir,
            fracE2_in_G1_undir))
    }

    rownames(out) = paste("path_", 1:maxK, sep = "")
    return(out)
}

run_Network_comparisions <- function(edges_all, real_network,
    probs = seq(0.1, 0.9, 0.1)) {

    all = c()
    for (prob in probs) {
        net = edges_all[edges_all$postProb == prob, c("node1",
            "node2")]
        comp = compare_twoBN(net, real_network, maxK = 10, text = F)
        comp = as.data.frame.matrix(comp)
        comp$post_prob_net1 = prob
        comp$path = 1:nrow(comp)
        all = rbind(all, comp)
    }
    cat("\n")
    return(all)
}

paths = c(50, 80, 90)

## In parallel calculating all the different comparisions:
all_out <- foreach(i = 1:nrow(tests), combine = rbind) %dopar%
    {
        print(paste("real network", i, "start time:", Sys.time()))
```

```
        out = run_Network_comparisions(edges_all[edges_all$data_type ==
            paths[tests[i, 2]] & edges_all$repRun == tests[i,
            3], ], real_network_all[real_network_all$postProb ==
            tests[i, 1]/10, c("node1", "node2")])

        out$Name = paste(paths[tests[i, 2]], tests[i, 3], sep = "_")
        out$post_prob_net2 = tests[i, 1]/10
        print(paste("real netowrk", i, "end time:", Sys.time()))

        out
    }

results = do.call(rbind, all_out)
save(results, file = "Fig2B_GTEx_updated.RData")
```

We are not displaying the figure here, however, below is the code to generate Figure 2B:

```
load("Fig2B_GTEx_updated.RData")

results$name = sapply(results$Name, function(x) {
    paste(strsplit(x, "_")[[1]][1], collapse = "_")
})
results$repRun = sapply(results$Name, function(x) {
    paste(strsplit(x, "_")[[1]][2], collapse = "_")
})
results = results[, colnames(results) != "Name"]
results = results[, colnames(results) != "fracE1_in_G2_undir"]
results = results[, colnames(results) != "fracE2_in_G1_undir"]

melted_data = melt(results, id = c("name", "path", "post_prob_net2",
    "post_prob_net1", "repRun"))
melted_data$path = as.factor(melted_data$path)

melted_data$post_prob_net1_name = paste("SubSampling Posterior: ",
    melted_data$post_prob_net1, sep = "")
melted_data$post_prob_net2_name = paste("Real Posterior: ", melted_data$post_prob_net2,
    sep = "")
ggplotDF = melted_data[melted_data$variable == "fracE1_in_G2" &
    is.element(melted_data$post_prob_net1_name, c("SubSampling Posterior: 0.1",
        "SubSampling Posterior: 0.5", "SubSampling Posterior: 0.9")),
    ]

ggplotDF2 = ggplotDF[is.element(ggplotDF$post_prob_net2, c("0.1",
    "0.5", "0.9")), ]

ggplotDF2_summary <- summarySE(ggplotDF2, "value", groupvars = c("name",
    "path", "post_prob_net1", "post_prob_net2"))
ggplotDF2_summary$path = as.numeric(as.character(ggplotDF2_summary$path))

ggplotDF2_summary$post_prob_net1_name = as.character(ggplotDF2_summary$post_prob_net1)
ggplotDF2_summary$post_prob_net2_name = as.character(ggplotDF2_summary$post_prob_net2)

fig_2b_gtex <- ggplot(ggplotDF2_summary, aes(path)) + geom_line(aes(y = value,
```

```
    colour = post_prob_net2_name)) + geom_ribbon(aes(ymin = value -
    sd, ymax = value + sd, fill = post_prob_net2_name), alpha = 0.8) +
    facet_wrap(~name + post_prob_net1_name, ncol = 3) + xlab("Path Length") +
    scale_color_manual(values = c("orange", "green", "grey50")) +
    scale_fill_manual(values = c("orange", "green", "grey50")) +
    ggtitle("Fraction of SubSampling Edges in Real BN at Different SubSamping Thresholds") +
    theme(axis.text.x = element_text(size = 12), title = element_text(size = 16)) +
    theme_bw() + scale_x_continuous(breaks = 1:10)
show(fig_2b_gtex)
```

Figure 2C: Clique Associates for Various Correlation Cutoffs for Edges

Create Edge files based on correlation thresholds

```
dataFlag <- c("GTEx", "STARNET")

subSets <- c(100, 90, 80, 50)
replicates <- c(1:5)

labels <- as.vector(sapply(subSets, function(x) sapply(replicates,
    function(y) paste(x, y, sep = "_"))))

dataFN <- as.vector(sapply(dataFlag, function(x) sapply(labels,
    function(y) paste(x, y, sep = "_"))))


for (labelsTagIndex in 1:length(dataFN)) {
    labelsTag <- dataFN[labelsTagIndex]
    for (cutoff in c(0.1, 0.05, 0.01)) {
        message(paste(dataSet, "-", cutoff, "-", labelsTag, "running..."))
        data = read.table(paste(gitPath, "RIMBAnetInput/", dataSet,
            "_", labelsTag, "_BC.txt", sep = ""), quote = "\"",
            comment.char = "", stringsAsFactors = FALSE, row.names = 1)
        cor_data = cor(t(data))^2

        rsqrd_cutoff = quantile(unlist(cor_data[upper.tri(cor_data)]),
            1 - cutoff)

        keep_edges = which(cor_data >= rsqrd_cutoff, arr.ind = T)

        edges = t(apply(keep_edges, 1, function(X) {
            t = c(rownames(cor_data)[X[1]], rownames(cor_data)[X[2]])
            t[order(t)]
        }))

        edges = unique(edges)
        edges = edges[edges[, 1] != edges[, 2], ]
        write.table(edges, sep = "\t", quote = F, col.names = F,
            row.names = F, file = paste(dataSet, labelsTag, "_",
                cutoff, "_topCorEdges.txt", sep = ""))
    }
}
```

We feed the resulting edge files into the COS program to determine which edges have both nodes in a single clique. Please see the README file that is provided in the GITHUB repository. The code below parses the results from COS and saves the cliques into a list for each dataset, GTEx and STARNET.

```
subSets <- c(100, 90, 80, 50)
replicates <- c(1:5)
postProbs <- c(seq(0.1, 0.9, 0.1))
thresholds <- c(0.01, 0.05, 0.1)
datasets <- c("GTEx", "STARNET")

labels <- as.vector(sapply(subSets, function(x) sapply(replicates,
    function(y) sapply(thresholds, function(z) paste(x, y, z,
        sep = "_")))))

for (datasetIndex in 1:length(datasets)) {
    message(paste(datasets[datasetIndex], "running..."))
    MappedCliques_i <- vector("list", length(labels))

    for (index in 1:length(labels)) {
        cliqueMap_i <- read.table(paste(datasets[datasetIndex],
            "_", labels[index], "_topCorEdges.txt.map", sep = ""),
            quote = "\"", comment.char = "")

        filename <- paste(datasets[datasetIndex], labels[index],
            "MappedCliques", sep = "_")

        if (file.exists(paste(datasets[datasetIndex], "_", labels[index],
            "_topCorEdges.txt.mcliques_3_communities.txt", sep = ""))) {
            cliqueCommunity_i <- read.table(paste(datasets[datasetIndex],
                "_", labels[index], "_topCorEdges.txt.mcliques_3_communities.txt",
                sep = ""), quote = "\"", comment.char = "", fill = TRUE)

            # First column is not delimited properly, so need to split
            # the label out:
            cliqueCommunity_i$V1 <- as.integer(as.vector(unlist(lapply(strsplit(cliqueCommunity_i$V1,
                ":"), function(x) x[2]))))

            mappedCliques <- apply(cliqueCommunity_i, 2, function(x) cliqueMap_i$V1[match(x,
                cliqueMap_i$V2)])
            # Store the mapped cliques locally in R workspace

            assign(filename, mappedCliques)

            # Store the mapped cliques into a list
            MappedCliques_i[[index]] <- mappedCliques
        }
        names(MappedCliques_i)[[index]] <- filename
    }

    # Save RData file
    save(MappedCliques_i, file = paste(datasets[datasetIndex],
        "_MappedCliquesList.RData", sep = ""))
    message(paste(datasets[datasetIndex], "Complete!"))
}
```

Once cliques are called, add a flag for each edge as to whether both nodes are found in the same clique. The results from this code should be saved and then loaded back in.

```
options(stringsAsFactors = F)
rm(list = ls())
library(ggplot2)
library(reshape)

gitPath <- c("https://raw.githubusercontent.com/divara01/PSB2017_ReproducibilityOfBNs/")

inclique <- function(Clique_list, name, node1, node2) {
    cliques <- Clique_list[[name]]
    if (class(cliques) == "character") {
        cliques = matrix(cliques, nrow = 1)
    }
    if (length(cliques) == 0) {
        cat("this run has NO cliques,", name, "\n")
        return(FALSE)
    }
    pos_n1 <- which(!is.na(apply(cliques, 1, function(x) match(node1,
        x))))
    pos_n2 <- which(!is.na(apply(cliques, 1, function(x) match(node2,
        x))))
    cliquesContainingBothNodes <- intersect(pos_n1, pos_n2)

    if (length(cliquesContainingBothNodes) > 0) {
        return(TRUE)
    } else {
        return(FALSE)
    }
}

## loading Clique List
load(paste(gitPath, "COSresults/STARNET_MappedCliquesList.RData",
    sep = ""))
## loading edges_all
load("STARNET_edges_all_RIMBAnet.RData")

CompleteCliqueFlag <- data.frame(matrix(NA, nrow = nrow(edges_all),
    ncol = 3))
colnames(CompleteCliqueFlag) <- c("0.01", "0.05", "0.1")

for (cutOffIndex in 1:ncol(CompleteCliqueFlag)) {
    print(paste(colnames(CompleteCliqueFlag)[cutOffIndex], " Running...",
        sep = ""))

    CompleteCliqueFlag[, cutOffIndex] <- sapply(1:nrow(edges_all),
        function(x) {
            MappedName = paste("STARNET", edges_all$data_type[x],
                edges_all$repRun[x], colnames(CompleteCliqueFlag)[cutOffIndex],
                "MappedCliques", sep = "_")
            inclique(MappedCliques_i, MappedName, edges_all$node1[x],
                edges_all$node2[x])
        })
```

```
    print(paste(colnames(CompleteCliqueFlag)[cutOffIndex], " Complete!",
        sep = ""))
}

colnames(CompleteCliqueFlag) <- paste("inClique", colnames(CompleteCliqueFlag),
    sep = "_")
edges_all <- cbind.data.frame(edges_all, CompleteCliqueFlag)

edges_all$inClique_0.1 <- as.integer(edges_all$inClique_0.1 +
    0)
edges_all$inClique_0.05 <- as.integer(edges_all$inClique_0.05 +
    0)
edges_all$inClique_0.01 <- as.integer(edges_all$inClique_0.01 +
    0)

save(edges_all, file = "STARNET_edges_all_RIMBAnet_cliqueAdded.RData")

## GTEx:
load(paste(gitPath, "COSresults/GTEx_MappedCliquesList.RData",
    sep = ""))
load("GTEx_edges_all_RIMBAnet.RData")

CompleteCliqueFlag <- data.frame(matrix(NA, nrow = nrow(edges_all),
    ncol = 3))
colnames(CompleteCliqueFlag) <- c("0.01", "0.05", "0.1")

for (cutOffIndex in 1:ncol(CompleteCliqueFlag)) {
    print(paste(colnames(CompleteCliqueFlag)[cutOffIndex], " Running...",
        sep = ""))
    CompleteCliqueFlag[, cutOffIndex] <- sapply(1:nrow(edges_all),
        function(x) {
            MappedName = paste("GTEx", edges_all$data_type[x],
                edges_all$repRun[x], colnames(CompleteCliqueFlag)[cutOffIndex],
                "MappedCliques", sep = "_")
            inclique(MappedCliques_i, MappedName, edges_all$node1[x],
                edges_all$node2[x])
        })
    print(paste(colnames(CompleteCliqueFlag)[cutOffIndex], " Complete!",
        sep = ""))
}

colnames(CompleteCliqueFlag) <- paste("inClique", colnames(CompleteCliqueFlag),
    sep = "_")
edges_all <- cbind.data.frame(edges_all, CompleteCliqueFlag)

edges_all$inClique_0.1 <- as.integer(edges_all$inClique_0.1 +
    0)
edges_all$inClique_0.05 <- as.integer(edges_all$inClique_0.05 +
    0)
edges_all$inClique_0.01 <- as.integer(edges_all$inClique_0.01 +
    0)

save(edges_all, file = "GTEx_edges_all_RIMBAnet_cliqueAdded.RData")
```

We are not displaying the figure here, however, below is the code to generate Figure 2C:

```
load("STARNET_edges_all_RIMBAnet_cliqueAdded.RData")
edges_all_starnet = edges_all
rm(edges_all)
load("GTEx_edges_all_RIMBAnet_cliqueAdded.RData")
edges_all_gtex = edges_all
edges_all_gtex$inClique_0.1 = edges_all_gtex$inClique_0.1 + 0
rm(edges_all)

edges_all_starnet$edge_name = paste(edges_all_starnet$node1,
    edges_all_starnet$node2, sep = "_")
edges_all_gtex$edge_name = paste(edges_all_gtex$node1, edges_all_gtex$node2,
    sep = "_")

fraction_tp_fp_fn <- function(edges_all, name, complete_name = "100_1",
    prob, clique_col, In_clique = 0) {
    tp = length(intersect(unique(edges_all$edge_name[edges_all$name ==
        complete_name]), edges_all$edge_name[edges_all$name ==
        name & edges_all$prob == prob & edges_all[, clique_col] ==
        In_clique]))
    fp = length(setdiff(edges_all$edge_name[edges_all$name ==
        name & edges_all$prob == prob & edges_all[, clique_col] ==
        In_clique], unique(edges_all$edge_name[edges_all$name ==
        complete_name])))
    fn = length(setdiff(unique(edges_all$edge_name[edges_all$name ==
        complete_name]), edges_all$edge_name[edges_all$name ==
        name & edges_all$prob == prob & edges_all[, clique_col] ==
        In_clique]))
    return(data.frame(TP = tp, FP = fp, FN = fn))
}

edges_all_starnet$name = paste(edges_all_starnet$data_type, edges_all_starnet$repRun,
    sep = "_")
edges_all_gtex$name = paste(edges_all_gtex$data_type, edges_all_gtex$repRun,
    sep = "_")
edges_all_starnet$prob = as.numeric(as.character(edges_all_starnet$postProb))
edges_all_gtex$prob = as.numeric(as.character(edges_all_gtex$postProb))

fig_2c_data = c()
for (prob in unique(edges_all_starnet$prob)) {
    # cat('----',prob,'----\n')
    for (subsampling in c(50, 80, 90, 100)) {
        for (rep in 1:5) {
            for (clique_col in c("inClique_0.01", "inClique_0.05",
                "inClique_0.1")) {

                gtex_NoClique = cbind(data.frame(prob = prob,
                  name = subsampling, run = rep, Data_source = "GTEx",
                  clique = clique_col, InClique = "No"), fraction_tp_fp_fn(edges_all_gtex,
                  paste(subsampling, rep, sep = "_"), "100_1",
                  prob, clique_col, 0))
                gtex_YesClique = cbind(data.frame(prob = prob,
                  name = subsampling, run = rep, Data_source = "GTEx",
```

```
                  clique = clique_col, InClique = "Yes"), fraction_tp_fp_fn(edges_all_gtex,
                  paste(subsampling, rep, sep = "_"), "100_1",
                  prob, clique_col, 1))

                starnet_NoClique = cbind(data.frame(prob = prob,
                  name = subsampling, run = rep, Data_source = "STARNET",
                  clique = clique_col, InClique = "No"), fraction_tp_fp_fn(edges_all_starnet,
                  paste(subsampling, rep, sep = "_"), "100_1",
                  prob, clique_col, 0))
                starnet_YesClique = cbind(data.frame(prob = prob,
                  name = subsampling, run = rep, Data_source = "STARNET",
                  clique = clique_col, InClique = "Yes"), fraction_tp_fp_fn(edges_all_starnet,
                  paste(subsampling, rep, sep = "_"), "100_1",
                  prob, clique_col, 1))
                fig_2c_data = rbind(fig_2c_data, gtex_NoClique,
                  gtex_YesClique, starnet_NoClique, starnet_YesClique)
            }
        }
    }
}

fig_2c_data$precision = fig_2c_data$TP/(fig_2c_data$TP + fig_2c_data$FP)
fig_2c_data$recall = fig_2c_data$TP/(fig_2c_data$TP + fig_2c_data$FN)
fig_2c_data = fig_2c_data[!is.na(fig_2c_data$precision), ]

fig_2c_summary = summarySE(fig_2c_data, "precision", groupvars = c("prob",
    "name", "Data_source", "clique", "InClique"))
fig_2c_summary$name = as.character(fig_2c_summary$name)
fig2c <- ggplot(fig_2c_summary, aes(prob)) + geom_line(aes(y = precision,
    colour = name)) + geom_ribbon(aes(ymin = precision - sd,
    ymax = precision + sd, fill = name), alpha = 0.6) + facet_wrap(~clique +
    Data_source + InClique, ncol = 4) + theme_bw() + scale_fill_manual(values = c("black",
    "cyan4", "magenta", "yellow4")) + scale_color_manual(values = c("black",
    "cyan4", "magenta", "yellow4")) + scale_x_continuous(breaks = seq(0.1,
    0.9, 0.4))
show(fig2c)
```

Figure 3: Simulation Data Precision / Recall

```
load("Simulation_edges_all_RIMBAnet.RData")
real_network = read.delim(paste(gitPath, "data/Simulation_true_network.txt",
    sep = ""), header = F)
colnames(real_network) = c("node1", "node2")

tpr <- function(edges, real_network) {
    edges_test = paste(edges$node1, edges$node2, sep = "_")
    edges_real = paste(real_network$node1, real_network$node2,
        sep = "_")
    return(length(intersect(edges_test, edges_real))/length(edges_test))
}
```

```
fdr <- function(edges, real_network) {
    edges_test = paste(edges$node1, edges$node2, sep = "_")
    edges_real = paste(real_network$node1, real_network$node2,
        sep = "_")
    return(length(setdiff(edges_test, edges_real))/length(edges_test))
}

recall <- function(edges, real_network) {
    edges_test = paste(edges$node1, edges$node2, sep = "_")
    edges_real = paste(real_network$node1, real_network$node2,
        sep = "_")
    return(length(intersect(edges_test, edges_real))/length(edges_real))
}

tpr_res = data.frame()
fdr_res = data.frame()
recall_res = data.frame()
for (p in seq(0.1, 0.9, 0.1)) {
    for (name in c(10, 50, 80, 90, 100)) {
        for (rep in 1:5) {
            test_data = edges_all[edges_all$postProb == p & edges_all$data_type ==
                name & edges_all$repRun == rep, ]
            fdr_res = rbind(fdr_res, data.frame(name = name,
                repRun = rep, postProb = p, fdr = fdr(test_data,
                  real_network)))
            tpr_res = rbind(tpr_res, data.frame(name = name,
                repRun = rep, postProb = p, tpr = tpr(test_data,
                  real_network)))

            recall_res = rbind(recall_res, data.frame(name = name,
                repRun = rep, postProb = p, recall = recall(test_data,
                  real_network)))
        }
    }
}

library(ggplot2)
tpr_res$repRun = factor(tpr_res$repRun)
tpr_res2 = tpr_res[is.element(tpr_res$postProb, c(seq(0.1, 0.9,
    0.2))), ]
tpr_res2$name = tpr_res2$name * 10

tpr_sum = summarySE(tpr_res[is.element(tpr_res$postProb, c(seq(0.1,
    0.9, 0.4))), ], "tpr", groupvars = c("name", "postProb"))

tpr_sum$postProb2 = as.character(tpr_sum$postProb)
tpr_sum$name = tpr_sum$name * 10

fdr_sum = summarySE(fdr_res[is.element(fdr_res$postProb, c(seq(0.1,
    0.9, 0.4))), ], "fdr", groupvars = c("name", "postProb"))
fdr_res$repRun = factor(fdr_res$repRun)
fdr_res2 = fdr_res[is.element(fdr_res$postProb, c(seq(0.1, 0.9,
    0.2))), ]
```

```
fdr_res2$name = fdr_res2$name * 10

recall_sum = summarySE(recall_res[is.element(recall_res$postProb,
    c(seq(0.1, 0.9, 0.4))), ], "recall", groupvars = c("name",
    "postProb"))
recall_sum$postProb2 = as.character(recall_sum$postProb)
recall_sum$name = recall_sum$name * 10

## precision recall:
tpr_res$type = "Precision"
recall_res$type = "Recall"
colnames(tpr_res)[colnames(tpr_res) == "tpr"] = "value"
colnames(recall_res)[colnames(recall_res) == "recall"] = "value"
total = rbind(tpr_res, recall_res)

total_sum = summarySE(total, "value", groupvars = c("name", "postProb",
    "type"))
total_sum$postProb2 = as.character(total_sum$postProb)
total_sum$postProb_type = paste(total_sum$postProb2, total_sum$type,
    sep = "_")
total_sum$name = total_sum$name * 10
```

Figure 3:

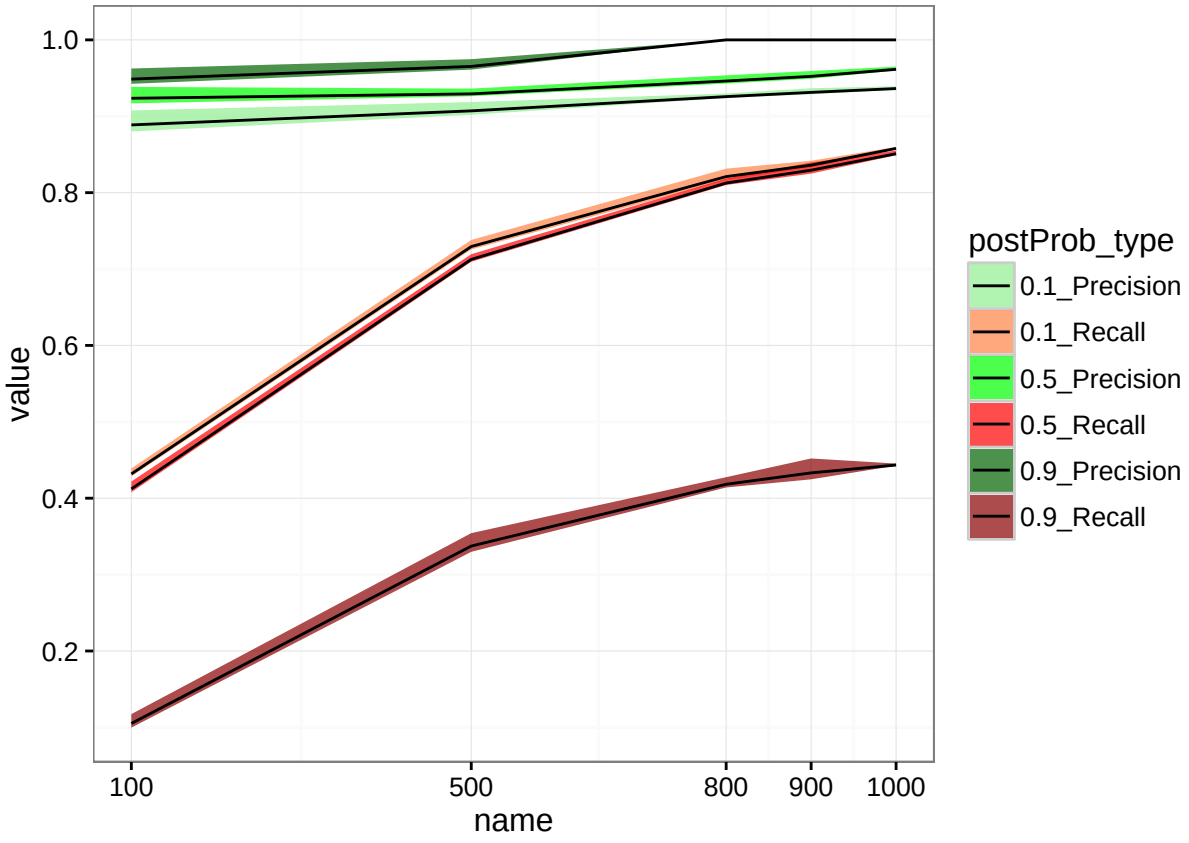

Figure 4: Reproducibility of Key Driver Nodes (KDs)

```
options(stringsAsFactors = F)
rm(list=ls())

library(ggplot2)
library("igraph")
library(plyr)
gitPath <- c("https://raw.githubusercontent.com/divara01/PSB2017_ReproducibilityOfBNs/")
## setting up the files for KD Calling --
load(paste(gitPath, "results/", "GTEx_edges_all_RIMBAnet_cliqueAdded.RData", sep = ""))

edges_all$type_name = paste(edges_all$data_type,edges_all$repRun,sep="_")

for(p in unique(edges_all$postProb)){
  for(d in unique(edges_all$type_name)){
    write.table(edges_all[edges_all$postProb == p & edges_all$type_name == d,
                          c('node1', 'node2')], sep = "\t", quote = F, row.names = F,
                col.names = F, file = paste("BN", d, p, "edges_GTEx.txt", sep="_"))
  }
}

## setting up the files for KD Calling --
load(paste(gitPath, "results/", "STARNET_edges_all_RIMBAnet_cliqueAdded.RData", sep = ""))

edges_all$type_name = paste(edges_all$data_type,edges_all$repRun,sep="_")

for(p in unique(edges_all$postProb)){
  for(d in unique(edges_all$type_name)){
    write.table(edges_all[edges_all$postProb == p & edges_all$type_name == d,
                          c('node1', 'node2')],sep = "\t", quote = F, row.names = F,
                col.names = F, file = paste("BN", d, p, "edges_STARNET.txt", sep = "_"))
  }
}

options(stringsAsFactors=F)
rm(list=ls())
args <- commandArgs(trailingOnly=TRUE) # disease = list of diseases split by ",", outputName

datasets <- c("GTEx", "STARNET")

for(datasetIndex in datasets)
  #Load network edge file
  load(paste(gitPath, "results/", datasetIndex, "edges_all_RIMBAnet_cliqueAdded.RData", sep = ""))
  Edge_File <- edges_all[, c(1, 2)]
  colnames(Edge_File) <- c("From", "To")

  #Create network in R
  #Determine connectivity between all genes in BN

  InverseLengthAllShortestPaths <- vector("list", 1)

  edgeDF_i <- Edge_File
```

```
  g_i <- graph.data.frame(d = edgeDF_i, directed = TRUE)

  summedInverseLengthAllShortestPaths <- vector("list", length(V(g_i)))

  for(node_i in 1:length(V(g_i))){
    SP_i <- get.all.shortest.paths(g_i, node_i, to = V(g_i), mode = "out")
    if(length(SP_i$res) > 0){
      #This tells you how many nodes between the starting node to the target node
      shortestPathLength_i <- sapply(SP_i$res, length)
      #Identifies the final target gene in the path
      target_i <- sapply(SP_i$res, function(x) tail(x,1))
      #number of shortest paths to the target
      spPerTarget <- split(x = shortestPathLength_i, f = target_i)
      #Calculates the score for each target gene from the source gene (1/#shortest paths)
      scorePerTarget <- sapply(spPerTarget, function(x) (1/unique(x)) * length(x))
      #Assigns a score of 0 to all genes in the subgraph
      op_i <- rep(0, length(V(g_i)))
      #This adds the score to the genes that were included in the shortest paths.
      #Remaining genes are still 0
      op_i[as.numeric(names(scorePerTarget))] <- as.numeric(scorePerTarget)
      summedInverseLengthAllShortestPaths[[node_i]] <- op_i
    }
  }
  #This converts the list of scores into a dataframe
  summedInverseLengthAllShortestPaths <- ldply(summedInverseLengthAllShortestPaths, rbind)
  #gene IDs are provided as row names
  rownames(summedInverseLengthAllShortestPaths) <- V(g_i)$name
  #gene IDs are provided as column names
  colnames(summedInverseLengthAllShortestPaths) <- V(g_i)$name

  totalScorePerGene <- rowSums(summedInverseLengthAllShortestPaths)

  quantileThr <- quantile(rowSums(summedInverseLengthAllShortestPaths), 0.95)

  KeyDrivers <- totalScorePerGene[which(totalScorePerGene >= quantileThr)]

  KD_flag <- as.integer(names(totalScorePerGene) %in% names(KeyDrivers))

  perc.rank <- function(x, xo)  length(x[x <= xo])/length(x)*100

  KDpercentile <- sapply(totalScorePerGene, function(x) perc.rank(totalScorePerGene, x))

  top_KDs = KDpercentile[KDpercentile >= 95]
  top_KDs = sort(top_KDs,decreasing=T)
  outputFN = paste(datasetIndex,"_KDs.txt",sep="")
  write.table(top_KDs,file = outputFN, sep="\t",quote=F, row.names=T, col.names=F)
}

options(stringsAsFactors=F)
rm(list=ls())
files = list.files(".","_STARNET_KDs.txt")
all_kda = c()
```

```
for(file in files){
  x = read.delim(file,header=F)
  data_type = strsplit(file,"_")[[1]][2]
  repRun = strsplit(file,"_")[[1]][3]
  postProb = strsplit(file,"_")[[1]][4]
  all_kda = rbind(all_kda, data.frame(gene = x[,1], data_type = data_type, repRun = repRun,
                                      postProb = postProb))
}

load(paste(gitPath, "results/", "STARNET_edges_all_RIMBAnet_cliqueAdded.RData", sep = ""))
head(edges_all)

edges_all$KDflagNode1_redo = sapply(1:nrow(edges_all),function(x){
  is.element(edges_all$node1[x],all_kda$gene[all_kda$repRun == edges_all$repRun[x] &
                                               all_kda$data_type == edges_all$data_type[x] &
                                               all_kda$postProb == edges_all$postProb[x]])
})

table(edges_all$KDflagNode1,edges_all$KDflagNode1_redo)

save(edges_all, file = "STARNET_edges_all_RIMBAnet_cliqueAdded_KDAdded.RData")

## reading in the KD Results --
options(stringsAsFactors=F)
rm(list=ls())
files = list.files(".","_GTEx_KDs.txt")
all_kda = c()
for(file in files){
  x = read.delim(file,header=F)
  data_type = strsplit(file,"_")[[1]][2]
  repRun = strsplit(file,"_")[[1]][3]
  postProb = strsplit(file,"_")[[1]][4]
  all_kda = rbind(all_kda, data.frame(gene = x[,1] , data_type = data_type,repRun = repRun,
                                      postProb = postProb))
}

load( "GTEx_edges_all_RIMBAnet_cliqueAdded.RData")
head(edges_all)

edges_all$KDflagNode1_redo = sapply(1:nrow(edges_all),function(x){
  is.element(edges_all$node1[x], all_kda$gene[all_kda$repRun == edges_all$repRun[x] &
                                                all_kda$data_type == edges_all$data_type[x] &
                                                all_kda$postProb == edges_all$postProb[x]])
})

table(edges_all$KDflagNode1,edges_all$KDflagNode1_redo)

save(edges_all, file = "GTEx_edges_all_RIMBAnet_cliqueAdded_KDAdded.RData")
```

Generate Node score for all nodes:

```
#Calculate Most Connected Node Conservation
datasets <- c("GTEx", "STARNET")
```

```
for(datasetIndex in 1:length(datasets)){
  message(paste(datasets[datasetIndex], "running..."))
  #loads in the edges_all file
  load(paste(datasets[datasetIndex], "_edges_all_RIMBAnet_cliqueAdded_KDAdded.RData", sep =""))

  allNodes <- unique(as.vector(unlist(edges_all[, c(1:2)])))
  totalRows <- length(unique(edges_all$postProb)) *
    length(unique(edges_all$data_type)) *
    length(unique(edges_all$repRun)) *
    length(allNodes)

  postProb <- rep(sort(as.vector(unique(edges_all$postProb))),
                  totalRows/length(unique(edges_all$postProb)))

  subsets <- rep(as.vector(sapply(unique(edges_all$data_type), function(x)
    rep(x, length(unique(edges_all$postProb)) *
          length(unique(edges_all$repRun))))), length(allNodes))

  nodes <- as.vector((sapply(allNodes, function(x)
    rep(x, length(unique(edges_all$data_type)) * length(unique(edges_all$repRun)) *
          length(unique(edges_all$postProb))))))

  repRun <- rep(as.vector(sapply(as.vector(unique(edges_all$repRun)), function(x)
    rep(x, length(unique(edges_all$postProb))))), length(allNodes) *
      length(unique(edges_all$data_type)))

  nodeTable <- as.data.frame(cbind(nodes, postProb, subsets, repRun))

  message(paste(datasets[datasetIndex], "evaluating each node's connections..."))
  for(i in 1:nrow(nodeTable)){
    node_i <- nodeTable$nodes[i]
    postProb_i <- nodeTable$postProb[i]
    subset_i <- nodeTable$subsets[i]
    repRun_i <- nodeTable$repRun[i]

    nodeTable$outDegree[i] <- length(which(edges_all$node1 == node_i &
                                             edges_all$postProb == postProb_i &
                                             edges_all$data_type == subset_i &
                                             edges_all$repRun == repRun_i))
    nodeTable$inDegree[i] <- length(which(edges_all$node2 == node_i &
                                            edges_all$postProb == postProb_i &
                                            edges_all$data_type == subset_i &
                                            edges_all$repRun == repRun_i))
    nodeTable$allDegree[i] <- nodeTable$outDegree[i] + nodeTable$inDegree[i]
    if(i %% 500 == 0){message(paste(nrow(nodeTable) - i, " remaining..."))}
  }

  assign(paste(datasets[datasetIndex], "allNodeScores", sep = "_"), nodeTable)
  write.table(nodeTable, file = paste(datasets[datasetIndex], "_allNodeScores", sep = ""),
              sep = "\t", row.names = FALSE, col.names = TRUE)
  message(paste(datasets[datasetIndex], "complete!"))
}
```

```
#Append values to the edges_all table
for(datasetIndex in 1:length(datasets)){
  message(paste(datasets[datasetIndex], "running..."))
  #loads in the edges_all file
  load(paste(datasets[datasetIndex], "_edges_all_RIMBAnet_cliqueAdded_KDAdded.RData", sep =""))
  nodeTable <- read.delim(paste(datasets[datasetIndex], "_allNodeScores", sep = ""), sep = "\t",
                          header = TRUE)

  edges_all$outDegree <- NA
  edges_all$inDegree <- NA
  edges_all$allDegree <- NA

  for(i in 1:nrow(edges_all)){
    node1_i <- edges_all$node1[i]
    postProb_i <- edges_all$postProb[i]
    repRun_i <- edges_all$repRun[i]
    data_type_i <- edges_all$data_type[i]

    nodeTablePos <- which(nodeTable$nodes == node1_i &
                            nodeTable$postProb == postProb_i &
                            nodeTable$subsets == data_type_i &
                            nodeTable$repRun == repRun_i)

    edges_all$outDegree[i] <- nodeTable$outDegree[nodeTablePos]
    edges_all$inDegree[i] <- nodeTable$inDegree[nodeTablePos]
    edges_all$allDegree[i] <- nodeTable$allDegree[nodeTablePos]

    if(i %% 500 == 0){message(paste(nrow(edges_all) - i, " remaining..."))}
  }

  edgesAll_filename <- paste(dataset[datasetIndex],

                             "_edges_all_RIMBAnet_cliqueAdded_KDAdded_HubNodes.RData", sep = "")

  save(edges_all,file=edgesAll_filename)
  message(paste(datasets[datasetIndex], "Complete!"))
}
```

Figure 5: Hub Nodes Reproducibility Rate

```
options(stringsAsFactors = F)
rm(list = ls())
require(ggplot2)
require(plyr)
require(gridExtra)
require(grid)
library(ggplot2)

gtex = read.delim("GTEx_allNodeScores")
starnet = read.delim("STARNET_allNodeScores")
gtex = gtex[gtex$allDegree != 0, ]
starnet = starnet[starnet$allDegree != 0, ]
```

```
starnet_fn = "STARNET_edges_all_RIMBAnet_cliqueAdded_KDAdded_HubNodes.RData"
gtex_fn = "GTEx_edges_all_RIMBAnet_cliqueAdded_KDAdded_HubNodes.RData"

load(starnet_fn)
edges_all_starnet = edges_all
rm(edges_all)

load(gtex_fn)
edges_all_gtex = edges_all
rm(edges_all)

edges_all_starnet$name = paste(edges_all_starnet$data_type, edges_all_starnet$repRun,
    sep = "_")
edges_all_gtex$name = paste(edges_all_gtex$data_type, edges_all_gtex$repRun,
    sep = "_")

edges_all_starnet = edges_all_starnet[edges_all_starnet$postProb !=
    1, ]
edges_all_gtex = edges_all_gtex[edges_all_gtex$postProb != 1,
    ]

gtex = gtex[gtex$postProb != 1, ]
starnet = starnet[starnet$postProb != 1, ]

## calculate the top connected out degree nodes:
gtex_top = data.frame()
starnet_top = data.frame()
for (p in unique(gtex$postProb)) {
    for (name in c(50, 80, 90, 100)) {
        for (rep in 1:5) {
            if (name == 100) {
                # for the complete networks, taking the same cutoff as the
                # 100_1 in all replicates because edge number is so similar
                # to avoid the issue of slight biase by different cutoffs.
                gtex_top = rbind(gtex_top, data.frame(postProb = p,
                  data_type = name, repRun = rep, outCutoff = quantile(gtex$outDegree[gtex$repRun ==
                    1 & gtex$postProb == p & gtex$subsets ==
                    name], 0.9)[[1]]))
                starnet_top = rbind(starnet_top, data.frame(postProb = p,
                  data_type = name, repRun = rep, outCutoff = quantile(starnet$outDegree[starnet$repRun
                    1 & starnet$postProb == p & starnet$subsets ==
                    name], 0.9)[[1]]))
            } else {
                gtex_top = rbind(gtex_top, data.frame(postProb = p,
                  data_type = name, repRun = rep, outCutoff = quantile(gtex$outDegree[gtex$repRun ==
                    rep & gtex$postProb == p & gtex$subsets ==
                    name], 0.9)[[1]]))
                starnet_top = rbind(starnet_top, data.frame(postProb = p,
                  data_type = name, repRun = rep, outCutoff = quantile(starnet$outDegree[starnet$repRun
                    rep & starnet$postProb == p & starnet$subsets ==
                    name], 0.9)[[1]]))
            }
        }
```

```
    }
}

gtex$name = paste(gtex$subsets, gtex$repRun, gtex$postProb, sep = "_")
starnet$name = paste(starnet$subsets, starnet$repRun, starnet$postProb,
    sep = "_")
starnet_top$name = paste(starnet_top$data_type, starnet_top$repRun,
    starnet_top$postProb, sep = "_")
gtex_top$name = paste(gtex_top$data_type, gtex_top$repRun, gtex_top$postProb,
    sep = "_")

## for each node1 assign if it is in the top connected node:
edges_all_starnet$name = paste(edges_all_starnet$data_type, edges_all_starnet$repRun,
    edges_all_starnet$postProb, sep = "_")
edges_all_gtex$name = paste(edges_all_gtex$data_type, edges_all_gtex$repRun,
    edges_all_starnet$postProb, sep = "_")

starnet_top1 = starnet_top$outCutoff
names(starnet_top1) = starnet_top$name
edges_all_starnet$TopConnNode1 = sapply(1:nrow(edges_all_starnet),
    function(x) {
        is.element(edges_all_starnet$node1[x], starnet$nodes[starnet$name ==
            edges_all_starnet$name[x] & starnet$outDegree >=
            starnet_top1[edges_all_starnet$name[x]]])
    })

gtex_top1 = gtex_top$outCutoff
names(gtex_top1) = gtex_top$name
edges_all_gtex$TopConnNode1 = sapply(1:nrow(edges_all_gtex),
    function(x) {
        is.element(edges_all_gtex$node1[x], gtex$nodes[gtex$name ==
            edges_all_gtex$name[x] & gtex$outDegree >= gtex_top1[edges_all_gtex$name[x]]])
    })

kda = rbind(data.frame(unique(edges_all_gtex[, c("node1", "postProb",
    "repRun", "data_type", "TopConnNode1")]), Data_source = "GTEx"),
    data.frame(unique(edges_all_starnet[, c("node1", "postProb",
        "repRun", "data_type", "TopConnNode1")]), Data_source = "STARNET"))
kda$name = paste(kda$Data_source, kda$data_type, kda$repRun,
    sep = "_")
kda$postProb = kda$postProb

kda = kda[kda$TopConnNode1 & kda$postProb != 1, ]
getFractionOverlap = function(kda, A_name, B_name, prob, fixed = TRUE) {
    if (A_name == B_name) {
        A_size = strsplit(A_name, "_")[[1]][2]
        B_name = setdiff(unique(kda$name[kda$data_type != A_size]),
            A_name)
    }
    if (fixed) {
        AandB = length(intersect(kda$node1[kda$name == A_name &
            kda$postProb == prob], kda$node1[is.element(kda$name,
            B_name) & kda$postProb == prob]))
```

```
    } else {
        AandB = length(intersect(kda$node1[kda$name == A_name &
            kda$postProb == prob], kda$node1[is.element(kda$name,
            B_name)]))
    }
    a = length(unique(kda$node1[kda$name == A_name & kda$postProb ==
        prob]))

    return(AandB/a)
}

kda_fractions = data.frame()
subsamp = c(50, 80, 90, 100)
for (p in unique(kda$postProb)) {
    for (sub in unique(kda$data_type)) {
        for (k in unique(kda$repRun)) {
            for (fixed in c(TRUE, FALSE)) {
                for (dataSet in unique(kda$Data_source)) {
                  # cat('on ',dataSet,sub, k, fixed,p,'\n')
                  if (sub == 100 & k == "1") {
                    break
                  }
                  kda_fractions = rbind(kda_fractions, data.frame(Data_source = dataSet,
                    data_type = sub, repRun = k, TP_fraction = getFractionOverlap(kda,
                      paste(dataSet, sub, k, sep = "_"), paste(dataSet,
                        100, 1, sep = "_"), p, fixed), fixed = fixed,
                    postProb = p, splitBy = "No Split"))
                }
            }
        }
    }
}

kda_fractions$repRun = factor(kda_fractions$repRun)
kda_fractions$fixed = factor(kda_fractions$fixed, levels = c("TRUE",
    "FALSE"))

kda_fractions$postProb = factor(kda_fractions$postProb)
kda_fractions$data_type = as.character(kda_fractions$data_type)
kda_fractions2 = kda_fractions[is.element(kda_fractions$postProb,
    c(seq(0.1, 0.5, 0.1))), ]

kda_summary <- summarySE(kda_fractions, "TP_fraction", groupvars = c("fixed",
    "postProb", "data_type", "Data_source"))

kda_summary$data_type = as.character(kda_summary$data_type)
kda_summary$postProb = as.numeric(as.character(kda_summary$postProb))

fig5 <- ggplot(kda_summary, aes(postProb)) + geom_line(aes(y = TP_fraction,
    colour = data_type), size = 1) + geom_ribbon(aes(ymin = TP_fraction -
    sd, ymax = TP_fraction + sd, fill = data_type), alpha = 0.6) +
    scale_color_manual(values = c("black", "cyan4", "magenta",
        "yellow4")) + scale_fill_manual(values = c("black", "cyan4",
```

```
    "magenta", "yellow4")) + facet_wrap(~fixed + Data_source,
    ncol = 2) + theme_bw() + scale_x_continuous(breaks = seq(0.1,
    0.9, 0.4))

show(fig5)
```

Table 2: Compare 5 Replicates of 100% Networks

```
options(stringsAsFactors = F)
rm(list = ls())

load(file = "STARNET_edges_all_RIMBAnet_cliqueAdded_KDAdded_HubNodes.RData")
edges_all_starnet = edges_all
edges_all_starnet$data_source = "STARNET"
rm(edges_all)
load(file = "GTEx_edges_all_RIMBAnet_cliqueAdded_KDAdded_HubNodes.RData")
edges_all_gtex = edges_all
edges_all_gtex$data_source = "GTEx"
rm(edges_all)

edges_all = rbind(edges_all_starnet, edges_all_gtex)

edges_all$unique_name = paste(edges_all$data_source, edges_all$data_type,
    edges_all$repRun, sep = "_")
edges_all$edge_name = paste(edges_all$node1, edges_all$node2,
    sep = "_")
edges_all = edges_all[edges_all$data_type == 100, ]
edges_all = edges_all[edges_all$postProb != 1, ]

testsToRun = combn(1:5, 2)

ab_ov_a = function(edges_all, a_name, b_name, prob1 = p, prob2 = p2) {
    ab = length(intersect(edges_all$edge_name[edges_all$unique_name ==
        a_name & edges_all$postProb == prob1], edges_all$edge_name[edges_all$unique_name ==
        b_name & edges_all$postProb == prob2]))
    a = sum(edges_all$unique_name == a_name & edges_all$postProb ==
        prob1)
    return(ab/a)
}

comp_res = data.frame()
for (i in 1:ncol(testsToRun)) {
    cat(i, "\n")
    for (p in unique(edges_all$postProb)) {
        comp_res = rbind(comp_res, data.frame(A_name = testsToRun[1,
            i], B_name = testsToRun[2, i], data_source = "GTEx",
            ab_ov_a = ab_ov_a(edges_all, paste("GTEx", 100, testsToRun[1,
                i], sep = "_"), paste("GTEx", 100, testsToRun[2,
                i], sep = "_"), prob1 = p, prob2 = p), postProb = p))
        comp_res = rbind(comp_res, data.frame(A_name = testsToRun[2,
            i], B_name = testsToRun[1, i], data_source = "GTEx",
```

```
            ab_ov_a = ab_ov_a(edges_all, paste("GTEx", 100, testsToRun[2,
                i], sep = "_"), paste("GTEx", 100, testsToRun[1,
                i], sep = "_"), prob1 = p, prob2 = p), postProb = p))

        comp_res = rbind(comp_res, data.frame(A_name = testsToRun[1,
            i], B_name = testsToRun[2, i], data_source = "STARNET",
            ab_ov_a = ab_ov_a(edges_all, paste("STARNET", 100,
                testsToRun[1, i], sep = "_"), paste("STARNET",
                100, testsToRun[2, i], sep = "_"), prob1 = p,
                prob2 = p), postProb = p))
        comp_res = rbind(comp_res, data.frame(A_name = testsToRun[2,
            i], B_name = testsToRun[1, i], data_source = "STARNET",
            ab_ov_a = ab_ov_a(edges_all, paste("STARNET", 100,
                testsToRun[2, i], sep = "_"), paste("STARNET",
                100, testsToRun[1, i], sep = "_"), prob1 = p,
                prob2 = p), postProb = p))
    }
}

tab_out = data.frame(postProb = seq(0.1, 0.9, 0.1), GTEx_mean = NA,
    GTEx_sd = NA, STARNET_mean = NA, STARNET_sd = NA)
tab_out$GTEx_mean = sapply(tab_out$postProb, function(x) {
    mean(comp_res$ab_ov_a[comp_res$data_source == "GTEx" & comp_res$postProb ==
        x])
})

tab_out$GTEx_sd = sapply(tab_out$postProb, function(x) {
    sd(comp_res$ab_ov_a[comp_res$data_source == "GTEx" & comp_res$postProb ==
        x])
})

tab_out$STARNET_mean = sapply(tab_out$postProb, function(x) {
    mean(comp_res$ab_ov_a[comp_res$data_source == "STARNET" &
        comp_res$postProb == x])
})

tab_out$STARNET_sd = sapply(tab_out$postProb, function(x) {
    sd(comp_res$ab_ov_a[comp_res$data_source == "STARNET" & comp_res$postProb ==
        x])
})

write.table(tab_out, sep = "\t", row.names = F, col.names = T,
    file = "Updated_Table3.txt", quote = F)
```

Table 3: Jaccard Index

```
get_jacard <- function(edges_all, name1, name2, prob) {
    aANDb = length(intersect(edges_all$edge_name[edges_all$postProb ==
        prob & edges_all$name == name1], edges_all$edge_name[edges_all$postProb ==
        prob & edges_all$name == name2]))
    a = sum(edges_all$postProb == prob & edges_all$name == name1)
```

```
    b = sum(edges_all$postProb == prob & edges_all$name == name2)

    jacard = aANDb/(a + b - aANDb)
    return(jacard)
}

edges_all_gtex$name = paste(edges_all_gtex$data_type, edges_all_gtex$repRun,
    sep = "_")
edges_all_starnet$name = paste(edges_all_starnet$data_type, edges_all_starnet$repRun,
    sep = "_")
edges_all_gtex$edge_name = paste(edges_all_gtex$node1, edges_all_gtex$node2,
    sep = "_")
edges_all_starnet$edge_name = paste(edges_all_starnet$node1,
    edges_all_starnet$node2, sep = "_")
jaccard_values = data.frame()
combos = combn(5, 2)
for (subsamp in c(50, 80, 90, 100)) {
    # cat('on subsampling',subsamp,'\n')
    for (p in seq(0.1, 0.9, 0.1)) {
        # cat('on prob',p,'\n')
        for (k in 1:ncol(combos)) {
            jaccard_values = rbind(jaccard_values, data.frame(name1 = paste(subsamp,
                combos[1, k], sep = "_"), name2 = paste(subsamp,
                combos[2, k], sep = "_"), prob = p, jaccard = get_jacard(edges_all_gtex,
                paste(subsamp, combos[1, k], sep = "_"), paste(subsamp,
                  combos[2, k], sep = "_"), p), dataSource = "GTEx"))
            jaccard_values = rbind(jaccard_values, data.frame(name1 = paste(subsamp,
                combos[1, k], sep = "_"), name2 = paste(subsamp,
                combos[2, k], sep = "_"), prob = p, jaccard = get_jacard(edges_all_starnet,
                paste(subsamp, combos[1, k], sep = "_"), paste(subsamp,
                  combos[2, k], sep = "_"), p), dataSource = "STARNET"))

            if (subsamp != 100) {
                jaccard_values = rbind(jaccard_values, data.frame(name1 = paste(subsamp,
                  combos[1, k], sep = "_"), name2 = "100_1",
                  prob = p, jaccard = get_jacard(edges_all_gtex,
                    paste(subsamp, combos[1, k], sep = "_"),
                    "100_1", p), dataSource = "GTEx"))
                jaccard_values = rbind(jaccard_values, data.frame(name1 = paste(subsamp,
                  combos[1, k], sep = "_"), name2 = "100_1",
                  prob = p, jaccard = get_jacard(edges_all_starnet,
                    paste(subsamp, combos[1, k], sep = "_"),
                    "100_1", p), dataSource = "STARNET"))
            }
        }
    }
}

jaccard_values_output = jaccard_values[is.element(jaccard_values$prob,
    seq(0.1, 0.9, 0.4)), ]
jaccard_values_output$type1 = sapply(jaccard_values_output$name1,
    function(x) {
        strsplit(x, "_")[[1]][1]
```

```
    })
jaccard_values_output$type2 = sapply(jaccard_values_output$name2,
    function(x) {
        strsplit(x, "_")[[1]][1]
    })

jaccard_table = summarySE(jaccard_values_output, "jaccard", groupvars = c("dataSource",
    "type1", "type2", "prob"))
print(jaccard_table[, c("dataSource", "type1", "type2", "prob",
    "N", "jaccard", "sd")])
```

Table 4: Edge Counts

```
load("Simulation_edges_all_RIMBAnet.RData")
edges_all_sim = edges_all
rm(edges_all)
edges_all_sim$type_name = paste(edges_all_sim$data_type, edges_all_sim$repRun,
    sep = "_")
edges_all_sim$edge_name = paste(edges_all_sim$node1, edges_all_sim$node2,
    sep = "_")

load("GTEx_edges_all_RIMBAnet_cliqueAdded_KDAdded_HubNodes.RData")
edges_all_gtex = edges_all
edges_all_gtex$type_name = paste(edges_all_gtex$data_type, edges_all_gtex$repRun,
    sep = "_")
edges_all_gtex$edge_name = paste(edges_all_gtex$node1, edges_all_gtex$node2,
    sep = "_")
rm(edges_all)

load("STARNET_edges_all_RIMBAnet_cliqueAdded_KDAdded_HubNodes.RData")
edges_all_starnet = edges_all
rm(edges_all)
edges_all_starnet$type_name = paste(edges_all_starnet$data_type,
    edges_all_starnet$repRun, sep = "_")
edges_all_starnet$edge_name = paste(edges_all_starnet$node1,
    edges_all_starnet$node2, sep = "_")

tab4_starnet = table(edges_all_starnet$type_name[is.element(edges_all_starnet$postProb,
    seq(0.1, 0.9, 0.4))], edges_all_starnet$postProb[is.element(edges_all_starnet$postProb,
    seq(0.1, 0.9, 0.4))])
colnames(tab4_starnet) = paste("STARNET", colnames(tab4_starnet))
edges_all_gtex$postProb = as.numeric(as.character(edges_all_gtex$postProb))
tab4_gtex = table(edges_all_gtex$type_name[is.element(edges_all_gtex$postProb,
    seq(0.1, 0.9, 0.4))], edges_all_gtex$postProb[is.element(edges_all_gtex$postProb,
    seq(0.1, 0.9, 0.4))])
colnames(tab4_gtex) = paste("GTEx", colnames(tab4_gtex))

edges_all_sim$postProb = as.numeric(as.character(edges_all_sim$postProb))
tab4_sim = table(edges_all_sim$type_name[is.element(edges_all_sim$postProb,
    seq(0.1, 0.9, 0.4))], edges_all_sim$postProb[is.element(edges_all_sim$postProb,
    seq(0.1, 0.9, 0.4))])
```

```
colnames(tab4_sim) = paste("Sim", colnames(tab4_sim))

tab4_starnet = rbind(matrix(NA, nrow = 5, ncol = 3), tab4_starnet)
tab4_gtex = rbind(matrix(NA, nrow = 5, ncol = 3), tab4_gtex)
rownames(tab4_starnet)[1:5] = rownames(tab4_sim)[1:5]
rownames(tab4_gtex)[1:5] = rownames(tab4_sim)[1:5]
tab4 = cbind(tab4_starnet, tab4_gtex, tab4_sim)
tab4_melt = melt(tab4)
tab4_melt$X1 = as.character(tab4_melt$X1)
tab4_melt$subsampling = sapply(tab4_melt$X1, function(x) {
    strsplit(x, "_")[[1]][1]
})
table4 <- summarySE(tab4_melt, "value", groupvars = c("subsampling",
    "X2"))
print(table4[!is.na(table4$value), c("subsampling", "X2", "N",
    "value", "sd")])
```