Reconstructing Boolean network ensembles from single-cell data for unraveling dynamics in the aging of human hematopoietic stem cells

The best-fit extension The following code chunks demonstrate our new approach with a pipeline to reconstruct Boolean network ensembles from single-cell data. In the following example, we use a published dataset by Ratliff et al., 2020 (cite, GSE138544).

Data Loading and Preprocessing

This data set contains 730 samples of isolated peripheral blood long-term HSCs (LT-HSCs) (lin^{-}CD{34}+ CD{38}ˆ- CD45RAˆ- CD49fˆ+) from four young (ages 19, 21, 37, 40) and four aged (ages 61, 66, 68, 70) human individuals (two males and two females per group) as suggested by Ratliff and colleagues {Ratliff et al., 2020, #69992}. The dataset contains 83 to 94 single-cell measurements per individual. Sequencing was performed on a NovaSeq6000. The available dataset shows log normalized counts per million (Ratliff et al., 2020, BMC Immunity & Aging).

```
#load aged samples from GSE138544
aged <- read.table("GSE138544_Partek_michelle_aged_Normalization_Normalized_counts.txt",
                   sep = "\t",
                   header = T)
rownames(aged) <- aged$Feature
aged <- aged[,-1] #cut feature description
#load young samples from GSE138544
young <- read.table("GSE138544_Partek_michelle_young_Normalization_Normalized_counts.txt",
                    sep = "\t",
                    header = T)
rownames(young) <- young$Feature
young <- young[,-1] #cut feature description

#merged data set of young and aged
gse138544 <- cbind(young,aged)
```

In a second preprocessing-step, all genes related to the NF-κB pathway (according to KEGG-DB) were selected in the dataset. The selection comprises 101 genes belonging to the human NF-κB (hsa04064).

```
library("Biobase")
library("GEOquery")
library("biomaRt")
library("tidyverse")
#function to download pathways from KEGG-DB
getKEGGSetByPathwayName <- function(spec){
  require(dplyr)
  #select species (hsa = hum, mmu = mouse, dme = drosophila, etc..)
  URL <- paste("http://rest.kegg.jp/link/pathway", spec, sep = "/")
  Path.Gene <- read.table(URL, sep = "\t", quote = "\"", fill = TRUE,
                          comment.char = "", stringsAsFactors = FALSE)
  colnames(Path.Gene) <- c("GeneID", "PathwayID")
```

```
  #select species (hsa = hum, mmu = mouse, etc..)
  URLname <- paste("http://rest.kegg.jp/list/pathway", spec,

                   sep = "/")
  Path.Name <- read.table(URLname, sep = "\t", quote = "\"", fill = TRUE,
                          comment.char = "", stringsAsFactors = FALSE)
  colnames(Path.Name) <- c("PathwayID","PathwayName")
  Path.Gene <- left_join(Path.Gene,Path.Name,by="PathwayID")
  Path.Gene[, 1] <- sub(paste0("^", "spec", ":"),
                        "", Path.Gene[, 1])

  return(Path.Gene)
}


#select pathway
kegg <- getKEGGSetByPathwayName("hsa")
nfkb <- kegg[grepl("NF-kappa B", kegg$PathwayName),]
nfkb$GeneID <- sapply(strsplit(nfkb$GeneID,"hsa:"), function(s) s[[2]])
#mapping
ensembl <- useMart(biomart= "ensembl", dataset="hsapiens_gene_ensembl")
tab_ensembl <- c("entrezgene_id", "ensembl_gene_id", "external_gene_name", "hgnc_symbol")
mapping <- getBM(attributes= tab_ensembl, mart= ensembl, values = "*", uniqueRows=T)
mapper <- data.frame(na.omit(mapping))
#select data
selected <- mapper[which(mapper$entrezgene_id %in% nfkb$GeneID),]
idc <- which(rownames(gse138544) %in% selected$hgnc_symbol)
#get relevant subset of data
selectedData <- gse138544[idc,]
```

As final preprocessing step, the data was binarized using the BASCA algorithm form the R-package BiTrinA. BASC significance test was used to evaluate which genes were significantly binarized. 96 out of the 101 genes were selected according to these results.

```
library("BiTrinA")
#binarize data using BASCA,
#FDR method is used to correct BASC significance test for multiple testing
binarizedData <- BiTrinA::binarizeMatrix(selectedData, method="BASCA",
                                         adjustment = "fdr")

#check for significants
signifDat <- binarizedData[binarizedData$p.value < 0.05,1:(ncol(binarizedData)-2)]

#Reorganize data for network reconstruction and split split by individual
agedA <- signifDat[,grepl("Aged_A",colnames(signifDat))]
agedB <- signifDat[,grepl("Aged_B",colnames(signifDat))]
agedC <- signifDat[,grepl("Aged_C",colnames(signifDat))]
agedD <- signifDat[,grepl("Aged_D",colnames(signifDat))]

youngA <- signifDat[,grepl("Yg_A",colnames(signifDat))]
youngB <- signifDat[,grepl("Yg_B",colnames(signifDat))]
youngC <- signifDat[,grepl("Yg_C",colnames(signifDat))]
youngD <- signifDat[,grepl("Yg_D",colnames(signifDat))]
```

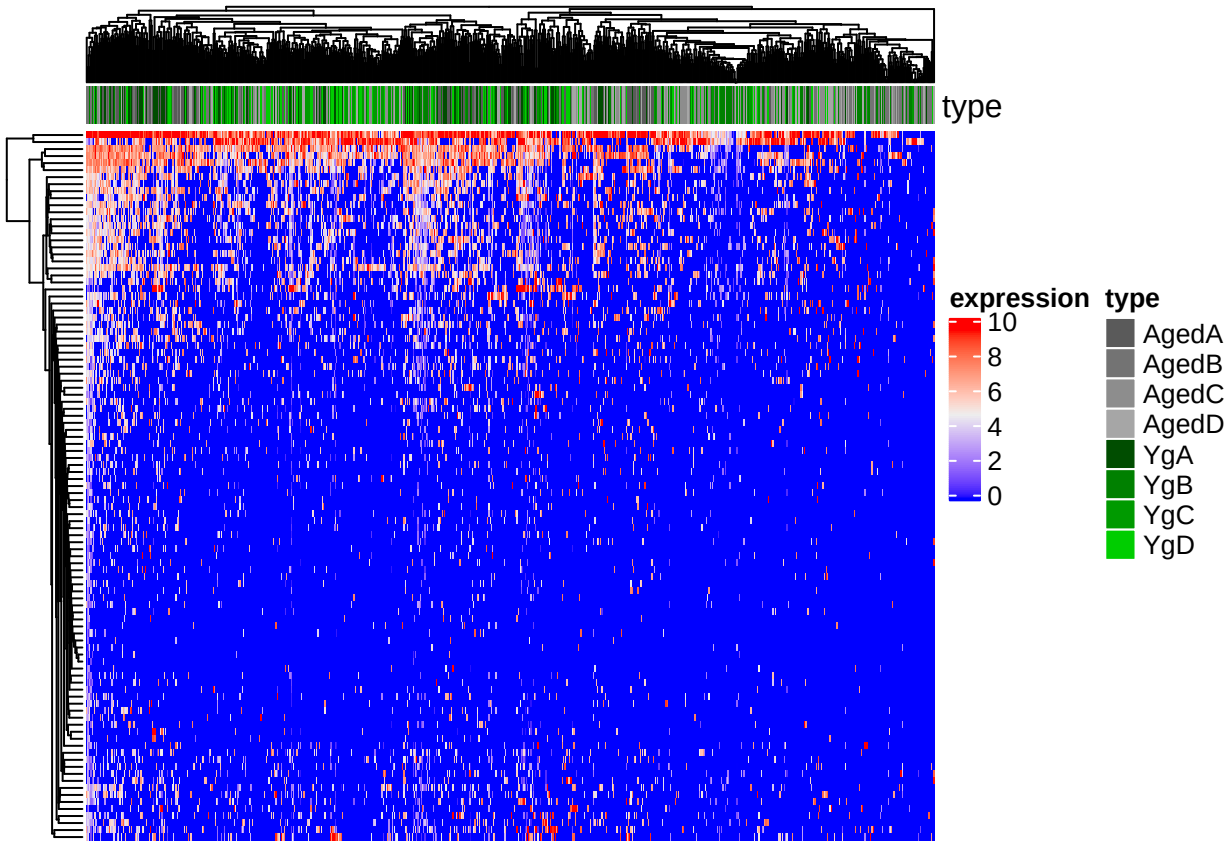
```
splittedData <- list(aged=list(a = agedA, b = agedB, c = agedC, d = agedD),
                     young=list(a = youngA, b = youngB, c = youngC, d = youngD))
```

Expression Data Analysis

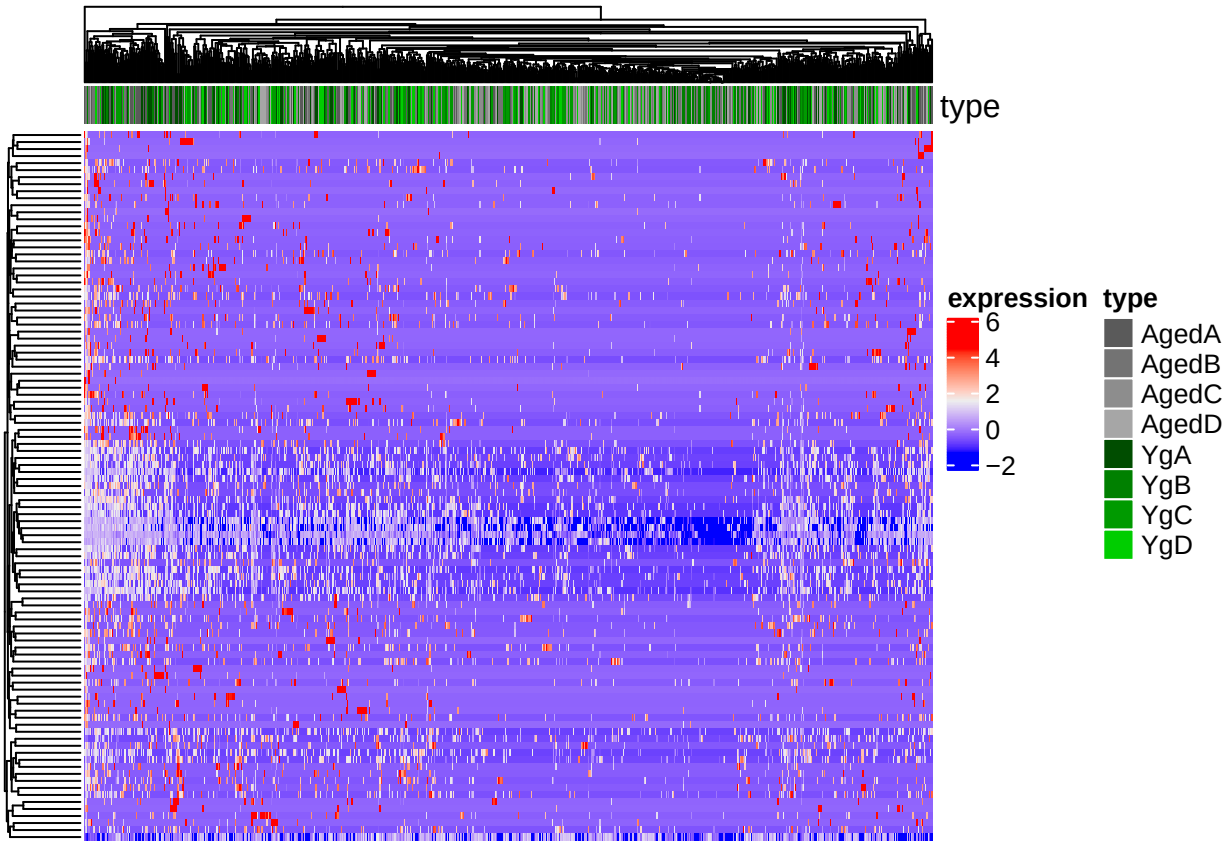
In a first step of data analysis, we investigated differences in the expression data among the different individuals and age groups. Thus, we plotted expression heatmaps clustering the different samples by similarity. This was repeatedly done for the log-normalized original data as provided in and the z-transformed data. Additionally, we performed tSNE analysis on this data.

Clustered Expression Data

```
library(ComplexHeatmap)
#generate matrix from data and create colData object with metadata
orderByAge <- order(sapply(strsplit(colnames(signifDat),split = "_"),
                           function(str) paste0(str[-1],collapse = "")))
mat <- as.matrix(selectedData[,orderByAge])
colData <- data.frame(samples = colnames(signifDat)[orderByAge],
                      individual = sapply(strsplit(colnames(signifDat)[orderByAge],
                                                   split = "_"),
                                          function(str) paste0(str[-1],collapse = "")))
type <- colData$individual
#set annotation for data
ha <- HeatmapAnnotation(
  df = data.frame(type = type),
  annotation_height = unit(4, "mm"),
  col = list(type= c(AgedA="#5A5A5A",AgedB="#737373",AgedC="#8d8d8d",AgedD="#a6a6a6",
                     YgA="#004d00",YgB="#008000",YgC="#009a00",YgD="#00cd00"))
)
#plot data
Heatmap(mat, name = "expression", top_annotation = ha,
        show_row_names = FALSE, show_column_names = FALSE)
```




```
#repeat with z-normalized data
library(ComplexHeatmap)
orderByAge <- order(sapply(strsplit(colnames(signifDat),split = "_"),
                           function(str) paste0(str[-1],collapse = "")))
#z transform data
mat <- t(scale(t(as.matrix(selectedData[,orderByAge]))))
colData <- data.frame(samples = colnames(signifDat)[orderByAge],
                      individual = sapply(strsplit(colnames(signifDat)[orderByAge],
                                                   split = "_"),
                                          function(str) paste0(str[-1],collapse = "")))
type <- colData$individual

ha <- HeatmapAnnotation(
  df = data.frame(type = type),
  annotation_height = unit(4, "mm"),
  col = list(type= c(AgedA="#5A5A5A",AgedB="#737373",
                     AgedC="#8d8d8d",AgedD="#a6a6a6",
                     YgA="#004d00",YgB="#008000",
                     YgC="#009a00",YgD="#00cd00"))
)

Heatmap(mat, name = "expression", top_annotation = ha,
        show_row_names = FALSE, show_column_names = FALSE)
```



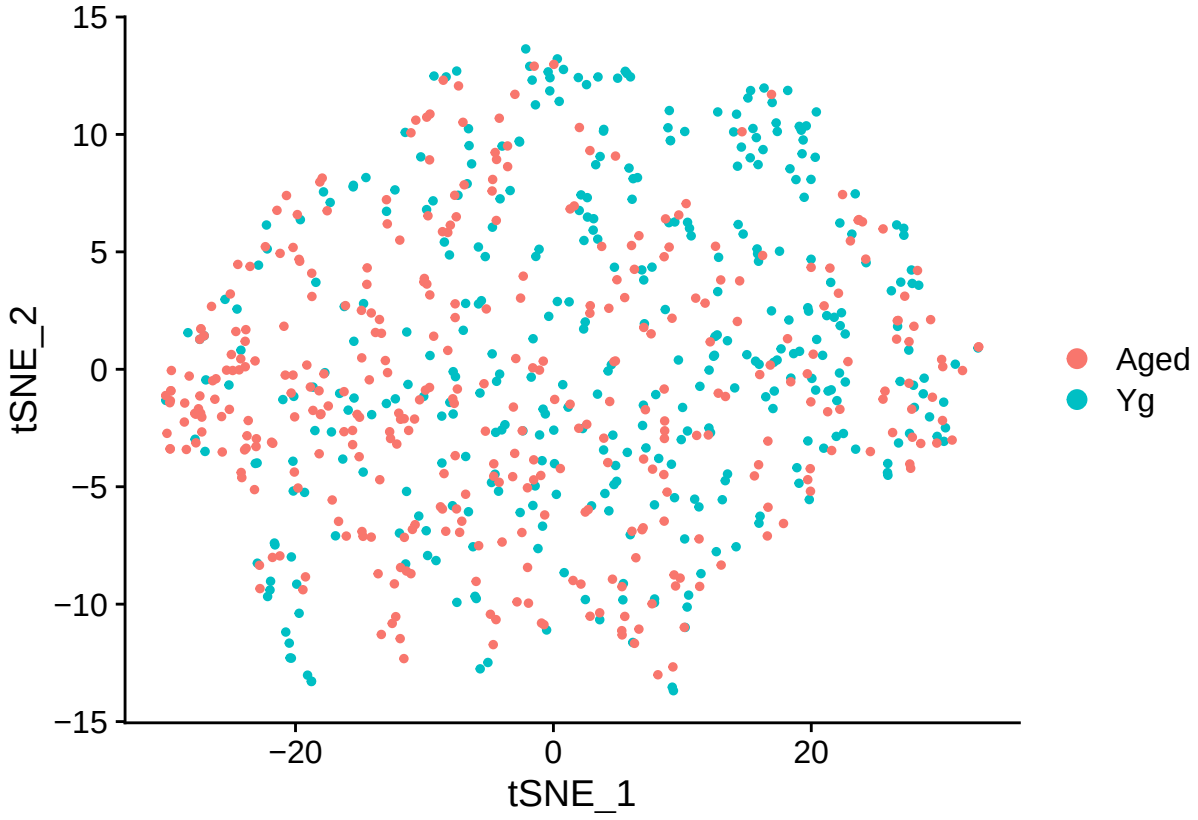
tSNE Plots of Expression Data

```
library("Seurat")

#plot tSNE colored by age of samples
colData <- data.frame(samples = colnames(signifDat),
                      individual = sapply(strsplit(colnames(signifDat),
                                                   split = "_"),
                                          function(str) paste0(str[-1],
                                                               collapse = "")),
                      age = sapply(strsplit(colnames(signifDat),
                                            split = "_"),
                                   function(str) str[2]))
ident <- as.factor(colData$age)
names(ident) <- colData$samples
#set up seurat object and preprocess data with seurat pipeline
seuratData <- CreateSeuratObject(signifDat)
seuratData <- AddMetaData(seuratData,colData)
all.genes <- rownames(seuratData)
seuratData <- ScaleData(seuratData, features = all.genes)
seuratData <- FindVariableFeatures(seuratData,
                                   selection.method = "vst",
                                   nfeatures = 2000)
seuratData <- RunPCA(seuratData)
seuratData <- FindNeighbors(seuratData, dims=1:10)
```

```
seuratData <- FindClusters(seuratData, resolution = 0.5)
```

```
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 729
## Number of edges: 26079
##
## Running Louvain algorithm...
## Maximum modularity in 10 random starts: 0.7109
## Number of communities: 4
## Elapsed time: 0 seconds
```

```
seuratData <- RunTSNE(seuratData, check_duplicates = F)
seuratData@active.ident <- ident
DimPlot(seuratData, reduction = "tsne")
```

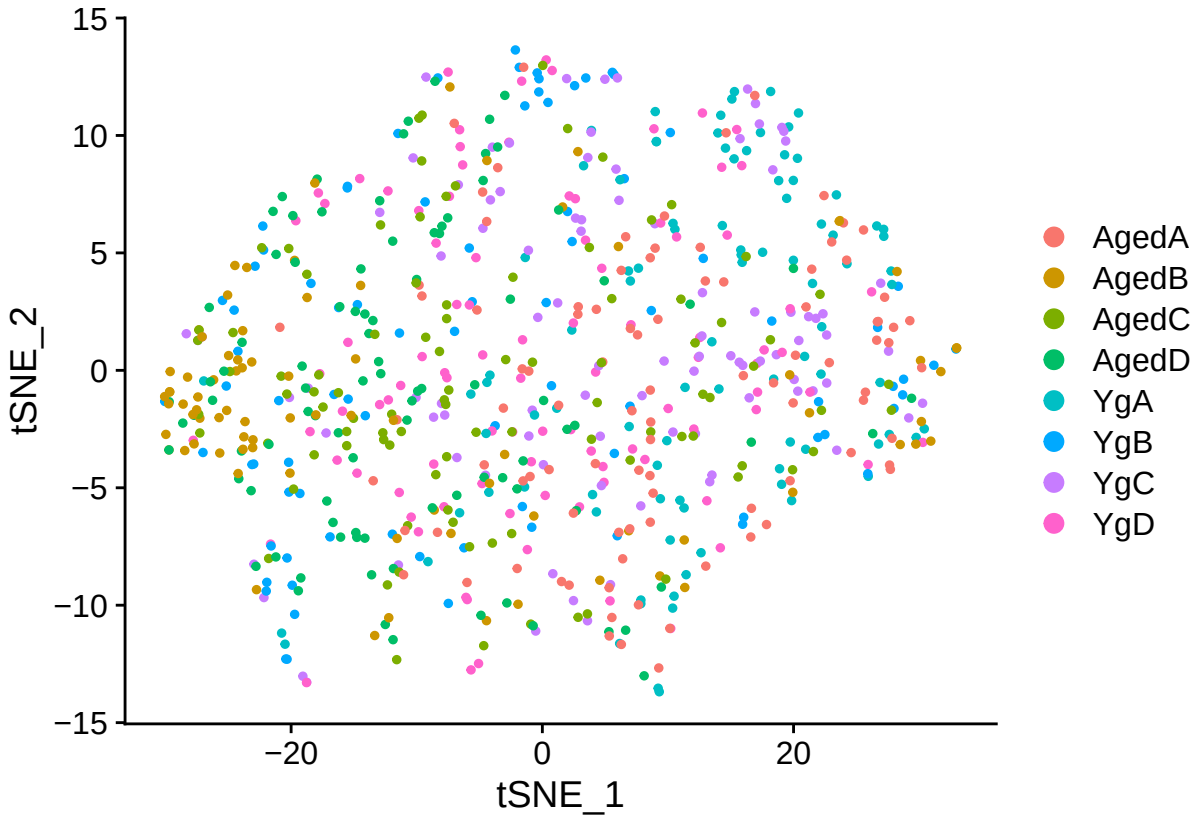



```
#plot tSNE colored by individual of samples
colData <- data.frame(samples = colnames(signifDat),
                      individual = sapply(strsplit(colnames(signifDat),
                                                   split = "_"),
                                          function(str) paste0(str[-1],
                                                               collapse = "")),
                      age = sapply(strsplit(colnames(signifDat),
                                            split = "_"),
                                   function(str) str[2]))
ident <- as.factor(colData$individual)
names(ident) <- colData$samples

seuratData <- CreateSeuratObject(signifDat)
```

```
seuratData <- AddMetaData(seuratData,colData)
all.genes <- rownames(seuratData)
seuratData <- ScaleData(seuratData, features = all.genes)
seuratData <- FindVariableFeatures(seuratData,
                                   selection.method = "vst",
                                   nfeatures = 2000)
seuratData <- RunPCA(seuratData)
seuratData <- FindNeighbors(seuratData, dims=1:10)
seuratData <- FindClusters(seuratData, resolution = 0.5)
```

```
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 729
## Number of edges: 26079
##
## Running Louvain algorithm...
## Maximum modularity in 10 random starts: 0.7109
## Number of communities: 4
## Elapsed time: 0 seconds
```

```
seuratData <- RunTSNE(seuratData, check_duplicates = F)
seuratData@active.ident <- ident
DimPlot(seuratData, reduction = "tsne")
```



Boolean network ensemble reconstruction

To reconstruct Boolean networks from single-cell data, a new approach was developed. In this approach, we are assuming that each single-cell measurement is a potential predecessor or successor of each other time step.

Evaluation of Reconstruction Pipeline with Random Networks

The proposed network reconstruction pipeline was evaluated in terms of speed and reconstruction quality based on randomly generated networks. The pipeline is set up using an algorithm to infer interactions from time-series of data (Maucher et al., 2012) and the best-fit algorithm (Lähdismähki et al., 2008) for reconstruction of Boolean networks. The first step is used as preprocessing to step. The suggested interactions by this algorithm are then used for reconstruction of Boolean networks using the best-fit approach.

Generate random networks and time series

100 random networks for sizes between 20 and 200 are generated. Each is created with scale-free topology and a maximum number of 5 inputs. Time-series are reconstructed on the basis of these networks. For a second experiment, noise is added to these time-series with a 5% chance to flip bits in the time series.

```
#parameter configuration
numSeries <- 20
library("BoolNet")
networkSize <- seq(20,200, by = 20)

#function to remove identical states
cleanTimeSeries <- function(timeSeries)
{
  lapply(timeSeries, function(ts) {
    if(identical(ts[,ncol(ts) - 1], ts[, ncol(ts)]))
      t(unique(t(ts)))
    else
      ts
  })
}

#apply random noise to time series by random bit flips
applyNoise <- function(timeSeries, probNoise = 0.1)
{
  #go over time-series and flip values by chance (according to probability)
  noisyDat <- apply(timeSeries, MARGIN = c(1,2),
                    function(x) if(runif(1,min = 0,max = 1)
                                   <=
                                   probNoise)
                      {1-x}
                    else
                      {x})
  return(noisyDat)
}


#generate random networks using scale free topology for network size between 10 and 100
rndNWs <- lapply(networkSize,
                 function(n) replicate(100,
```

```
                              generateRandomNKNetwork(n,
                              k = 5,
                              topology = "scale_free"),
                            simplify = F))
#generate time-series for corresponding networks
timeSeries <- lapply(rndNWs,
                     function(nw) lapply(nw,
                                 function(s) generateTimeSeries(s,
                                                                numSeries = 1,
                                                                numMeasurements = numSeries)))
timeSeries <- lapply(timeSeries, function(ts) lapply(ts,cleanTimeSeries))
#apply noise with 5% change to flip bits in original time series
timeSeriesNoise <- lapply(timeSeries,
                          function(series) lapply(series,
                                                  function(ts)
                                                    lapply(ts,
                                                           applyNoise,
                                                           probNoise = 0.05)))
#save time series object
save(rndNWs, timeSeries, timeSeriesNoise, file="randomNetworks_timeSeries.RData")
```

Reconstruct networks

Networks are reconstructed from the previously generated time series with and without noise with pure best-fit and filtered best-fit. Reconstruction time is measured for all of these reconstruction runs.

```
library("doParallel")
load("randomNetworks_timeSeries.RData")
numCores <- 100
#reconstruct Boolean networks using best-fit algorithm without preprocessing
bestfitNetworks <- list()
bestfitNetworksNoise <- list()
bestfitNetworksTime <- list()
bestfitNetworksNoiseTime <- list()
#reconstruct Boolean networks using best-fit algorithm
#with preprocessing using Maucher algorithm
inferedNetworks <- list()
inferedNetworksNoise <- list()
inferedNetworksTime <- list()
inferedNetworksNoiseTime <- list()

#predefined threshold for network reconstruction

simResults <- list()
#iterate over network size
for(ts in seq_along(timeSeries))
{
  bestfitNetworks[[ts]] <- list()
  bestfitNetworksNoise[[ts]] <- list()
  bestfitNetworksTime[[ts]] <- list()
  bestfitNetworksNoiseTime[[ts]] <- list()
  #reconstruct Boolean networks using best-fit algorithm
  #with preprocessing using Maucher algorithm
```

```
inferedNetworks[[ts]] <- list()
inferedNetworksNoise[[ts]] <- list()
inferedNetworksTime[[ts]] <- list()
inferedNetworksNoiseTime[[ts]] <- list()
#init cluster
cl <- parallel::makeCluster(numCores)
doParallel::registerDoParallel(cl)

#iterate randomly generated network
result <- foreach(t = seq_along(timeSeries[[ts]]),.packages = 'BoolNet') %dopar%
{
  #predefined threshold for network reconstruction
  thresh <- 0.35

  inferedReconstruction <- function(timeSeries, threshold)
  {
    source("InferViaCorrelation.R")
    #see compilation of c-function in README in C_code folder
    dyn.load("C_code/InferViaCorrelation.so")

    inf_corr <- InferViaCorrelation_transitions(timeSeries,threshold)
    excluded <- lapply(inf_corr[[2]], function(i)
    {       setdiff(1:nrow(timeSeries[[1]]),i[[1]])
    })
    return(reconstructNetwork(timeSeries,
                              method='bestfit',
                              returnPBN=F,
                              excludedDependencies=excluded))
  }

  res <- list()
  #reconstruct using only best-fit algorithm
  res$bestFitTime <- system.time(
    res$bestFitNet <- reconstructNetwork(timeSeries[[ts]][[t]],
                                         method = "bestfit",
                                         maxK = 5))
  res$bestFitTimeNoise <- system.time(
    res$bestFitNetNoise <- reconstructNetwork(timeSeriesNoise[[ts]][[t]],
                                              method = "bestfit",
                                              maxK = 5))
  #reconstruct using filtered best-fit
  res$inferedTime <- system.time(
    res$inferedNet <- inferedReconstruction(timeSeries[[ts]][[t]],
                                            thresh))
  res$inferedTimeNoise <- system.time(
    res$inferedNetNoise <- inferedReconstruction(timeSeries[[ts]][[t]],
                                                 thresh))

  return(res)
}
simResults[[length(simResults) + 1]] <- result

parallel::stopCluster(cl)
```

```
}
save(simResults, timeSeries, file="randomNetworkReconstructionPub.RData")
```

Runtime evaluation

For runtime comparison, the time required for the reconstructed networks (with and without noise), was compared between the original best-fit approach and best-fit approach combined with the algorithm by Maucher et al. for preprocessing (here called infered best-fit).

```
load("randomNetworkReconstructionPub.RData")
load("randomNetworks_timeSeries.RData")
#format stored results from parallel processing
bestfitNetworks <- lapply(simResults,
                          function(replicates)
                            {lapply(replicates,
                                    function(r) r$bestFitNet)})
bestfitNetworksNoise <- lapply(simResults,
                               function(replicates)
                                 {lapply(replicates,
                                         function(r) r$bestFitNetNoise)})
bestfitNetworksTime <- lapply(simResults,
                              function(replicates)
                                {lapply(replicates,
                                        function(r) r$bestFitTime)})
bestfitNetworksNoiseTime <- lapply(simResults,
                                   function(replicates)
                                     {lapply(replicates,
                                             function(r) r$bestFitTimeNoise)})
inferedNetworks <- lapply(simResults,
                          function(replicates)
                            {lapply(replicates,
                                    function(r) r$inferedNet)})
inferedNetworksNoise <- lapply(simResults,
                               function(replicates)
                                 {lapply(replicates,
                                         function(r) r$inferedNetNoise)})
inferedNetworksTime <- lapply(simResults,
                              function(replicates)
                                {lapply(replicates,
                                        function(r) r$inferedTimeNoise)})
inferedNetworksNoiseTime <- lapply(simResults,
                                   function(replicates)
                                     {lapply(replicates,
                                             function(r) r$inferedTimeNoise)})

#measure computation time for best fit algorithm
#extract complete runtime elapsed
meanBestFitTime <- lapply(bestfitNetworksTime,
                          function(s) sapply(s, function(x) x[3]))

meanBestFitTimeNoise <- lapply(bestfitNetworksNoiseTime,
                               function(s) sapply(s, function(x) x[3]))
#measure computation time for infered algorithm
```

```
meanInferedTime <- lapply(inferedNetworksTime,
                          function(s) sapply(s, function(x) x[3]))
meanInferedTimeNoise <- lapply(inferedNetworksNoiseTime,
                               function(s) sapply(s, function(x) x[3]))

names(meanBestFitTime) <-
  names(meanBestFitTimeNoise) <-
  names(meanInferedTime) <-
  names(meanInferedTimeNoise) <- seq(20,200,20)
df <- list(bestfit = meanBestFitTime,
           bestFitNoisy = meanBestFitTimeNoise,
           infered = meanInferedTime,
           inferedNoisy = meanInferedTimeNoise)

library(reshape2)
```

```
##
## Attaching package: 'reshape2'
```

```
## The following object is masked from 'package:tidyr':
##
##     smiths
```

```
library(ggplot2)
library(scales)
```

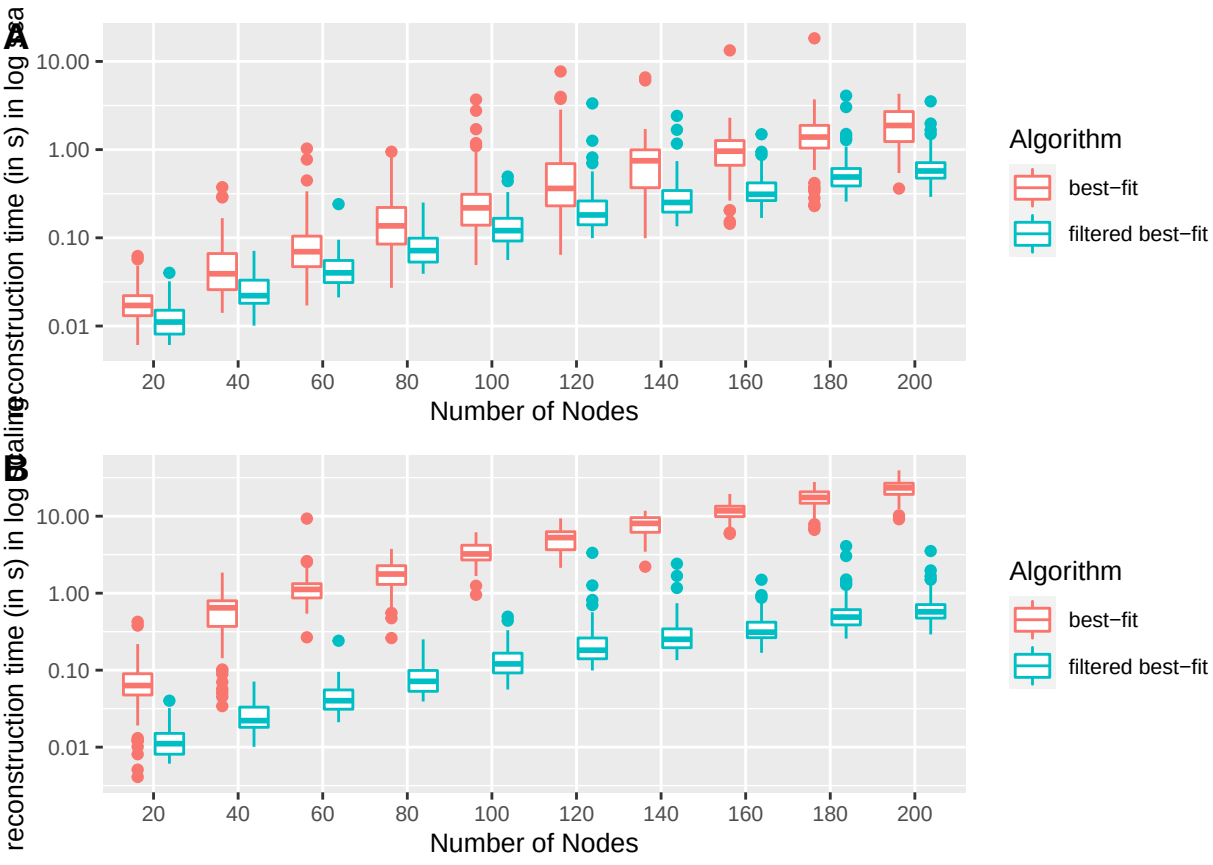
```
##
## Attaching package: 'scales'
```

```
## The following object is masked from 'package:purrr':
##
##     discard
```

```
## The following object is masked from 'package:readr':
##
##     col_factor
```

```
library(cowplot)
df <- melt(df)
names(df) <- c("time", "size", "type")
df$time <- df$time + 0.0001
df$size <- as.numeric(df$size)
df$size <- as.factor(df$size)
networkSize <- seq(20,200, by = 20)

point <- format_format(big.mark = " ", decimal.mark = ".", scientific = FALSE)

par(xpd = T)
par(mar = c(4,8,2,2),omi=c(4,0,0,0))
theme_set(theme_grey())
p1 <- ggplot(data=df[!grepl(pattern = "Noisy", df$type),],
             aes(x=size, y=time, col = type, na.rm = T)) +
  geom_boxplot() +
  scale_x_discrete(name = "Number of Nodes",
                   breaks = networkSize,
                   labels = seq(20,200, by = 20)) +
```

```
  scale_y_log10(name = "reconstruction time (in s) in log scaling",
                labels = point) +
  theme(text = element_text(size=10)) +
  scale_color_discrete(name = "Algorithm",
                       labels = c("best-fit", "filtered best-fit"))

p2 <- ggplot(data=df[grepl(pattern = "Noisy", df$type),],
             aes(x=size, y=time, col = type, na.rm = T)) +
  geom_boxplot() +
  scale_x_discrete(name = "Number of Nodes",
                   breaks = networkSize,
                   labels = seq(20,200, by = 20)) +
  scale_y_log10(name = "reconstruction time (in s) in log scaling",
                labels = point) +
  theme(text = element_text(size=10)) +
  scale_color_discrete(name = "Algorithm",
                       labels = c("best-fit", "filtered best-fit"))

plot_grid(p1, p2, ncol=1, labels = c("A", "B"), label_size=14, align = "v")
```



Reconstruction quality

Next, we compared the reconstruction quality between the original best-fit algorithm and the filtered best-fit approach. Based on the randomly generated Boolean networks as ground-truth models, we compared the interactions found by the reconstruction algorithms as measured specificity and sensitivity of these predicted

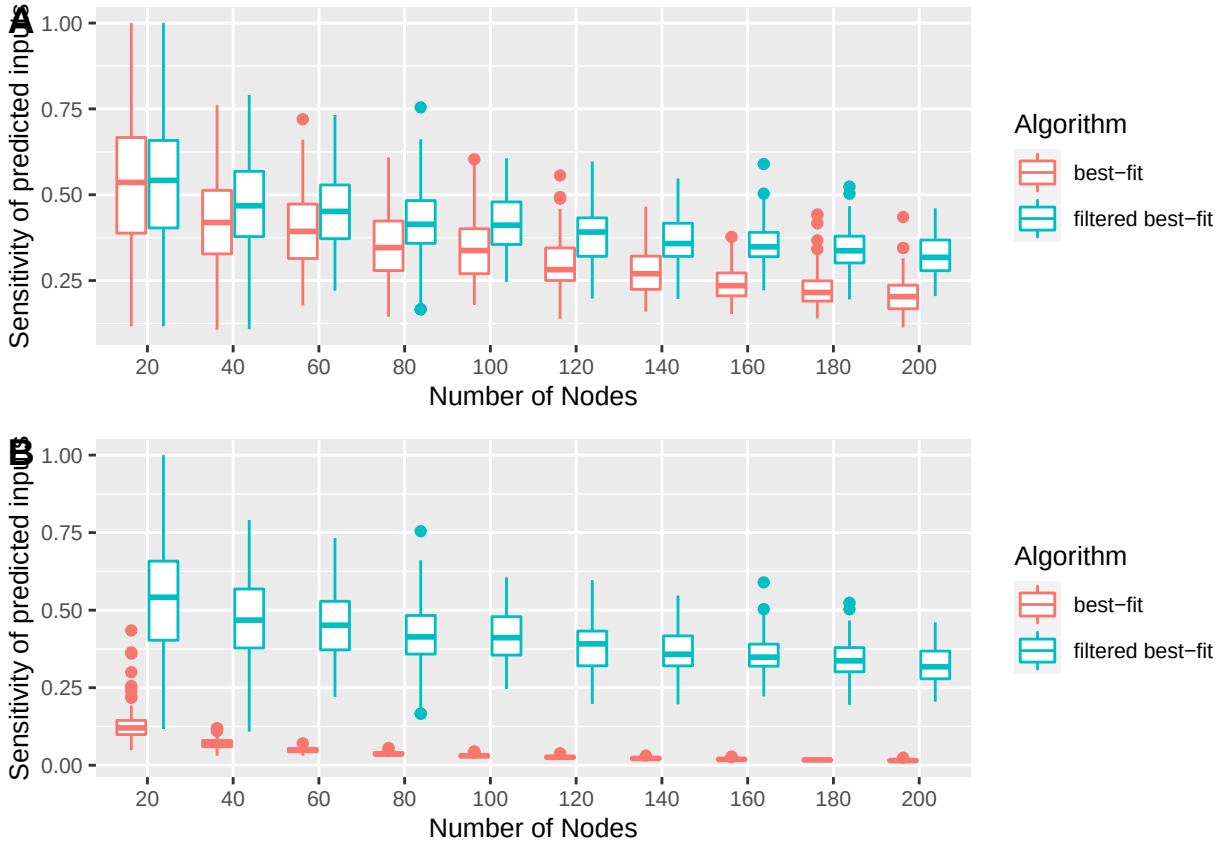
interactions.

```
#adjacency matrix from BoolNet file network
transformBoolNetSetToAdjacency <- function(net)
{
  adjacency <- matrix(0, ncol = length(net$genes), nrow = length(net$genes))
  dump <- sapply(1:length(net$genes), function(id) {
    adjacency[id, unique(unlist(lapply(net$interactions[[id]],
                                       function(x) x$input)))] <<- 1
  })

  return(adjacency)
}

#adjacency matrix from reconset of functions
transformBoolNetToAdjacency <- function(net)
{
  adjacency <- matrix(0, ncol = length(net$genes), nrow = length(net$genes))
  dump <- sapply(1:length(net$genes), function(id) {
    adjacency[id, net$interactions[[id]]$input] <<- 1
  })

  return(adjacency)
}

#compare adjacency matrices and return evaluation of classification
compareAdjacencyMatrices <- function(origMatrix, reconMatrix)
{
  if(!all(dim(origMatrix) == dim(reconMatrix)))
  {
    warning("Adjacency matrices are not of same size")
    return()
  }
  else{
    equalMat <- origMatrix== reconMatrix

    tp <- sum(equalMat & (origMatrix == 1))
    fp <- sum(!equalMat & (origMatrix == 1))
    tn <- sum(equalMat & (origMatrix == 0))
    fn <- sum(!equalMat & (origMatrix == 0))
  }
  return(list(TP = tp, FP = fp, TN = tn, FN = fn))
}

#functions for different measures based on the calculated confusion matrix

calculateAccuracy <- function(binaryEval)
{
  return((binaryEval$TP + binaryEval$TN)/(sum(unlist(binaryEval))))
}

calculateSensitivity <- function(binaryEval)
{
  return(binaryEval$TP/(binaryEval$TP + binaryEval$FN))
```

```
}

calculateSpecificity <- function(binaryEval)
{
  return(binaryEval$TN/(binaryEval$TN + binaryEval$FP))
}


#calculate sensitivity for reconstruction using best-fit
sensBestFit <- mapply(function(origSet, recSet) {
  mapply(function(orig, recon) {
    recMat <- transformBoolNetSetToAdjacency(recon)
    origMat <- transformBoolNetToAdjacency(orig)

    binEval <- compareAdjacencyMatrices(origMatrix = origMat,
                                        reconMatrix = recMat)

    return(calculateSensitivity(binEval))
  }, origSet, recSet)},rndNWs,bestfitNetworks, SIMPLIFY = F)

#calculate sensitivity for reconstruction using best-fit with preprocessing
sensInferred <- mapply(function(origSet, recSet) {
  mapply(function(orig, recon) {
    recMat <- transformBoolNetSetToAdjacency(recon)
    origMat <- transformBoolNetToAdjacency(orig)

    binEval <- compareAdjacencyMatrices(origMatrix = origMat,
                                        reconMatrix = recMat)

    return(calculateSensitivity(binEval))
  }, origSet, recSet)},rndNWs,inferedNetworks, SIMPLIFY = F)

#calculate sensitivity for reconstruction using best-fit with 5% noise
sensBestFitNoise <- mapply(function(origSet, recSet) {
  mapply(function(orig, recon) {
    recMat <- transformBoolNetSetToAdjacency(recon)
    origMat <- transformBoolNetToAdjacency(orig)

    binEval <- compareAdjacencyMatrices(origMatrix = origMat,
                                        reconMatrix = recMat)

    return(calculateSensitivity(binEval))
  }, origSet, recSet)},rndNWs,bestfitNetworksNoise, SIMPLIFY = F)

#calculate sensitivity for reconstruction using best-fit with preprocessing with 5% noise
sensinferredNoise <- mapply(function(origSet, recSet) {
  mapply(function(orig, recon) {
    recMat <- transformBoolNetSetToAdjacency(recon)
    origMat <- transformBoolNetToAdjacency(orig)

    binEval <- compareAdjacencyMatrices(origMatrix = origMat,
                                        reconMatrix = recMat)
```

```
    return(calculateSensitivity(binEval))
  }, origSet, recSet)},rndNWs,inferedNetworksNoise, SIMPLIFY = F)

#collect data and extract min, mean, max sensitivity for each approach
sensitivity <- t(rbind(bfNoise = sapply(sensBestFitNoise,
                                        function(x) c(min(x),mean(x),max(x))),
                       filteredNoise = sapply(sensinferredNoise,
                                              function(x) c(min(x),mean(x),max(x))),
                       bf = sapply(sensBestFit,
                                   function(x) c(min(x),mean(x),max(x))),
                       filtered = sapply(sensInferred,
                                         function(x) c(min(x),mean(x),max(x)))))

rownames(sensitivity) <- seq(20,200,by=20)

#print plots
df <- list(bfNoise = sensBestFitNoise,
                     filteredNoise = sensinferredNoise,
                     bf = sensBestFit,
                     filtered = sensInferred)
library(reshape2)
library(ggplot2)
library(scales)
library(cowplot)
df <- melt(df)
names(df) <- c("sensitivity", "size", "type")
df$size <- as.numeric(df$size) * 20
df$size <- as.factor(df$size)
networkSize <- seq(20,200, by = 20)
point <- format_format(big.mark = " ", decimal.mark = ".", scientific = FALSE)

par(xpd = T)
par(mar = c(4,8,2,2),omi=c(4,0,0,0))
theme_set(theme_grey())
#plot non-noisy data
p1 <- ggplot(data=df[!grepl(pattern = "Noise", df$type),],
             aes(x=size, y=sensitivity, col = type, na.rm = T)) +
  geom_boxplot() +
  scale_x_discrete(name = "Number of Nodes",
                   breaks = networkSize,
                   labels = seq(20,200, by = 20)) +
  scale_y_continuous(name = "Sensitivity of predicted inputs",
                     labels = point) +
  theme(text = element_text(size=10)) +
  scale_color_discrete(name = "Algorithm",
                       labels = c("best-fit", "filtered best-fit"))
#plot noisy data
p2 <- ggplot(data=df[grepl(pattern = "Noise", df$type),],
             aes(x=size, y=sensitivity, col = type, na.rm = T)) +
  geom_boxplot() +
  scale_x_discrete(name = "Number of Nodes",
                   breaks = networkSize,
                   labels = seq(20,200, by = 20)) +
```

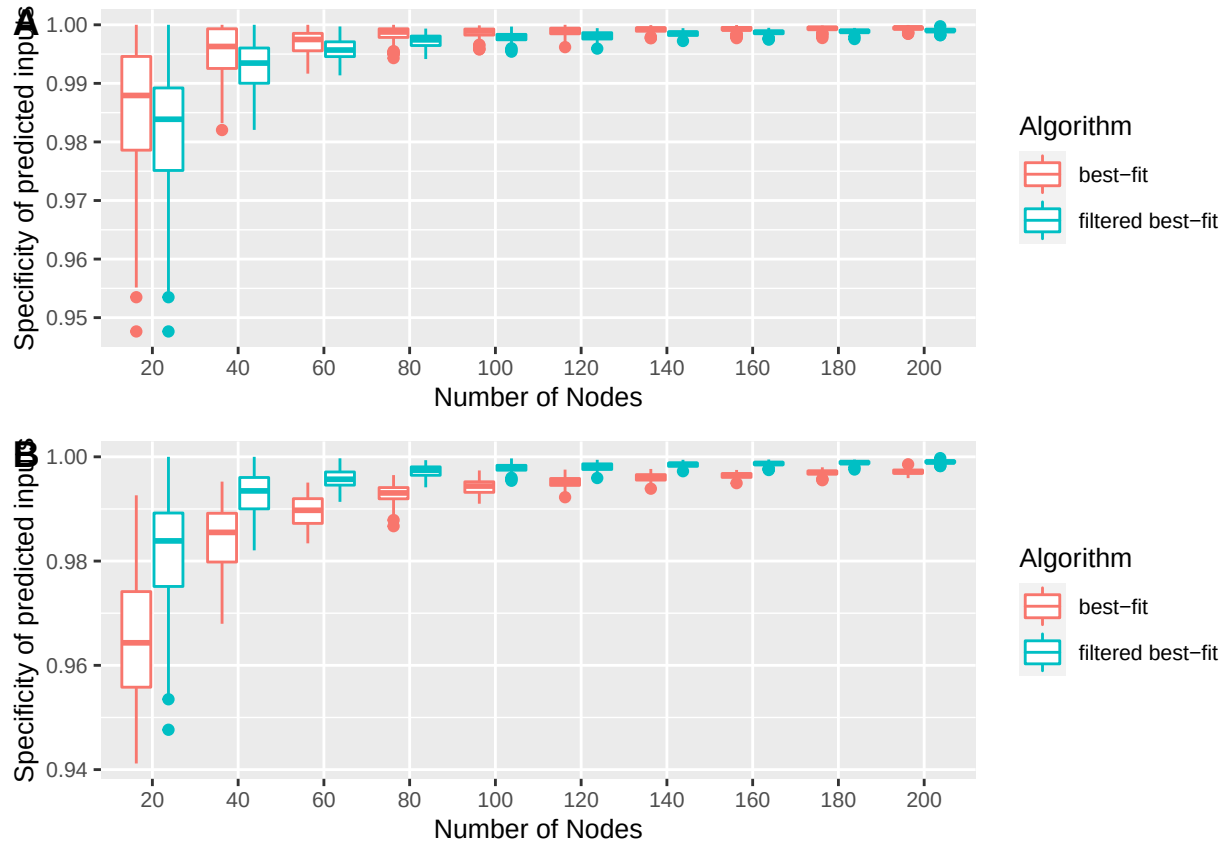
```
  scale_y_continuous(name = "Sensitivity of predicted inputs",
                     labels = point) +
  theme(text = element_text(size=10)) +
  scale_color_discrete(name = "Algorithm",
                       labels = c("best-fit", "filtered best-fit"))
#wrap both plots together
plot_grid(p1, p2, ncol=1, labels = c("A", "B"), label_size=14, align = "v")
```




```
#calculate specificity of the four different runs
specBestFit <- mapply(function(origSet, recSet) {
  mapply(function(orig, recon) {
    recMat <- transformBoolNetSetToAdjacency(recon)
    origMat <- transformBoolNetToAdjacency(orig)

    binEval <- compareAdjacencyMatrices(origMatrix = origMat, reconMatrix = recMat)

    return(calculateSpecificity(binEval))
  }, origSet, recSet)},rndNWs,bestfitNetworks, SIMPLIFY = F)

specInferred <- mapply(function(origSet, recSet) {
  mapply(function(orig, recon) {
    recMat <- transformBoolNetSetToAdjacency(recon)
    origMat <- transformBoolNetToAdjacency(orig)

    binEval <- compareAdjacencyMatrices(origMatrix = origMat, reconMatrix = recMat)

    return(calculateSpecificity(binEval))
```

```
  }, origSet, recSet)},rndNWs,inferedNetworks, SIMPLIFY = F)

specBestFitNoise <- mapply(function(origSet, recSet) {
  mapply(function(orig, recon) {
    recMat <- transformBoolNetSetToAdjacency(recon)
    origMat <- transformBoolNetToAdjacency(orig)

    binEval <- compareAdjacencyMatrices(origMatrix = origMat, reconMatrix = recMat)

    return(calculateSpecificity(binEval))
  }, origSet, recSet)},rndNWs,bestfitNetworksNoise, SIMPLIFY = F)

specinferredNoise <- mapply(function(origSet, recSet) {
  mapply(function(orig, recon) {
    recMat <- transformBoolNetSetToAdjacency(recon)
    origMat <- transformBoolNetToAdjacency(orig)

    binEval <- compareAdjacencyMatrices(origMatrix = origMat, reconMatrix = recMat)

    return(calculateSpecificity(binEval))
  }, origSet, recSet)},rndNWs,inferedNetworksNoise, SIMPLIFY = F)

specificity <- t(rbind(bfNoise = sapply(specBestFitNoise,
                                        function(x)c(min(x),mean(x),max(x))),
                       filteredNoise = sapply(specinferredNoise,
                                              function(x)c(min(x),mean(x),max(x))),
                       bf = sapply(specBestFit,
                                   function(x)c(min(x),mean(x),max(x))),
                       filtered = sapply(specInferred,
                                         function(x)c(min(x),mean(x),max(x)))))

rownames(specificity) <- seq(20,200,by=20)


df <- list(bfNoise = specBestFitNoise,
                    filteredNoise = specinferredNoise,
                    bf = specBestFit,
                    filtered = specInferred)
library(reshape2)
library(ggplot2)
library(scales)
library(cowplot)
df <- melt(df)
names(df) <- c("specificity", "size", "type")
df$size <- as.numeric(df$size) * 20
df$size <- as.factor(df$size)
networkSize <- seq(20,200, by = 20)
point <- format_format(big.mark = " ", decimal.mark = ".", scientific = FALSE)

par(xpd = T)
par(mar = c(4,8,2,2),omi=c(4,0,0,0))
theme_set(theme_grey())
#plot specificity of non-noisy data
```

```
p1 <- ggplot(data=df[!grepl(pattern = "Noise", df$type),],
             aes(x=size, y=specificity, col = type, na.rm = T)) +
  geom_boxplot() +
  scale_x_discrete(name = "Number of Nodes",
                   breaks = networkSize,
                   labels = seq(20,200, by = 20)) +
  scale_y_continuous(name = "Specificity of predicted inputs",
                     labels = point) +
  theme(text = element_text(size=10)) +
  scale_color_discrete(name = "Algorithm",
                       labels = c("best-fit", "filtered best-fit"))
#plot specificity of noisy data
p2 <- ggplot(data=df[grepl(pattern = "Noise", df$type),],
             aes(x=size, y=specificity, col = type, na.rm = T)) +
  geom_boxplot() +
  scale_x_discrete(name = "Number of Nodes",
                   breaks = networkSize,
                   labels = seq(20,200, by = 20)) +
  scale_y_continuous(name = "Specificity of predicted inputs",
                     labels = point) +
  theme(text = element_text(size=10)) +
  scale_color_discrete(name = "Algorithm",
                       labels = c("best-fit", "filtered best-fit"))

#wrap both plots together
plot_grid(p1, p2, ncol=1, labels = c("A", "B"), label_size=14, align = "v")
```



Generate random networks with long time series

Analysis of Boolean networks reconstruction using "long" time-series. In the following, we repeated the previous analysis with 10 networks with 20 to 200 nodes, each of the were generated randomly. Each network was created using scale-free topology to mimic biological behavior. For each network a time-series of (n + 10) time points was generated (according to the evaluation in Berestovski et al., PLOS ONE, 2013). A copy of each time-series was generated applying noise by adding a 5% chance to flip a bit. Each time-series was reconstructed to the respective Boolean network using the best-fit algorithm and the best-fit algorithm with filtered input. Results by both algorithms were compared for original and noisy time-series respectively.

```
library("BoolNet")
networkSize <- seq(20,200, by = 20)
numSeries <- networkSize + 10
#generate random networks using scale free topology for network size between 10 and 100
rndNWs <- lapply(networkSize,
                 function(n) replicate(10,
                                       generateRandomNKNetwork(n,
                                                               k = 5,
                                                               topology = "scale_free"),
                                       simplify = F))

#generate time-series for corresponding networks
timeSeries <- mapply(function(nw,noTs) lapply(nw,
                                              function(s) generateTimeSeries(s,
                                                                             numSeries = noTs,
                                                                             numMeasurements = 2)),
                     rndNWs,
                     numSeries,
                     SIMPLIFY = F)

timeSeries <- lapply(timeSeries, function(ts) lapply(ts,cleanTimeSeries))
#apply noise with 5% change to flip bits in original time series
timeSeriesNoise <- lapply(timeSeries, function(series)
  lapply(series, function(ts)
    lapply(ts, applyNoise, probNoise = 0.05)))
```

Reconstruct networks

Networks are reconstructed from the previously generated time series with and without noise with original best-fit and filtered best-fit. Reconstruction time is measured for all of these networks.

```
#reconstruct Boolean networks using best-fit algorithm with/without preprocessing
#predefined threshold for network reconstruction
thresh <- 0.35
library("doParallel")
load("rndNetworks_timeSeries_Long.RData")
no_cores <- 5
inferedRecon <- list()
#run reconstruction in parallel for best-fit with preprocessing on non-noisy data
for(ts in seq_along(timeSeries))
{
  cl <- makeCluster(no_cores)
  registerDoParallel(cl)
  inferedRecon[[ts]] <- foreach(t=seq_along(timeSeries[[ts]]),.packages = "BoolNet")
```

```
  %dopar%
  {
    #insert required function in parallel sessions
    inferedReconstruction <- function(timeSeries, threshold)
    {
      source("InferViaCorrelation.R")
      #see compilation of c-function in README in C_code folder
      dyn.load("C_code/InferViaCorrelation.so")

      inf_corr <- InferViaCorrelation_transitions(timeSeries,threshold)
      excluded <- lapply(inf_corr[[2]], function(i)
      {       setdiff(1:nrow(timeSeries[[1]]),i[[1]])
      })
      return(reconstructNetwork(timeSeries,
                                method='bestfit',
                                returnPBN=F,
                                excludedDependencies=excluded))
    }
    runtime <- system.time(inferedNet <- inferedReconstruction(timeSeries[[ts]][[t]],
                                                               threshold = thresh))

    return(list(time = runtime, network = inferedNet))
  }
  stopCluster(cl)
}
#save results
save(inferedRecon, file = "inferedRndLong.RData")
rm(inferedRecon)
#run reconstruction in parallel for best-fit on non-noisy data
bestfitRecon <- list()
#run reconstruction
for(ts in seq_along(timeSeries))
{
  cl <- makeCluster(no_cores)
  registerDoParallel(cl)
  bestfitRecon[[ts]] <- foreach(t=seq_along(timeSeries[[ts]]),.packages = "BoolNet")
  %dopar%
  {
    runtime <- system.time(bfNet <- reconstructNetwork(timeSeries[[ts]][[t]],
                                                       method = "bestfit"))

    return(list(time = runtime, network = bfNet))
  }
  stopCluster(cl)
}
#save data
save(bestfitRecon, file = "bestfitRndLong.RData")
rm(bestfitRecon)

#run reconstruction in parallel for best-fit with preprocessing on noisy data
inferedReconNoise <- list()

for(ts in seq_along(timeSeries))
```

```
{
  cl <- makeCluster(no_cores)
  registerDoParallel(cl)
  inferedReconNoise[[ts]] <- foreach(t=seq_along(timeSeriesNoise[[ts]]),.packages = "BoolNet")
  %dopar%
  {
    #insert required function in parallel sessions
    inferedReconstruction <- function(timeSeries, threshold)
    {
      source("InferViaCorrelation.R")
      #see compilation of c-function in README in C_code folder
      dyn.load("C_code/InferViaCorrelation.so")

      inf_corr <- InferViaCorrelation_transitions(timeSeries,threshold)
      excluded <- lapply(inf_corr[[2]], function(i)
      {       setdiff(1:nrow(timeSeries[[1]]),i[[1]])
      })
      return(reconstructNetwork(timeSeries,
                                method='bestfit',
                                returnPBN=F,
                                excludedDependencies=excluded))
    }
    runtime <- system.time(inferedNet <- inferedReconstruction(timeSeriesNoise[[ts]][[t]],
                                                               threshold = thresh))

    return(list(time = runtime, network = inferedNet))
  }
  stopCluster(cl)
}
#save data
save(inferedReconNoise, file = "inferedNoisRndLong.RData")
rm(inferedReconNoise)

#run reconstruction in parallel for best-fit on noisy data
bestfitReconNoise <- list()

for(ts in seq_along(timeSeriesNoise))
{
  cl <- makeCluster(no_cores)
  registerDoParallel(cl)
  bestfitReconNoise[[ts]] <- foreach(t=seq_along(timeSeries[[ts]]),.packages = "BoolNet")
  %dopar%
  {
    runtime <- system.time(bfNet <- reconstructNetwork(timeSeriesNoise[[ts]][[t]],
                                                       method = "bestfit"))

    return(list(time = runtime, network = bfNet))
  }
  stopCluster(cl)
}
#save data
save(bestfitRecon, file = "bestfitNoiseRndLong.RData")
```

Runtime Evaluation

First, runtime of each reconstruction process was measured. The following tables/plots show the factor of the mean speed increase using the filtered best-fit approach compared to the original best-fit algorithm. Reconstruction time is measured in seconds.

```
#static analysis of reconstructed networks
library("reshape2")
library("dplyr")
library("ggplot2")
library("lubridate")
```

```
##
## Attaching package: 'lubridate'
```

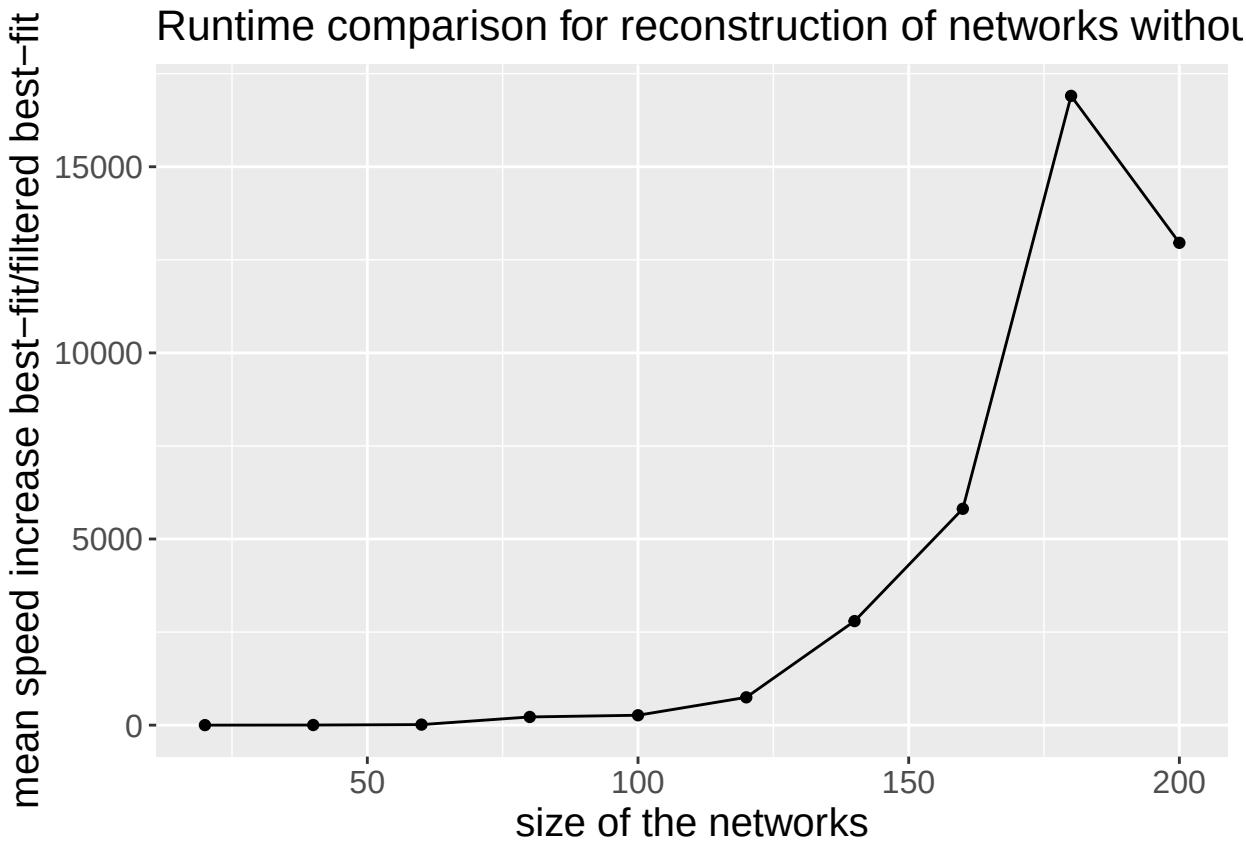
```
## The following object is masked from 'package:cowplot':
##
##     stamp
```

```
## The following objects are masked from 'package:BiocGenerics':
##
##     intersect, setdiff, union
```

```
## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

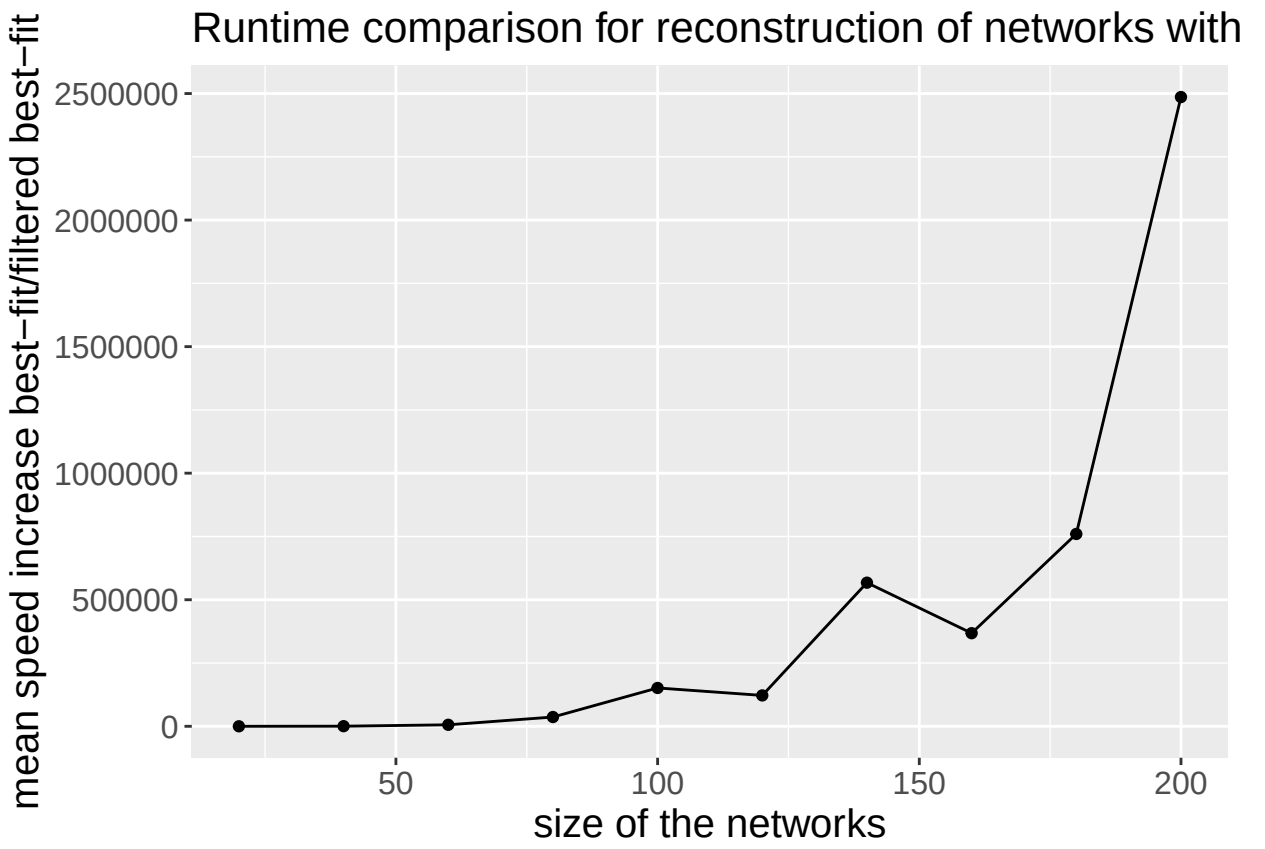
```
#load variables
load("rndNetworks_timeSeries_Long.RData")
load("inferedRndLong.RData")
load("bestfitRndLong.RData")
load("inferedNoiseRndLong.RData")
load("bestfitNoiseRndLong.RData")

#comparison of reconstruction time
bfTime <- lapply(bestfitRecon,
                 function(size)
                   sapply(size, function(n) n[[1]][[3]]))
infTime <- lapply(inferedRecon,
                  function(size)
                    sapply(size, function(n) n[[1]][[3]]))
#compute increase of speed
speedFactor <- mapply(function(x,y) x/y, bfTime, infTime ,SIMPLIFY = F)
meanSpeedFactor <- sapply(speedFactor, mean)
names(speedFactor) <- seq(20,200,by=20)
df <- melt(speedFactor)
colnames(df) <- c("runtime", "size")
df <- df %>% group_by(size) %>% summarise(meanFactor = mean(runtime))
df$size <- as.numeric(df$size)
par(xpd = T)
par(mar = c(0,8,2,2),omi=c(0,0,0,0))
ggplot(df, aes(x = size, y = meanFactor, group = 1)) +
  geom_line() +
  geom_point() +
  labs(x = "size of the networks",
       y = "mean speed increase best-fit/filtered best-fit",
       title = "Runtime comparison for reconstruction of networks without noise") +
```

```
  theme(text = element_text(size=15), plot.title= element_text(size=16))
```



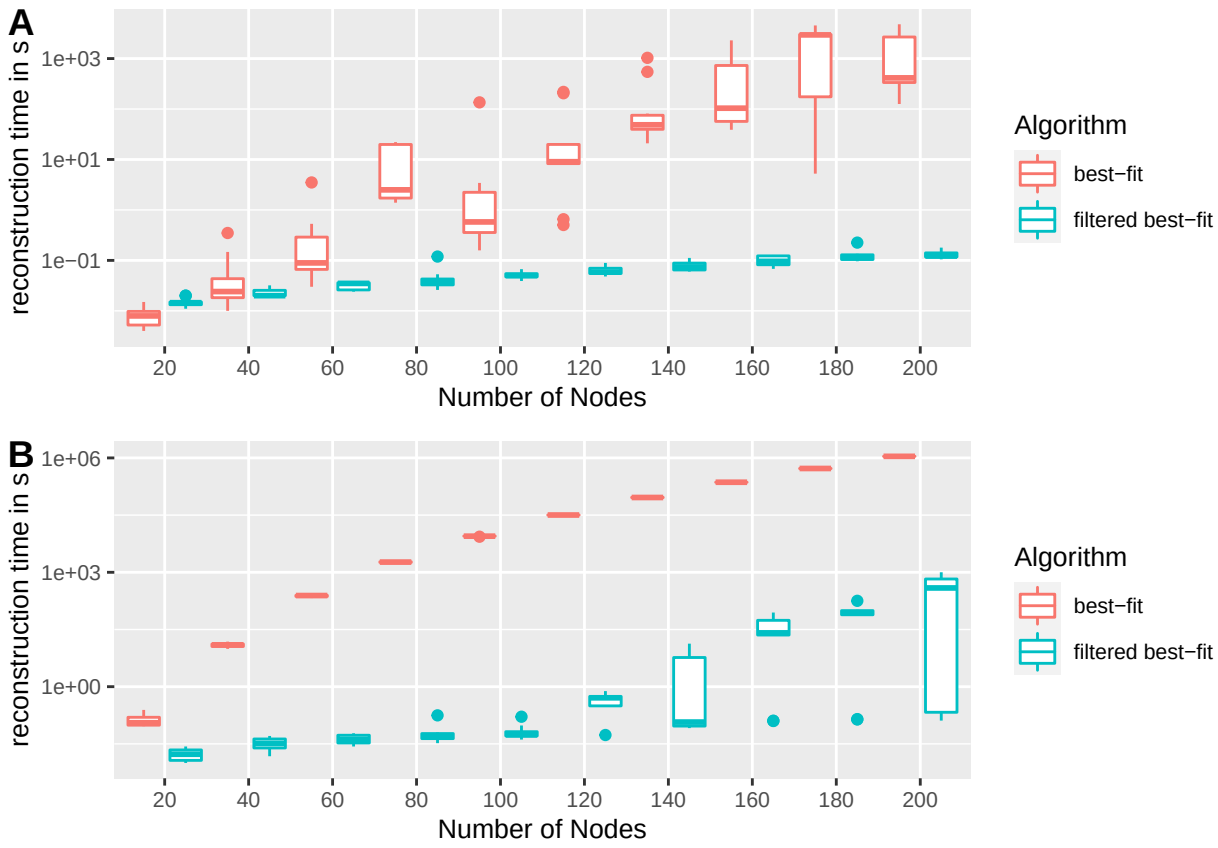

```
overview <- data.frame(filtered = sapply(infTime,
                                         function(x) as.character(duration(s= round(mean(x),
                                                                                    digits = 2)))),
                       bf = sapply(bfTime,
                                   function(x) as.character(duration(s= round(mean(x),
                                                                              digits = 2)))),
                      factor = sapply(meanSpeedFactor, mean))
colnames(overview) <- c("filtered best-fit", "best-fit", "factor")
rownames(overview) <- seq(20,200,by=20)

bfTimeNoise <- lapply(bestfitReconNoise,
                      function(size) sapply(size, function(n) n[[1]][[3]]))
infTimeNoise <- lapply(inferedReconNoise,
                       function(size) sapply(size, function(n) n[[1]][[3]]))

speedFactorNoise <- mapply(function(x,y) x/y, bfTimeNoise,
                           infTimeNoise,
                           SIMPLIFY = F)
names(speedFactorNoise) <- seq(20,200,by=20)
meanSpeedFactorNoise <- sapply(speedFactorNoise, mean)
df <- melt(speedFactorNoise)
colnames(df) <- c("runtime", "size")
df <- df %>% group_by(size) %>% summarise(meanFactor = mean(runtime))
df$size <- as.numeric(df$size)
par(xpd = T)
```

```
par(mar = c(0,8,2,2),omi=c(0,0,0,0))
ggplot(df, aes(x = size, y = meanFactor, group = 1)) +
  geom_line() +
  geom_point() +
  labs(x = "size of the networks",
       y = "mean speed increase best-fit/filtered best-fit",
       title = "Runtime comparison for reconstruction of networks without noise") +
  theme(text = element_text(size=15), plot.title= element_text(size=16))
```




```
overview <- data.frame(filtered = sapply(infTimeNoise,
                                         function(x) as.character(duration(s = round(mean(x),
                                                                                     digits = 2)))),
                       bf = sapply(bfTimeNoise,
                                   function(x) as.character(duration(s = round(mean(x),
                                                                               digits = 2)))),
                      factor = sapply(meanSpeedFactorNoise,
                                      mean))

colnames(overview) <- c("filtered best-fit",
                        "best-fit",
                        "factor")

rownames(overview) <- seq(20,200,by=20)

#boxplots for runtime
timing <- list(bestfit = bfTime,
               filtered = infTime,
```

```
                bestfitNoise = bfTimeNoise,
                filteredNoise = infTimeNoise)
df <- melt(timing)
colnames(df) <- c("time", "size", "type")
df$size <- as.numeric(df$size) * 20
df$size <- as.factor(df$size)

library(scales)
library(cowplot)

par(xpd = T)
par(mar = c(4,8,2,2),omi=c(4,0,0,0))
theme_set(theme_grey())
p1 <- ggplot(data = df[!grepl(df$type,pattern = "Noise"), ],
             aes(x = size, y = time, col = type)) +
  geom_boxplot(position=position_dodge(1)) +
  scale_x_discrete(name = "Number of Nodes",
                   breaks = networkSize,
                   labels = seq(20,200, by = 20)) +
  scale_y_log10(name = "reconstruction time in s") +
  theme(text = element_text(size=10)) +
  scale_color_discrete(name = "Algorithm",
                       labels = c("best-fit", "filtered best-fit"))


p2 <-ggplot(data = df[grepl(df$type,pattern = "Noise"), ],
            aes(x = size, y = time, col = type)) +
  geom_boxplot(position=position_dodge(1)) +
  scale_x_discrete(name = "Number of Nodes",
                   breaks = networkSize,
                   labels = seq(20,200, by = 20)) +
  scale_y_log10(name = "reconstruction time in s") +
  theme(text = element_text(size=10)) +
  scale_color_discrete(name = "Algorithm",
                       labels = c("best-fit", "filtered best-fit"))

plot_grid(p1, p2, ncol=1, labels = c("A", "B"), label_size=14, align = "v")
```



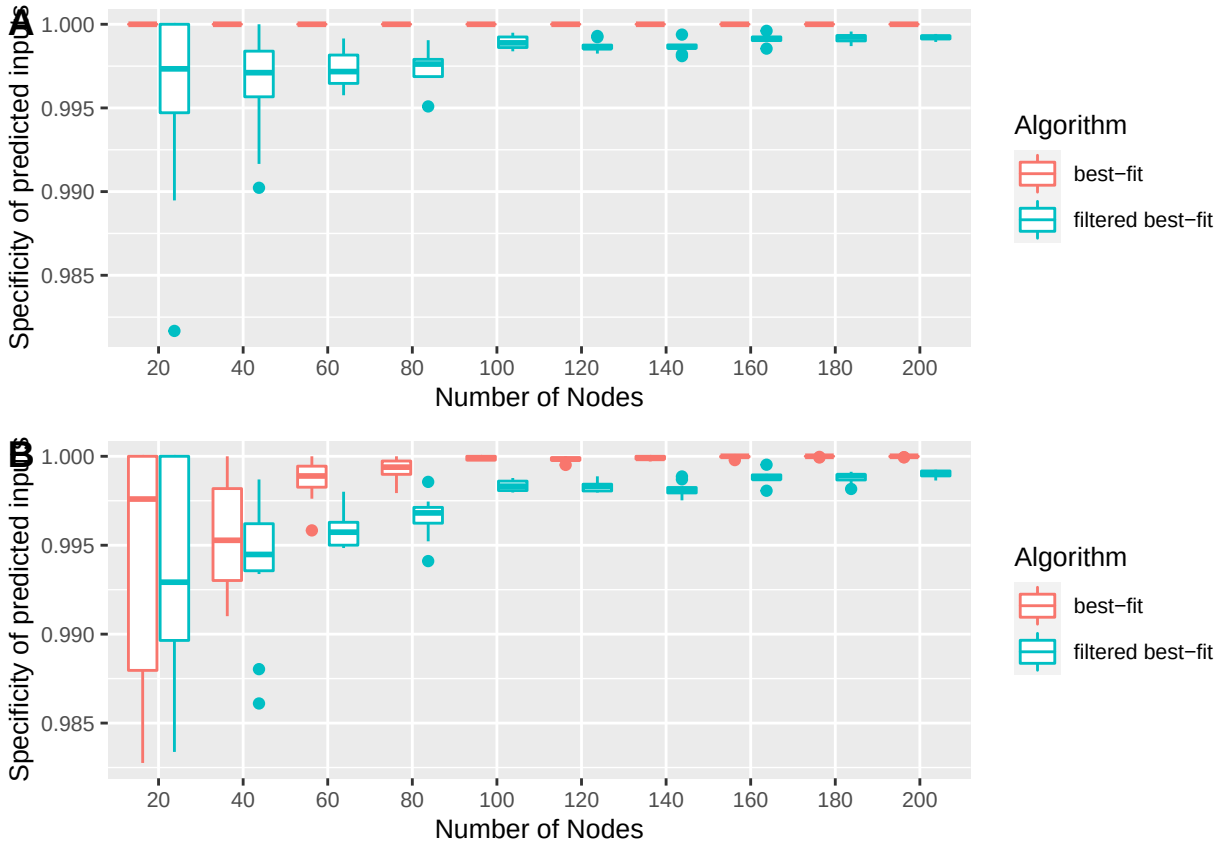
Reconstruction quality

```
bestfitNetworks <- lapply(bestfitRecon, function(set)
  lapply(set, function(n) n$network))
inferredNetworks <- lapply(inferedRecon, function(set)
  lapply(set, function(n) n$network))
sensBestFit <- mapply(function(origSet, recSet) {
  mapply(function(orig, recon) {
    recMat <- transformBoolNetSetToAdjacency(recon)
    origMat <- transformBoolNetToAdjacency(orig)

    binEval <- compareAdjacencyMatrices(origMatrix = origMat,
                                        reconMatrix = recMat)

    return(calculateSensitivity(binEval))
  }, origSet, recSet)},rndNWs,bestfitNetworks, SIMPLIFY = F)

sensInferred <- mapply(function(origSet, recSet) {
  mapply(function(orig, recon) {
    recMat <- transformBoolNetSetToAdjacency(recon)
    origMat <- transformBoolNetToAdjacency(orig)

    binEval <- compareAdjacencyMatrices(origMatrix = origMat,
                                        reconMatrix = recMat)
```

```
    return(calculateSensitivity(binEval))
  }, origSet, recSet)},rndNWs,inferredNetworks, SIMPLIFY = F)


bestfitNetworksNoise <- lapply(bestfitReconNoise,
                               function(set) lapply(set,
                                                    function(n) n$network))
inferredNetworksNoise <- lapply(inferedReconNoise,
                                function(set) lapply(set,
                                                     function(n) n$network))

sensBestFitNoise <- mapply(function(origSet, recSet) {
  mapply(function(orig, recon) {
    recMat <- transformBoolNetSetToAdjacency(recon)
    origMat <- transformBoolNetToAdjacency(orig)

    binEval <- compareAdjacencyMatrices(origMatrix = origMat,
                                        reconMatrix = recMat)

    return(calculateSensitivity(binEval))
  }, origSet, recSet)},rndNWs,bestfitNetworksNoise, SIMPLIFY = F)

sensinferredNoise <- mapply(function(origSet, recSet) {
  mapply(function(orig, recon) {
    recMat <- transformBoolNetSetToAdjacency(recon)
    origMat <- transformBoolNetToAdjacency(orig)

    binEval <- compareAdjacencyMatrices(origMatrix = origMat,
                                        reconMatrix = recMat)

    return(calculateSensitivity(binEval))
  }, origSet, recSet)},rndNWs,inferredNetworksNoise, SIMPLIFY = F)

sensitivity <- cbind(bfNoise = sapply(sensBestFitNoise,mean),
      filteredNoise = sapply(sensinferredNoise,mean),
      filtered = sapply(sensInferred,mean),
      bf = sapply(sensBestFit,mean))

sensitivitySD <- cbind(bfNoise = sapply(sensBestFitNoise,sd),
      filteredNoise = sapply(sensinferredNoise,sd),
      filtered = sapply(sensInferred,sd),
      bf = sapply(sensBestFit,sd))

sensitivity <- cbind(sensitivity[,1], sensitivitySD[,1],
                     sensitivity[,2], sensitivitySD[,2],
                     sensitivity[,3], sensitivitySD[,3],
                     sensitivity[,4], sensitivitySD[,4])

rownames(sensitivity) <- seq(20,200,by=20)


specBestFit <- mapply(function(origSet, recSet) {
  mapply(function(orig, recon) {
```

```
    recMat <- transformBoolNetSetToAdjacency(recon)
    origMat <- transformBoolNetToAdjacency(orig)

    binEval <- compareAdjacencyMatrices(origMatrix = origMat,
                                        reconMatrix = recMat)

    return(calculateSpecificity(binEval))
  }, origSet, recSet)},rndNWs,bestfitNetworks, SIMPLIFY = F)

specInferred <- mapply(function(origSet, recSet) {
  mapply(function(orig, recon) {
    recMat <- transformBoolNetSetToAdjacency(recon)
    origMat <- transformBoolNetToAdjacency(orig)

    binEval <- compareAdjacencyMatrices(origMatrix = origMat,
                                        reconMatrix = recMat)

    return(calculateSpecificity(binEval))
  }, origSet, recSet)},rndNWs,inferredNetworks, SIMPLIFY = F)

specBestFitNoise <- mapply(function(origSet, recSet) {
  mapply(function(orig, recon) {
    recMat <- transformBoolNetSetToAdjacency(recon)
    origMat <- transformBoolNetToAdjacency(orig)

    binEval <- compareAdjacencyMatrices(origMatrix = origMat,
                                        reconMatrix = recMat)

    return(calculateSpecificity(binEval))
  }, origSet, recSet)},rndNWs,bestfitNetworksNoise, SIMPLIFY = F)

specinferredNoise <- mapply(function(origSet, recSet) {
  mapply(function(orig, recon) {
    recMat <- transformBoolNetSetToAdjacency(recon)
    origMat <- transformBoolNetToAdjacency(orig)

    binEval <- compareAdjacencyMatrices(origMatrix = origMat,
                                        reconMatrix = recMat)

    return(calculateSpecificity(binEval))
  }, origSet, recSet)},rndNWs,inferredNetworksNoise, SIMPLIFY = F)

specificity <- cbind(bfNoise = sapply(specBestFitNoise,mean),
      filteredNoise = sapply(specinferredNoise,mean),
      filtered = sapply(specInferred,mean),
      bf = sapply(specBestFit,mean))

specificitySD <- cbind(bfNoise = sapply(specBestFitNoise,sd),
      filteredNoise = sapply(specinferredNoise,sd),
      filtered = sapply(specInferred,sd),
      bf = sapply(specBestFit,sd))

specificity <- cbind(specificity[,1], specificitySD[,1],
```

```
                     specificity[,2], specificitySD[,2],
                     specificity[,3], specificitySD[,3],
                     specificity[,4], specificitySD[,4])

rownames(specificity) <- seq(20,200,by=20)
df <- list(bfNoise = specBestFitNoise,
                   filteredNoise = specinferredNoise,
                   bf = specBestFit,
                   filtered = specInferred)
library(reshape2)
library(ggplot2)
library(scales)
library(cowplot)
df <- melt(df)
names(df) <- c("specificity", "size", "type")
df$size <- as.numeric(df$size) * 20
df$size <- as.factor(df$size)
networkSize <- seq(20,200, by = 20)


point <- format_format(big.mark = " ", decimal.mark = ".", scientific = FALSE)

par(xpd = T)
par(mar = c(4,8,2,2),omi=c(4,0,0,0))
theme_set(theme_grey())
p1 <- ggplot(data=df[!grepl(pattern = "Noise", df$type),],
             aes(x=size, y=specificity, col = type, na.rm = T)) +
  geom_boxplot() +
  scale_x_discrete(name = "Number of Nodes",
                   breaks = networkSize,
                   labels = seq(20,200, by = 20)) +
  scale_y_continuous(name = "Specificity of predicted inputs",
                     labels = point) +
  theme(text = element_text(size=10)) +
  scale_color_discrete(name = "Algorithm",
                       labels = c("best-fit", "filtered best-fit"))

p2 <-ggplot(data=df[grepl(pattern = "Noise", df$type),],
            aes(x=size, y=specificity, col = type, na.rm = T)) +
  geom_boxplot() +
  scale_x_discrete(name = "Number of Nodes",
                   breaks = networkSize,
                   labels = seq(20,200, by = 20)) +
  scale_y_continuous(name = "Specificity of predicted inputs",
                     labels = point) +
  theme(text = element_text(size=10)) +
  scale_color_discrete(name = "Algorithm",
                       labels = c("best-fit", "filtered best-fit"))

plot_grid(p1, p2, ncol=1, labels = c("A", "B"), label_size=14, align = "v")
```



Reconstruction of Boolean networks from single cell data

After evaluation of the pipeline using random data, we applied the constructed pipeline to single-cell data using pseudo-time steps.

```
#helper function to sample random tuples from single cell data
generateTuples <- function(dataset, noSamples = 1000, seed = 534598)
{
  tuples <- combn(ncol(dataset), 2, simplify=FALSE)
  tuples <- c(tuples, lapply(tuples,rev))
  set.seed(seed)
  selectedTuples <- tuples[sample(1:length(tuples), noSamples)]

  return(lapply(selectedTuples, function(t) {apply(dataset[,t],
                                                   MARGIN = c(1,2),
                                                   as.numeric)}))
}
#save(splittedData, generateTuples,  file="NFKBdataset.RData")
```

Reconstruct Boolean networks

Boolean networks

```
library("BoolNet")
seeds <- c(294857,6547611,719537,37444,99837,
           9162,186,234,8678,4345,8795875,1892,
```

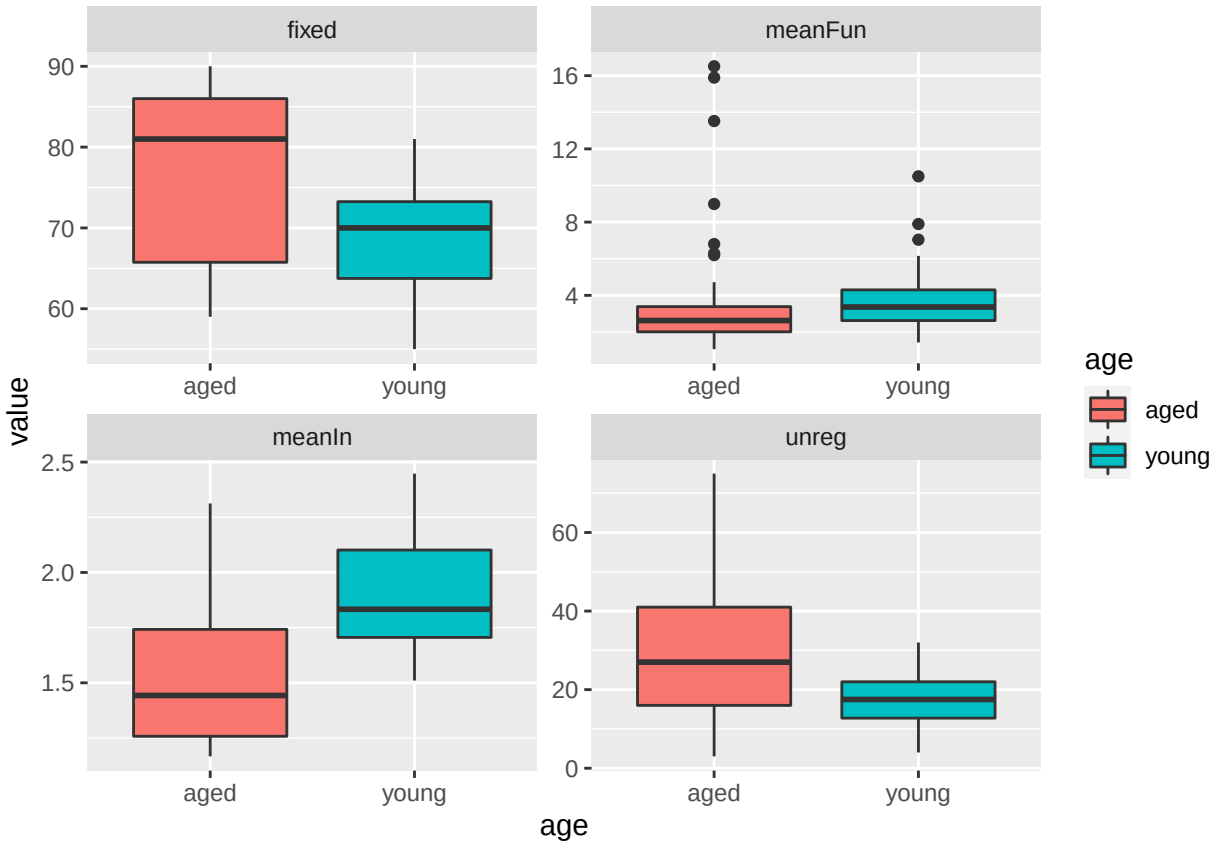
```
           57733,8334,12834,42323,6666,27487234,
           7875695,89798)

dataset <- "NFKBdataset.RData"
library("doParallel")
cl <- makeCluster(length(seeds))
registerDoParallel(cl)


networks <- foreach(s=seq_along(seeds)) %dopar%
{
load(dataset)
library("BoolNet")
source("InferViaCorrelation.R")
dyn.load("C_code/InferViaCorrelation.so")

inferedReconstruction <- function(timeSeries, threshold)
{
   inf_corr <- InferViaCorrelation_transitions(timeSeries,threshold)
   excluded <- lapply(inf_corr[[2]],
                      function(i) {setdiff(1:nrow(timeSeries[[1]]),i[[1]])})
   return(reconstructNetwork(timeSeries,
                             method='bestfit',
                             returnPBN=F,
                             excludedDependencies=excluded))
}


tuples <- lapply(splittedData, function(age)
  {
    lapply(age, function(smple)
        {
                seed <- seeds[s]
                generateTuples(smple,
                               noSamples = 1000,
                               seed = seed)})})


nets <- lapply(tuples, function(age)
{
    lapply(age, function(smple)
        {
          inferedReconstruction(smple, 0.03)
         })
})

return(nets)
}

stopCluster(cl)

save(networks, file = "reconstructedNetworksNFKB.RData")
```

Evaluation of reconstruction data

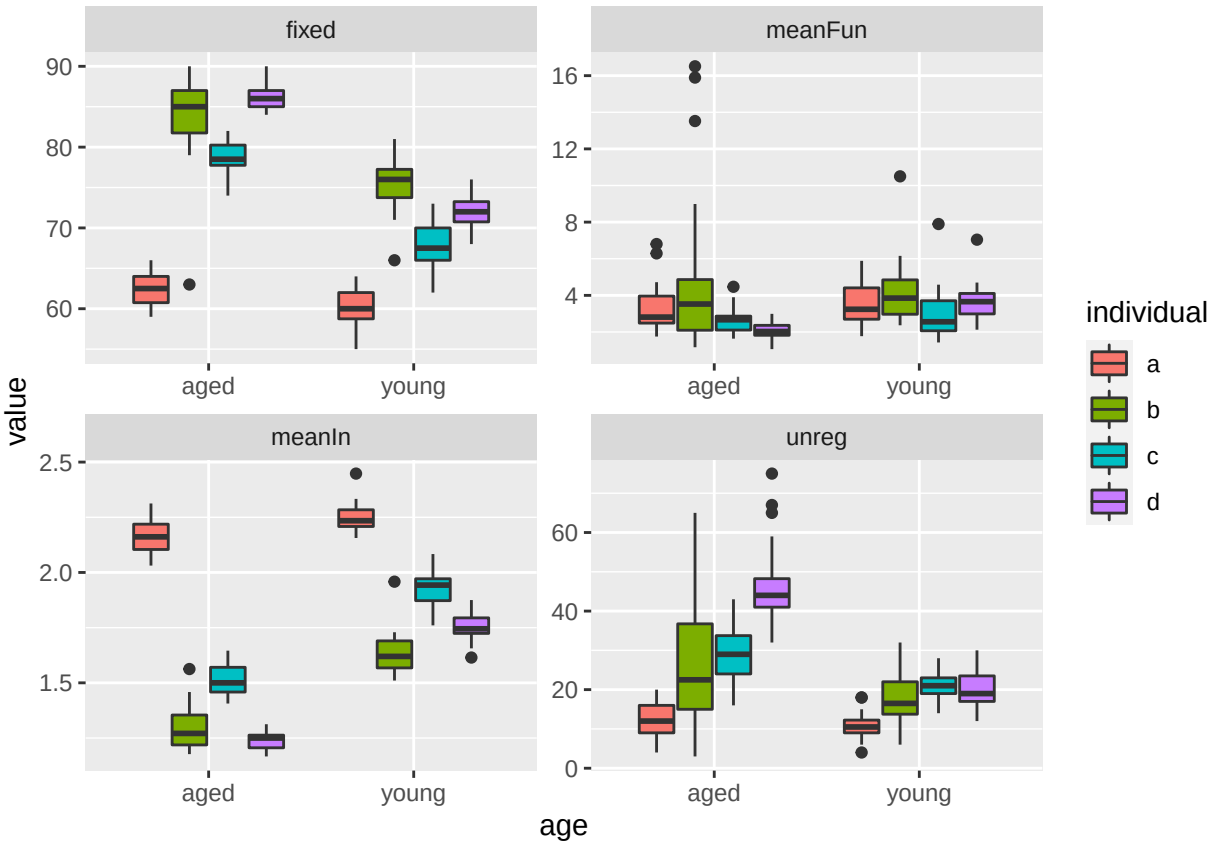
After reconstruction of the Boolean networks, we run different analyses such as measuring the mean number of regulatory inputs per Boolean function across the different networks. This analysis is repeatedly done per individual and differed only by aged group (young / aged).

```
load("reconstructedNetworksNFKB.RData")

getMeanNumberOfFunctions <- function(network)
{
  return(mean(sapply(network$interactions, length)))
}

getMeanNumberOfInputs <- function(network)
{
  return(mean(sapply(network$interactions, function(i) length(i[[1]]$input))))
}

getNumberofFixed <- function(network)
{
  return(sum(network$fixed !=  -1))
}

getInputNodes <- function(nw)
{
  regulators <- unique(unlist(sapply(nw$interactions,
                                     function(inp) sapply(inp, function(i) i$input,
                                                          simplify = F))))
  unregulated <- which(nw$fixed != -1)
  autoregulated <- getAutoRegulated(nw)
  return(intersect(regulators,union(unregulated, autoregulated)))
}


getUnregulatedNodes <- function(nw, assignment = c(0,1))
{
  regulators <- unique(unlist(sapply(nw$interactions,
                                     function(inp) sapply(inp,
                                                          function(i) i$input,
                                                          simplify = F))))
  unregulated <- which(nw$fixed %in% assignment)

  return(setdiff(unregulated,regulators))
}

getAutoRegulated <- function(nw)
{
  regulators <- unique(sapply(seq_along(nw$interactions), function(i) {

    if(any(sapply(nw$interactions[[i]],
                  function(fun) length(fun$input) == 1))
       &&
       any(sapply(nw$interactions[[i]], function(fun) fun$input == i)))
      return(i)
```

```
    else
      return(-1)


    }))
  return(regulators[which(regulators != -1)])
}

library("reshape2")
#measure mean connectivity / number of reconstructed function as network

aged <- lapply(networks, function(run) {lapply(run$aged, function(aged) {
  list(meanFun = getMeanNumberOfFunctions(aged),
       meanIn = getMeanNumberOfInputs(aged),
       fixed = getNumberofFixed(aged),
       unreg = length(getUnregulatedNodes(aged)))
})})
young <- lapply(networks, function(run) {lapply(run$young, function(young) {
  list(meanFun = getMeanNumberOfFunctions(young),
       meanIn = getMeanNumberOfInputs(young),
       fixed = getNumberofFixed(young),
       unreg = length(getUnregulatedNodes(young)))
})})

agedTab <- melt(aged)
youngTab <- melt(young)

colnames(agedTab) <- colnames(youngTab) <- c("value", "parameter", "age", "run")

overallTab <- melt(list(aged=aged,young=young))
colnames(overallTab) <- c("value", "parameter", "individual", "run", "age")

ggplot(overallTab, aes(y=value, x=age, fill=age)) +
  geom_boxplot() +
  facet_wrap(~parameter, scale="free")
```




```
ggplot(overallTab, aes(y=value, x=age, fill=individual)) +
  geom_boxplot() +
  facet_wrap(~parameter, scale="free")
```



Validation of network reconstruction using STRING DB

The reconstructed network interactions are validated by comparison to interactions of the same gene set from STRING DB.

```
#Required libraries:
library("BoolNet")
library("ggplot2")
library("tidyr")
library("tibble")
library("igraph")
#Read in networks:
load("reconstructedNetworksNFKB.RData") #loads as variable "networks"
#Read in tables:
STRINGtsv <- read.table("all_string_interactions.tsv")
STRINGtsv_dbAndExperimentOnly <- read.table("dbExp_string_interactions.tsv")
colnames(STRINGtsv) <-
  colnames(STRINGtsv_dbAndExperimentOnly) <-
  c("node1",    "node2", "node1_string_id",
    "node2_string_id",  "neighborhood_on_chromosome",
    "gene_fusion", "phylogenetic_cooccurrence", "homology",
    "coexpression", "experimentally_determined_interaction",    "database_annotated",
    "automated_textmining", "combined_score")

GeneNames <- networks[[1]]$aged$a$genes
```

```r
#Comparison function
compareAllRunsToSTRINGDB <- function(networks, STRINGtsv, age="young"){
  G <- length(networks[[1]]$aged$a$genes)
  GeneNames <- networks[[1]]$aged$a$genes
  #CCL4L2 is used in BN, STRING could only map to CCL4L1
  GeneNames[which(GeneNames == "CCL4L2")] <- "CCL4L1"
  UnionAdjmatOverRuns <- matrix(0, nrow = G, ncol = G)
  for (run in seq_along(networks)){
    #for every gene, loop over all possible regulatory functions of equal error
    #get union set of all possible regulators, put these into unionAdjmat->unionGraph,
    #i.e. keep adding 1s
    #Get names of Regulators, find indices of these names in GeneNames
    #Write these indices into corresponding rows of COL=g of unionMatrix
    unionAdjmatA <-
      unionAdjmatB <-
      unionAdjmatC <-
      unionAdjmatD <-
      matrix(0, nrow = G, ncol = G)
    for (g in 1:G){
      if (age == "young"){
        #list with each entry having $input (integer index) of Regulators
        AllAlternativeRules <- networks[[run]]$young$a$interactions[[g]]
      } else if (age == "aged"){
        AllAlternativeRules <- networks[[run]]$aged$a$interactions[[g]]
      }
      for (ar in AllAlternativeRules){
        inputIndices <- ar$input
        unionAdjmatA[inputIndices, g] <- 1
      }
    }
    for (g in 1:G){
      if (age == "young"){
         #list with each entry having $input (integer index) of Regulators
        AllAlternativeRules <- networks[[run]]$young$b$interactions[[g]]
      } else if (age == "aged"){
        AllAlternativeRules <- networks[[run]]$aged$b$interactions[[g]]
      }
      for (ar in AllAlternativeRules){
        inputIndices <- ar$input
        unionAdjmatB[inputIndices, g] <- 1
      }
    }
    for (g in 1:G){
      if (age == "young"){
        #list with each entry having $input (integer index) of Regulators
        AllAlternativeRules <- networks[[run]]$young$c$interactions[[g]]
      } else if (age == "aged"){
        AllAlternativeRules <- networks[[run]]$aged$c$interactions[[g]]
      }
      for (ar in AllAlternativeRules){
        inputIndices <- ar$input
        unionAdjmatC[inputIndices, g] <- 1
      }
```

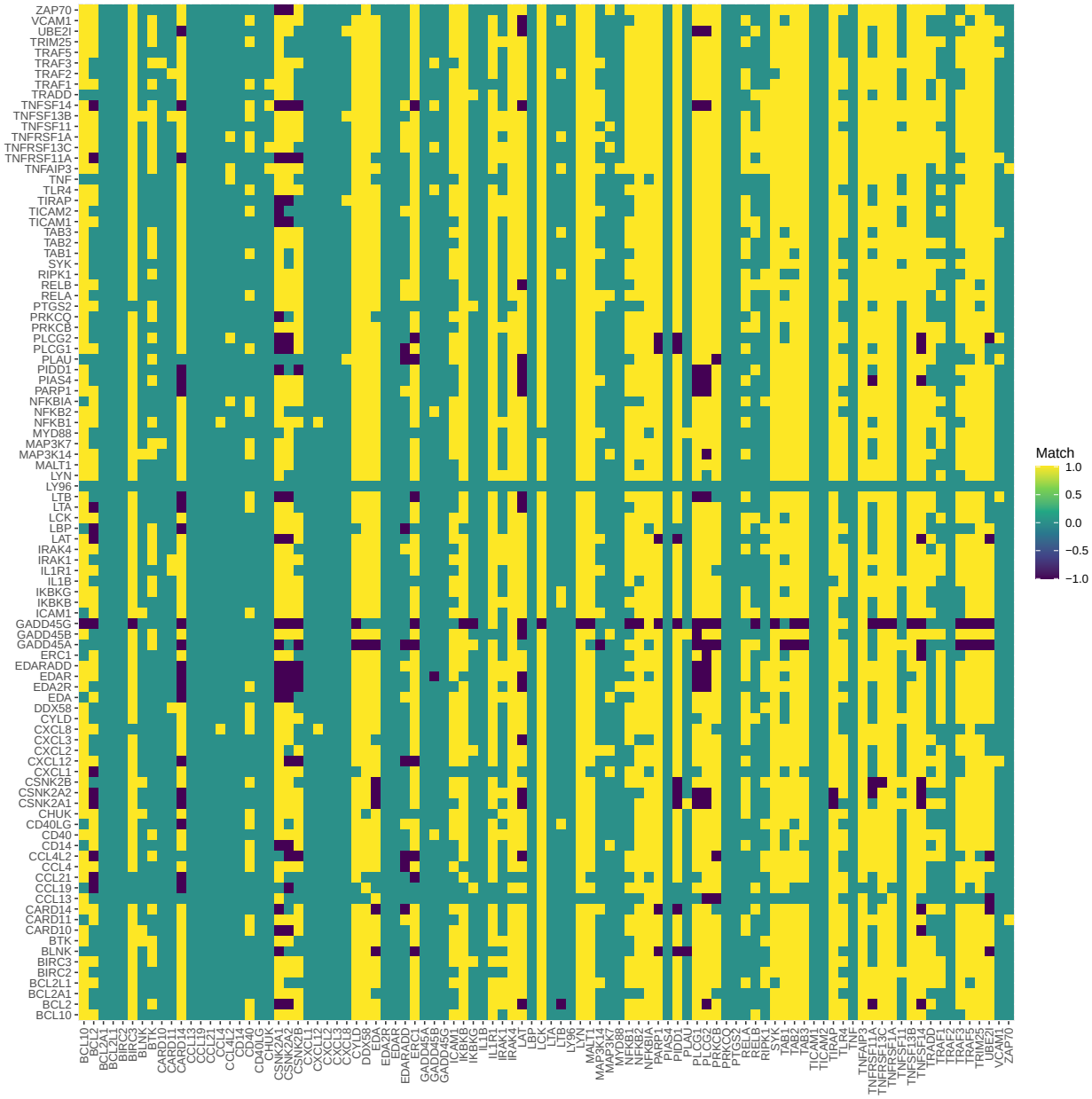
```
  }
  for (g in 1:G){
    if (age == "young"){
      #list with each entry having $input (integer index) of Regulators
      AllAlternativeRules <- networks[[run]]$young$d$interactions[[g]]
    } else if (age == "aged"){
      AllAlternativeRules <- networks[[run]]$aged$d$interactions[[g]]
    }
    for (ar in AllAlternativeRules){
      inputIndices <- ar$input
      unionAdjmatD[inputIndices, g] <- 1
    }
  }
  UnionAdjmatOverRuns <- UnionAdjmatOverRuns +
    unionAdjmatA +
    unionAdjmatB +
    unionAdjmatC +
    unionAdjmatD

}#end loop over runs

#binary matrix: edge exists in at least one of 20 runs or not
UnionAdjmatOverRuns[which(UnionAdjmatOverRuns > 1)] <- 1

#make igraph object out of stringtsv table -> stringGraph
stringGraph <- igraph::graph_from_edgelist(as.matrix(STRINGtsv[,c(1,2)]),
                                           directed = FALSE)
#directed interaction by default
#Need to match VertexNamesiGraph to order of GeneNames
permutationVec <- rep(NA, G)

if (length(V(stringGraph)) < length(GeneNames)){
  #some BN nodes were not captured by string, need to add them
  #manually to stringGraph, not connected to anything
  #Get subset GeneNames \ V(stringGraph)$name
  missingNodes <- setdiff(GeneNames, V(stringGraph)$name)
  stringGraph <- add_vertices(stringGraph, nv=length(missingNodes), name=missingNodes)
  # => Nr of genes in BN matches nr of nodes in iGraph again -> permutation possible
}
VertexNamesiGraph <- V(stringGraph)$name

for (g in 1:G){
  newIndex <- which(GeneNames == VertexNamesiGraph[g])
  permutationVec[g] <- newIndex
}
 #First element is new id of vertex 1 etc.
stringGraph <- igraph::permute(stringGraph, permutation=permutationVec)
shortestPathAdjmat <- matrix(NA, nrow = G, ncol = G)
colnames(shortestPathAdjmat) <- rownames(shortestPathAdjmat) <- GeneNames
#For every entry in unionGraph,
#get shortest path between these interactions in stringGraph
for (i in 1:G){
  for (j in 1:G){
```

```
      ShortestSTRINGpath <- get.shortest.paths(stringGraph,
                                               from=V(stringGraph)[i],
                                               to=V(stringGraph)[j], mode = "all")
      shortestPathAdjmat[i,j] <- length(ShortestSTRINGpath$vpath[[1]])
    }
  }
  entriesInSTRING <- shortestPathAdjmat
  entriesInSTRING[which(entriesInSTRING > 3)] <- 0

  #entriesInSTRING:     binary matrix to check which paths exists in STRING
  #(either directly or indirectly)
  #UnionAdjmatOverRuns: binary matrix to check which interactions exist in BN
  trinaryResult <- UnionAdjmatOverRuns
  for (i in 1:G){
    for (j in 1:G){
      if ((trinaryResult[i,j] == 1)
          &
          (entriesInSTRING[i,j] == 0) )
        {trinaryResult[i,j] <- -1}#exists in BN but doesn't match to STRING
    }
  }

  return(trinaryResult)
}

#Get 4 matrices, all young/old, all interactions/dbexp only => PLOT
trinaryMatrix_youngAll <- compareAllRunsToSTRINGDB(networks,
                                                   STRINGtsv,
                                                   age="young")
trinaryMatrix_agedAll <- compareAllRunsToSTRINGDB(networks,
                                                  STRINGtsv,
                                                  age="aged")
trinaryMatrix_young_dbExp <- compareAllRunsToSTRINGDB(networks,
                                                      STRINGtsv_dbAndExperimentOnly,
                                                      age="young")
trinaryMatrix_aged_dbExp <- compareAllRunsToSTRINGDB(networks,
                                                     STRINGtsv_dbAndExperimentOnly,
                                                     age="aged")


##### PLOT YOUNG+ALL INTERACTIONS #####
print("Plotting all young individuals, all interactions:")
```

```
## [1] "Plotting all young individuals, all interactions:"
```

```
df <- as.data.frame(trinaryMatrix_youngAll)
colnames(df) <- rownames(df) <- GeneNames
df %>%
  as.data.frame() %>%
  rownames_to_column("id") %>%
  pivot_longer(-c(id), names_to = "Gene", values_to = "Match") %>%
  ggplot(aes(x=Gene, y=id, fill=Match)) +
  geom_tile() +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1))+
```

```
  ylab(" ") +
  xlab(" ") +
  #ggtitle("Young, all interactions")+
  scale_fill_viridis_c()
```

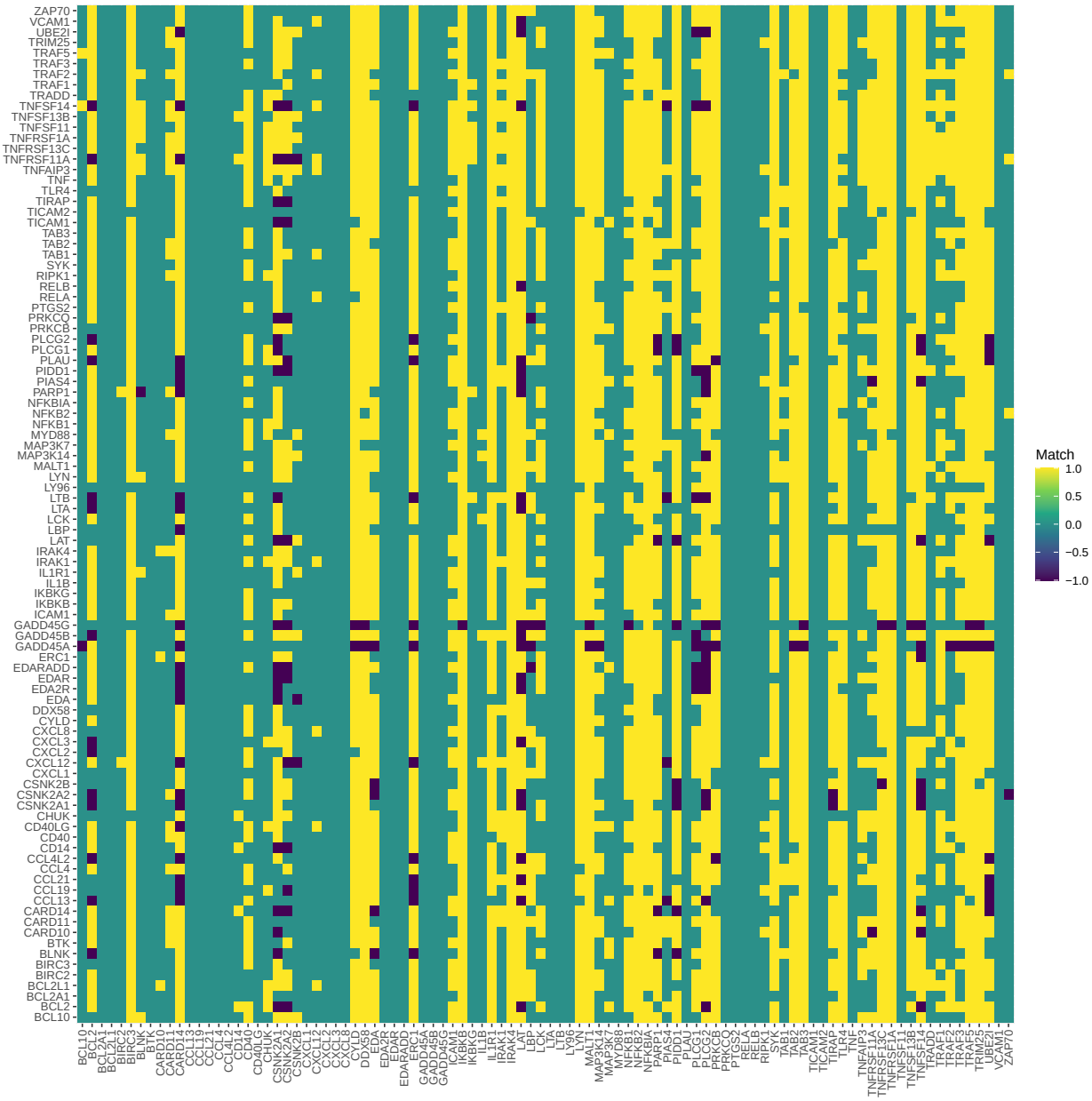



```
##### PLOT AGED+ALL INTERACTIONS #####
print("Plotting all aged individuals, all interactions:")
```

```
## [1] "Plotting all aged individuals, all interactions:"
```

```
df <- as.data.frame(trinaryMatrix_agedAll)
colnames(df) <- rownames(df) <- GeneNames
df %>%
  as.data.frame() %>%
  rownames_to_column("id") %>%
```

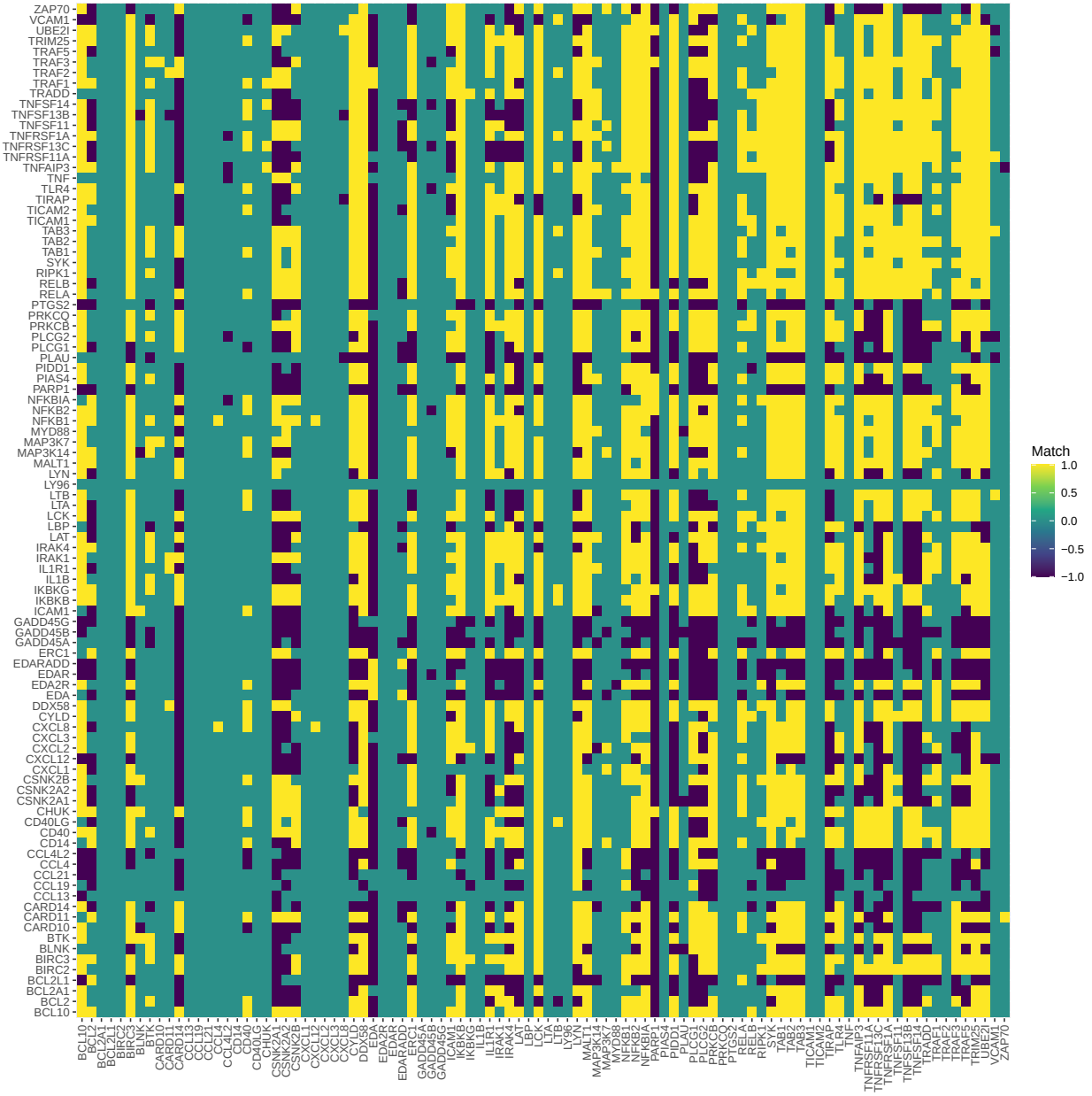
```
  pivot_longer(-c(id), names_to = "Gene", values_to = "Match") %>%
  ggplot(aes(x=Gene, y=id, fill=Match)) +
  geom_tile() +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1))+
  ylab(" ")+xlab(" ")+
  #ggtitle("Aged, all interactions")+
  scale_fill_viridis_c()
```




```
##### PLOT YOUNG+DB&EXP INTERACTIONS #####
print("Plotting all young individuals, db+exp interactions only:")
```

```
## [1] "Plotting all young individuals, db+exp interactions only:"
```

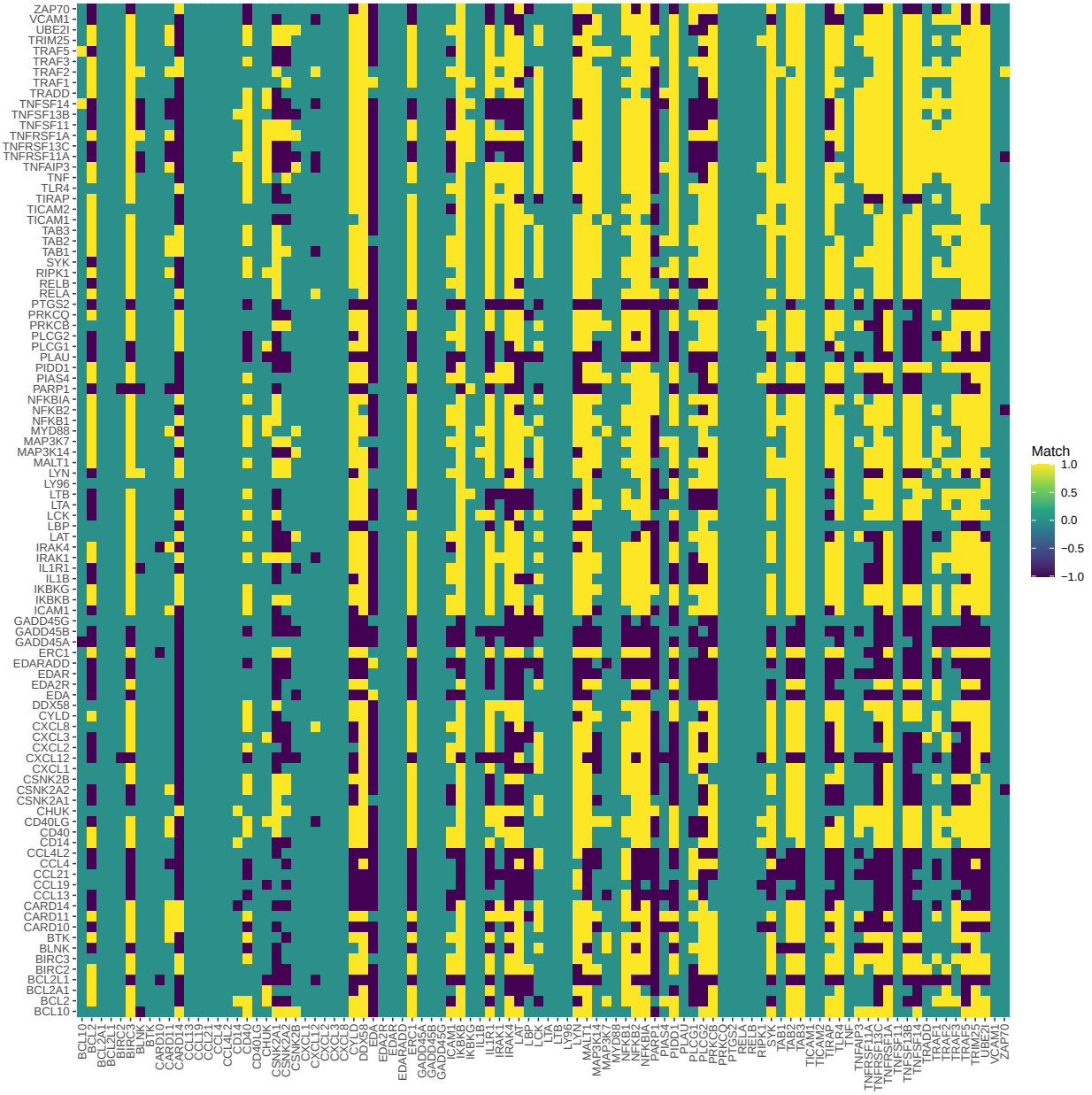
```r
df <- as.data.frame(trinaryMatrix_young_dbExp)
colnames(df) <- rownames(df) <- GeneNames
df %>%
  as.data.frame() %>%
  rownames_to_column("id") %>%
  pivot_longer(-c(id), names_to = "Gene", values_to = "Match") %>%
  ggplot(aes(x=Gene, y=id, fill=Match)) +
  geom_tile() +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1))+
  ylab(" ")+xlab(" ")+
  #ggtitle("Aged, db+exp interactions")+
  scale_fill_viridis_c()
```



```r
##### PLOT AGED+DB&EXP INTERACTIONS #####
print("Plotting all aged individuals, db+exp interactions only:")
```

```
## [1] "Plotting all aged individuals, db+exp interactions only:"
```

```r
df <- as.data.frame(trinaryMatrix_aged_dbExp)
colnames(df) <- rownames(df) <- GeneNames
df %>%
  as.data.frame() %>%
  rownames_to_column("id") %>%
  pivot_longer(-c(id), names_to = "Gene", values_to = "Match") %>%
  ggplot(aes(x=Gene, y=id, fill=Match)) +
  geom_tile() +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1))+
  ylab(" ")+xlab(" ")+
  #ggtitle("Aged, db+exp interactions")+
  scale_fill_viridis_c()
```



Analysis of Dynamic Network Properties

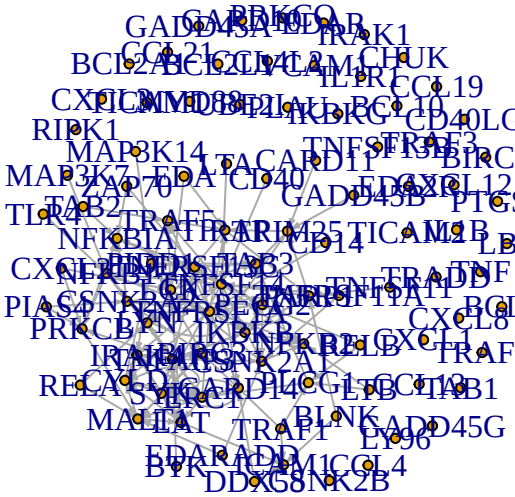
The dynamic properties of the different reconstructed Boolean networks were analysed using an ensemble approach. 100 Boolean networks were drawn randomly from the set of reconstructed functions for each individual in each run. For reconstructed functions with wildcards in the functions truth table (independent if 0/1), the wildcards were replaced by 0/1 randomly. The attractor landscape was computed for each drawn network using the sat exhaustive attractor search implemented in the BoolNet package. Results such as attractor length, number of attractors per network and the active genes in the different attractor landscapes were evaluated.

```
library(igraph)
library(qgraph)
#plot igraph object
```

```
plotNW <- function(nw)
{
  plotGraph <- plotNetworkWiring(nw, plotIt = F)
  lay <- qgraph.layout.fruchtermanreingold(get.edgelist(plotGraph,names=F),
                                           vcount=vcount(plotGraph),
                                           area=8*(vcount(plotGraph)^2),
                                           repulse.rad=(vcount(plotGraph)^3.1))

  plot(plotGraph,layout=lay,
       vertex.size=4,
       edge.arrow.size=0.2)

}

#function to sample random network from Boolean network collection
#as reconstructed from time series
sampleRandomNetwork <- function(reconstructedSet)
{
  functionIndices <- sapply(reconstructedSet$interactions,
                            function(x) sample(1:length(x),1))
  probabilisticNetwork <- reconstructedSet
  stopifnot(inherits(probabilisticNetwork, "ProbabilisticBooleanNetwork") |
              inherits(probabilisticNetwork, "BooleanNetworkCollection"))

  if (length(functionIndices) != length(probabilisticNetwork$genes))
    stop("Please provide a vector of function indices for each gene!")

  if (inherits(probabilisticNetwork, "ProbabilisticBooleanNetwork")) {
    interactions <- mapply(function(interaction, index) {
      list(input = interaction[[index]]$input, func = interaction[[index]]$func,
           expression = interaction[[index]]$expression)
    }, probabilisticNetwork$interactions, functionIndices,
    SIMPLIFY = FALSE)
  }
  else {
    interactions <- mapply(function(interaction, index, gene) {
      func <- interaction[[index]]$func
      dcPos <- which(func == -1)
      if (length(dcPos) > 0) {
        func[dcPos] <- sample(0:1,length(dcPos), r = T)
        expression <- BoolNet:::getInteractionString(F,
                                    func,
                                    probabilisticNetwork$genes[interaction[[index]]$input])
      }
      else expression <- interaction[[index]]$expression
      list(input = interaction[[index]]$input, func = func,
           expression = expression)
    }, probabilisticNetwork$interactions, functionIndices,
    probabilisticNetwork$genes, SIMPLIFY = FALSE)
  }
  res <- list(genes = probabilisticNetwork$genes, interactions = interactions,
              fixed = probabilisticNetwork$fixed)
  class(res) <- "BooleanNetwork"
```

```
  return(res)
}

set.seed(68867876)
agedSample <- sampleRandomNetwork(networks[[sample(1:20,1)]]$aged[[sample(1:4,1)]])
set.seed(345345)
youngSample <- sampleRandomNetwork(networks[[sample(1:20,1)]]$young[[sample(1:4,1)]])


plotNW(agedSample)
```

```
plotNW(youngSample)
```



```
#function to measure the overlapping attractors between to attractor objects
attractorIntersect <- function(attrs1, attrs2){
  intersect <- list()
  counter <- 1

  #skip escaped attrs
  if(!is.list(attrs2))
    return(attrs1)
```

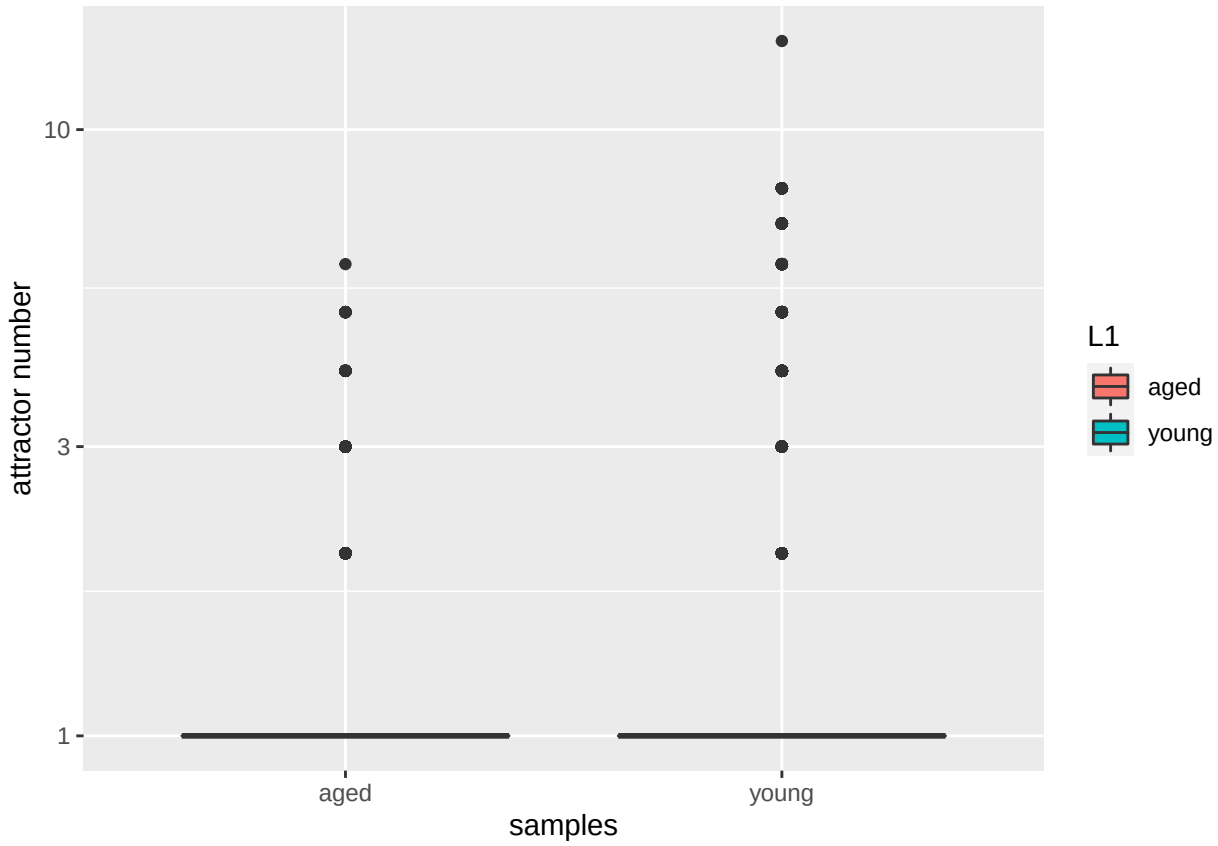
```
  for (a1 in 1:length(attrs1$attractors)){
    attr1 <- attrs1$attractors[[a1]]
    for (a2 in 1:length(attrs2$attractors)){
      attr2 <- attrs2$attractors[[a2]]
      if (identical(attr1$involvedStates, attr2$involvedStates)){
        intersect[[counter]] <- attr1
        counter <- counter + 1
        break
      }
    }
  }
  if(length(intersect) == 0)
  {
    intersect <- list(list(involvedStates = 0, basinSize = NA))
  }
  res <- list(attractors = intersect,stateInfo = attrs1$stateInfo )
  return(res)
}



createAttractorSummary <- function(results,
                                   seed=13342,
                                   networkName="NFkB",
                                   threads = detectCores() - 1)
{
  #resort by age
  young <- Reduce(function(acc, new) {
    acc$a <- c(acc$a, new["a"])
    acc$b <- c(acc$b, new["b"])
    acc$c <- c(acc$c, new["c"])
    acc$d <- c(acc$d, new["d"])
    return(acc)
  }, lapply(results, function(r) r$young), init = list(a=list(),
                                                       b=list(),
                                                       c=list(),
                                                       d=list()))

  aged <- Reduce(function(acc, new) {
    acc$a <- c(acc$a, new["a"])
    acc$b <- c(acc$b, new["b"])
    acc$c <- c(acc$c, new["c"])
    acc$d <- c(acc$d, new["d"])
    return(acc)
  }, lapply(results, function(r) r$aged), init = list(a=list(),
                                                      b=list(),
                                                      c=list(),
                                                      d=list()))

  library("BoolNet")
  library("doParallel")
  attractorsYoung <- lapply(young, function(age)
  {
```

```
  lapply(age, function(run)
  {
    set.seed(seed) #354345
    networks <- replicate(100, sampleRandomNetwork(run), simplify = F)
    cl <- makeCluster(threads)
    registerDoParallel(cl)
    attractors <- foreach(i=seq_along(networks)) %dopar%
      {
        library("BoolNet")
        attractorsRun <- tryCatch(attractorsRun <- getAttractors(networks[[i]],
                                                  method = "sat.exhaustive"),
                                  error = function(e) {attractorsRun <- 0})

        attractorsRun
      }
    stopCluster(cl)
    attractors
  })
})

nameList <- c("A", "B", "C", "D")
nameCount <- 1
commonAttractorsYoung <- lapply(attractorsYoung, function(age) {
  count <- 1
  lapply(age, function(run){
    res <- Reduce(f = attractorIntersect, run, init = run[[1]], accumulate = F)
    class(res) <- "AttractorInfo"

    if(length(res$attractors[[1]]$involvedStates == 1)
       &&
       res$attractors[[1]]$involvedStates != 0){
      png(paste("./", networkName,
                "attractorYoung_",
                nameList[nameCount],
                "_Run" ,
                count,
                ".png", sep = ""), height = 2500, width = 500)
      plotAttractors(res)
      dev.off()
      count <<- count + 1
    }
    return(res)
  })
  nameCount <<- nameCount + 1
})

attractorsAged <- lapply(aged, function(age)
{
  lapply(age, function(run)
  {
    set.seed(seed)
    networks <- replicate(100, sampleRandomNetwork(run), simplify = F)
    cl <- makeCluster(threads)
```

```
      registerDoParallel(cl)
      attractors <- foreach(i=seq_along(networks)) %dopar%
        {
          library("BoolNet")
          attractorsRun <- tryCatch(attractorsRun <- getAttractors(networks[[i]],
                                                    method = "sat.exhaustive"),
                                    error = function(e) {attractorsRun <- 0})
          attractorsRun
        }
      stopCluster(cl)
      attractors
      return(attractors)
    })
  })

  return(list(young = attractorsYoung,
              aged = attractorsAged,
              runwiseAttractorsYoung = commonAttractorsYoung,
              runwiseAttractorsAged = commonAttractorsAged))
}

dynNFkB <- createAttractorSummary(results = networks, networkName = "NFkB")
#contains all results in young/aged and reduced attractors by run
#(Reduce over all 100 randomly sampled networks per run) in runwiseAttractorsYoung/Aged
save(dynNFkB, file="./dynResultsNFkB.RData")
```

```
load("dynResultsNFkB.RData")

#order individuals according to their corresponding age
orderYoung <- c("a","c","d","b")
orderAged <- c("c","a","d","b")

getAttractorProperties <- function(summaryResults, pathwayName="NFkB")
{
  totalResults <- list(young = summaryResults$young, aged = summaryResults$aged)

  attractorLength <- lapply(totalResults, function(age) {
    lapply(age, function(sample)
      {
        lapply(sample, function(run)
        {
          lapply(run,function(simulation) {
            if("attractors" %in% names(simulation))
              sapply(simulation$attractors, function(attr)
                {if ("involvedStates" %in% names(attr))
                  ncol(attr$involvedStates)
                else
                  0})
            else
              0
         })
      })
    })
  })
```

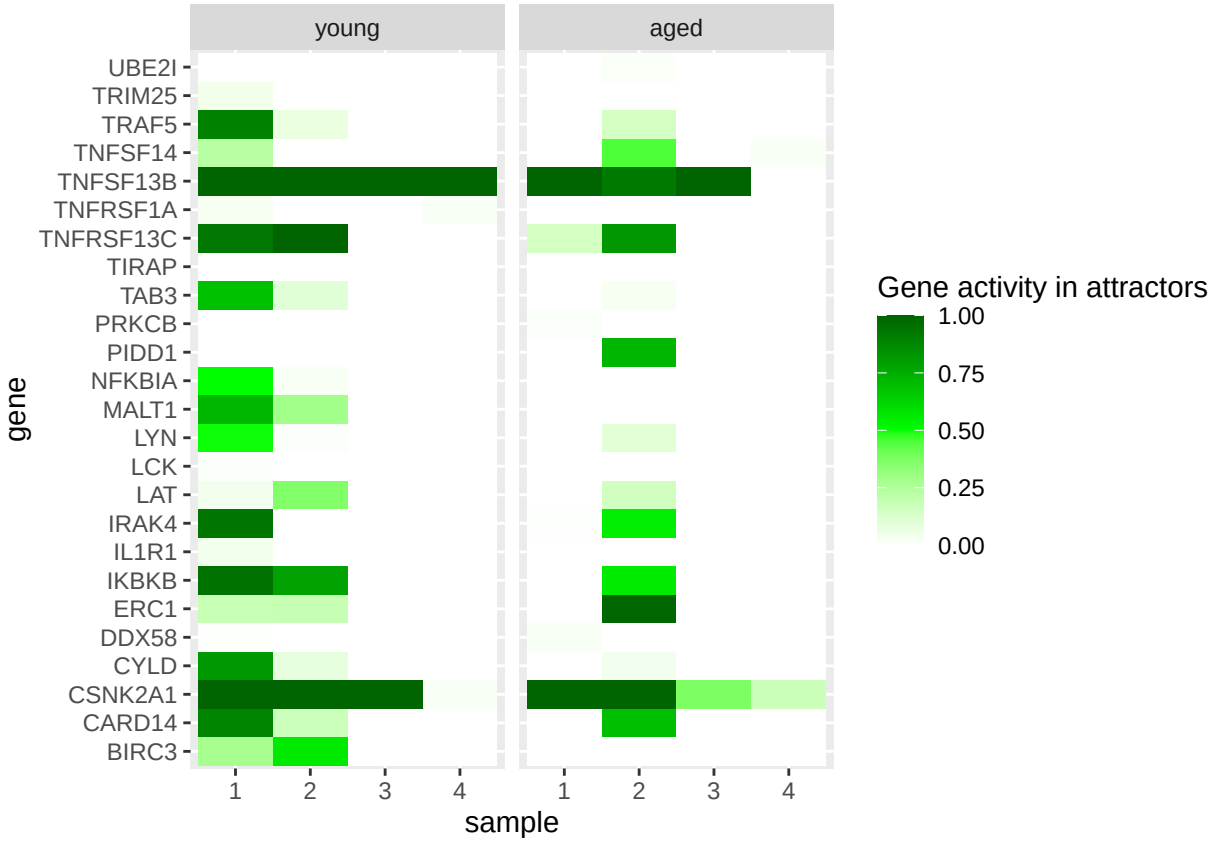
```
  attractorNumbers <- lapply(totalResults, function(age) {
    lapply(age, function(sample)
    {
      lapply(sample, function(run)
      {
        lapply(run,function(simulation) {
          if("attractors" %in% names(simulation))
            length(simulation$attractors)
          else
            0
        })
      })
    })
  })

  library("reshape2")
  library("ggplot2")

  attractorLengthTab <- melt(attractorLength)
  plot <- ggplot(attractorLengthTab, aes(y=value, x=L1, fill=L1)) +
    geom_boxplot() +
    scale_y_log10() +
    ylab("attractor length") +
    xlab("samples")
  plot



  attractorNumberTab <- melt(attractorNumbers)

  plot <- ggplot(attractorNumberTab, aes(y=value, x=L1, fill=L1)) +
    geom_boxplot() +
    scale_y_log10() +
    ylab("attractor number") +
    xlab("samples")

  plot
}

#plot attractor properties
getAttractorProperties(dynNFkB,pathwayName = "NFkB")
```




```
#measure genewise activity inside attractors
geneActivityInAttractors <- function(summaryResults)
{
  totalResults <- list(young = summaryResults$young, aged = summaryResults$aged)

  attractorActivity <- lapply(totalResults, function(age) {
    lapply(age, function(sample)
    {
      summedRuns <- lapply(sample, function(run)
      {
        summed <- lapply(run,function(simulation) {
          if("attractors" %in% names(simulation)){
            binarizedStates <- lapply(simulation$attractors, function(attr)
            {
              bin <- BoolNet:::dec2bin(attr$involvedStates,
                                       length(simulation$stateInfo$genes))
              bin
            })
            Reduce("+", binarizedStates) / length(simulation$attractors)
          }
        })
        Reduce("+", summed) / length(run)
     })
      res <- Reduce("+", summedRuns) / length(sample)
      names(res)  <- dynNFkB$young$a[[1]][[1]]$stateInfo$genes
      res
  })
```

```
  })

  return(attractorActivity)
}

genewiseActivity <- geneActivityInAttractors(dynNFkB)

countAge <- 1
plots <- lapply(genewiseActivity, function(age) {
  countSample <- 1
    lapply(age, function(sample)
    {
      p <- ggplot(data.frame(value = sample,
                             name = dynNFkB$young$a[[1]][[1]]$stateInfo$genes),
                  aes(x = value, y = name, fill = value)) +
        geom_bar(stat="identity")
      p
      countSample <<- countSample + 1
  })
  countAge <<- countAge + 1

})


genewiseActivity <- geneActivityInAttractors(dynNFkB)
impactGenes <- Reduce("union",
                      lapply(genewiseActivity,
                             function(age) {Reduce("union",
                                                   lapply(age, function(g) names(g[g > 0])))}))

genewiseActivity <- lapply(genewiseActivity, function(age) {
  lapply(age,
         function(sample)
           lapply(split(sample[which(names(sample) %in% impactGenes)],
                        names(sample[which(names(sample) %in% impactGenes)])),unname))
})
library("reshape2")
genewiseActivityTab <- melt(genewiseActivity)

colnames(genewiseActivityTab) <- c("value", "gene", "sample", "age")

genewiseActivityTab$age <- factor(genewiseActivityTab$age,      # Reordering group factor levels
                                  levels = c("young", "aged"))
for(i in 1:length(orderYoung))
  genewiseActivityTab[genewiseActivityTab$age
                      ==
                      "young", ]$sample[genewiseActivityTab[genewiseActivityTab$age
                                                            ==
                                                            "young", ]$sample
                                        ==
                                        orderYoung[i]] <- i

for(i in 1:length(orderAged))
```

```
  genewiseActivityTab[genewiseActivityTab$age
                      ==
                      "aged", ]$sample[genewiseActivityTab[genewiseActivityTab$age
                                                           ==
                                                           "aged", ]$sample
                                       ==
                                       orderAged[i]] <- i


ggplot(genewiseActivityTab, aes(x = sample, y = gene, fill = value)) +
  geom_tile() +
  facet_wrap("age") +
  scale_fill_gradient2(low = "white", high = "darkgreen", mid = "green",
                       midpoint = 0.5, limit = c(0,1), space = "Lab",
                       name="Gene activity in attractors")
```




```
geneActivityInAttractorsRaw <- function(summaryResults)
{
  totalResults <- list(young = summaryResults$young, aged = summaryResults$aged)

  attractorActivity <- lapply(totalResults, function(age) {
    lapply(age, function(sample)
    {
      summedRuns <- lapply(sample, function(run)
      {
        summed <- lapply(run,function(simulation) {
          if("attractors" %in% names(simulation)){
```

```
            binarizedStates <- lapply(simulation$attractors, function(attr)
            {
              bin <- BoolNet:::dec2bin(attr$involvedStates,
                                       length(simulation$stateInfo$genes))
              names(bin) <- simulation$stateInfo$genes
              as.list(bin)
            })
            binarizedStates
          }
        })
        summed
    })
     summedRuns
  })
  })

  return(attractorActivity)
}
```

Screen Boolean networks for motifs

We screened randomly sampled Boolean networks from each ensemble for the presence of feed forward loop and bi-fan motifs using the R-package igraph. To do so, we counted the number of occurrences of each motif in each of the sampled networks and created box plots for these. Differences between young and aged group were measured using wilcoxon test.

```
library("BoolNet")
library("igraph")
library("gtools")
load("reconstructedNetworksNFKB.RData")

library("doParallel")
motifScreening <- list(aged = list(a= list(),
                                   b= list(),
                                   c=list(),
                                   d=list()),
                       young = list(a= list(),
                                    b= list(),
                                    c=list(),
                                    d=list()))
for(n in seq_along(networks)){
  for(age in seq_along(networks[[n]])){
      for(indiv in seq_along(networks[[n]][[age]])){
        {
          sampledNetworks <- replicate(100,
                                       sampleRandomNetwork(networks[[n]][[age]][[indiv]]),
                                       simplify = F)
          graphObjects <- lapply(sampledNetworks,
                                 function(net) plotNetworkWiring(net,plotIt = F))
          cl <- makeCluster(100)
          registerDoParallel(cl)
         ffl <- foreach(s=seq_along(sampledNetworks),.packages= c("BoolNet",
                                                                  "igraph",
```

```
                                                                  "gtools"))
        %dopar%
         {
           return(igraph::count_subgraph_isomorphisms(
             patter = igraph::graph_from_isomorphism_class(3,7), graphObjects[[s]]))
         }
         bifan <- foreach(s=seq_along(sampledNetworks),.packages= c("BoolNet",
                                                                    "igraph",
                                                                    "gtools"))
         %dopar%
         {
           return(igraph::count_subgraph_isomorphisms(
             patter = igraph::graph_from_isomorphism_class(4,19), graphObjects[[s]]))
         }
         stopCluster(cl)

         motifScreening[[age]][[indiv]][[n]] <- list(ffl = ffl,bifan = bifan)
        }
}}}

save(motifScreening, file="motifScreeningiGraph.RData")
```

```
load("motifScreeningiGraph.RData")
library("ggpubr")
```

```
##
## Attaching package: 'ggpubr'
```

```
## The following object is masked from 'package:cowplot':
##
##     get_legend
```

```
library("rstatix")
```

```
##
## Attaching package: 'rstatix'
```

```
## The following object is masked from 'package:biomaRt':
##
##     select
```

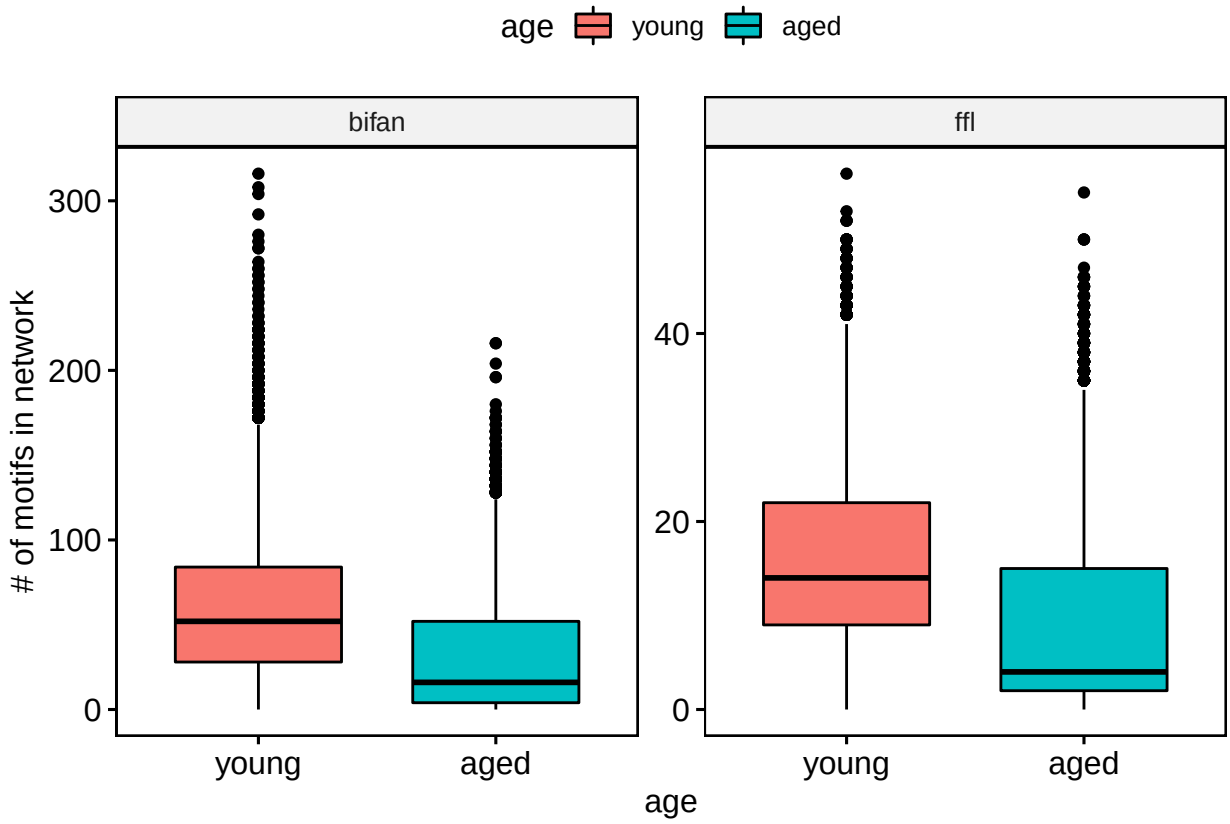
```
## The following object is masked from 'package:stats':
##
##     filter
```

```
motifSummed <- lapply(motifScreening,
                      function(age) lapply(age, function(indiv)
                        lapply(indiv, function(rep)
                          lapply(rep, function(motif)
                            lapply(motif, sum)))))
motifTab <- melt(motifSummed)
colnames(motifTab) <- c("value", "network", "motif", "repetition", "individual", "age")

pValBifan <- wilcox.test(motifTab$value[motifTab$motif == "bifan"
                                        &
                                        motifTab$age == "young"],
                         motifTab$value[motifTab$motif == "bifan"
```

```
                                          &
                                          motifTab$age == "aged"])
pValffl <- wilcox.test(motifTab$value[motifTab$motif == "ffl"
                                      &
                                      motifTab$age == "young"],
                       motifTab$value[motifTab$motif == "ffl"
                                      &
                                      motifTab$age == "aged"])
p.adjust(c(pValBifan$p.value, pValffl$p.value), method = "bonferroni")
```

```
## [1] 0 0
```

```
ggboxplot(motifTab,
          y = "value",
          x = "age",
          fill = "age",
          facet.by = "motif",
          scales = "free_y",
          order = c("young", "aged"),
          ylab = "# of motifs in network")
```




```
ggboxplot(motifTab,
          y = "value",
          x = "age",
          fill = "individual",
          facet.by = "motif",
          scales = "free_y",
          order = c("young", "aged"),
```

```
        ylab = "# of motifs in network")
```