simulation_SS_extraction

Quarto

Quarto enables you to weave together content and executable code into a finished document. To learn more about Quarto see https://quarto.org.

Running Code

When you click the **Render** button a document will be generated that includes both content and the output of embedded code. You can embed code like this:

It may be necessary to run “git push -f” in the terminal to do a “force” push each time local files are modified.

```r
# For R/4.4.0}

# install.packages("htmlwidgets")
# install.packages("devtools")
library(devtools)
```

```
Loading required package: usethis
```

```r
# install_version("apTreeshape", version = "1.5-0.1")
library(apTreeshape)
```

```
Loading required package: ape
```

```
Registered S3 method overwritten by 'apTreeshape':
  method          from
  is.binary.phylo ape
```

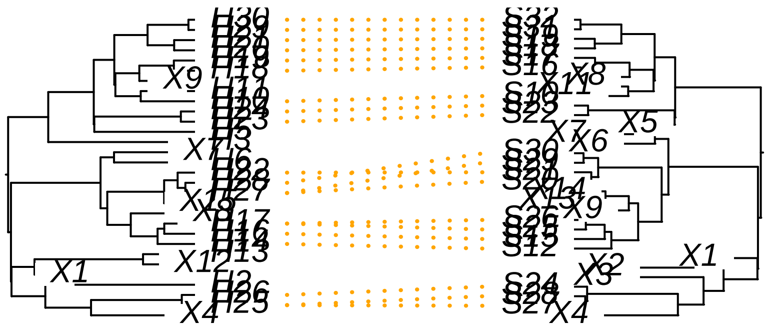
```
Attaching package: 'apTreeshape'


The following object is masked from 'package:ape':

    is.binary.phylo
```

```
# install_version("treeducken", version = "1.1.0")
library(treeducken)
library(phytools)
```

```
Loading required package: maps
```

```
set.seed(54)

# host speciation
lambda_H <- rexp(n=1)

# host extinction fraction
mu_H <- 0.3

# cospeciation
lambda_C <- rexp(n=1)

# time
time <- 2

# symbiont speciation
lambda_S <- rexp(n=1)

# symbiont extinction fraction
mu_S <- 0.3

lambda_total_H <- lambda_H + lambda_C
lambda_total_S <- lambda_S + lambda_C

# calculate the expected number of tips for the host and symbiont trees
H_tips <- ave_tips_st(lambda = lambda_total_H, mu = mu_H, t = time)
S_tips <- ave_tips_st(lambda = lambda_total_S, mu = mu_S, t = time)

cophy_obj <- sim_cophyBD(hbr = lambda_H,
```

```r
                           hdr = mu_H,
                           sbr = lambda_S,
                           sdr = mu_S,
                           cosp_rate =lambda_C,
                           host_exp_rate = 0.0,
                           time_to_sim = time,
                           numbsim = 1)

# visualization using treeDucken, with imperfections
plot.cophy(cophy_obj[[1]], col = "orange", lty = "dotted", lwd = 2)
```

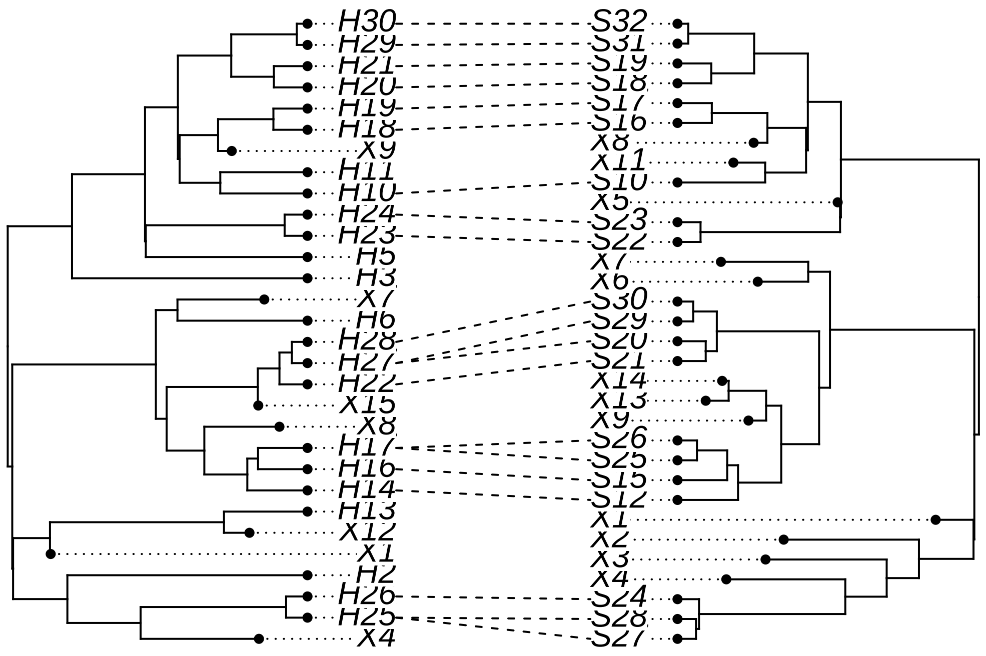


```r
# visualization using ape or phytools, transformation of the association matrix needed
ass_mat_trans <- NULL
ass_mat <- cophy_obj[[1]][[3]]
for(i in 1:nrow(ass_mat)){
  for (j in 1:ncol(ass_mat)){
    if (ass_mat[i,j] == 1)

      ass_mat_trans <- rbind(ass_mat_trans, c(rownames(ass_mat)[i], colnames(ass_mat)[j]))

  }
}
```

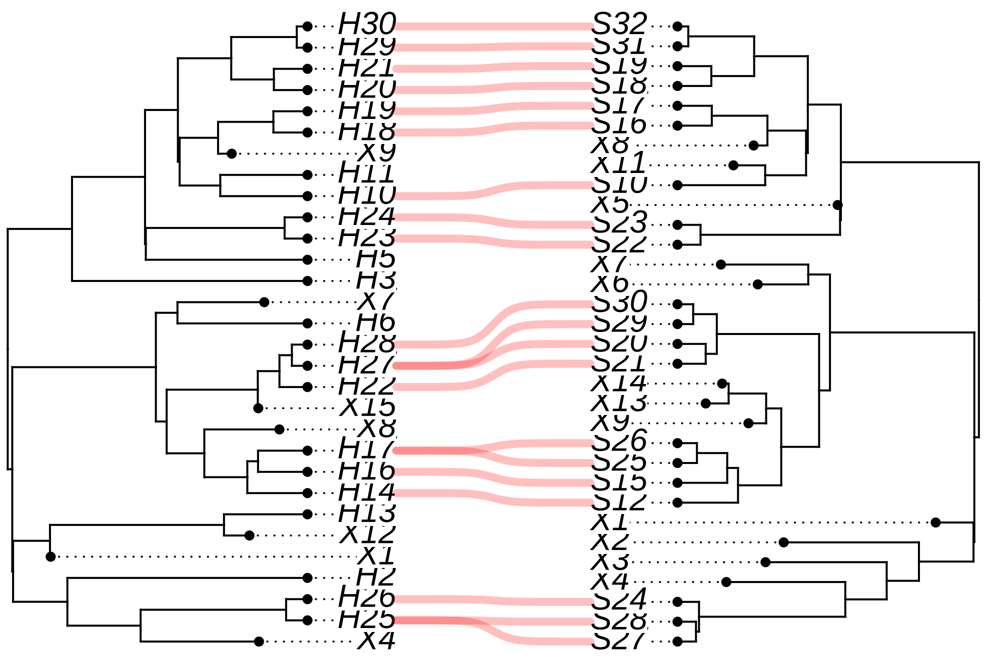
```r
# using ape / phytools
cophylo_plot <- cophylo(cophy_obj[[1]][[1]], cophy_obj[[1]][[2]], assoc = ass_mat_trans)
```

```
Rotating nodes to optimize matching...
Done.
```

```r
plot(cophylo_plot)
```



```r
plot(cophylo_plot,link.type="curved",link.lwd=4,
     link.lty="solid",link.col=make.transparent("red",
                                                0.25))
```



```
# par(mar=c(5.1,4.1,4.1,2.1))
```

Now, generate multiple co-phylogenies and plot the three representative summary statistics as a distribution.

```
cophy <- NULL

for (cophy_id in 1:15) {
  print(cophy_id)

  cophy_obj <- NULL

  # time
  time <- 2

  # host speciation
  lambda_H <- rexp(n=1)

  # symbiont speciation
  lambda_S <- rexp(n=1)

  # cospeciation
  lambda_C <- rexp(n=1)

  # host extinction rate
  mu_H_frac <- 0.3
  mu_H <- mu_H_frac * (lambda_H + lambda_C)

  # symbiont extinction rate
  mu_S_frac <- 0.3
  mu_S <- mu_S_frac * (lambda_S + lambda_C)

  lambda_total_H <- lambda_H + lambda_C
  lambda_total_S <- lambda_S + lambda_C

  # calculate the expected number of tips for the host and symbiont trees
  H_tips <- ave_tips_st(lambda = lambda_total_H, mu = mu_H, t = time)
  S_tips <- ave_tips_st(lambda = lambda_total_S, mu = mu_S, t = time)

  cophy_obj <- sim_cophyBD(hbr = lambda_H,
                           hdr = mu_H,
                           sbr = lambda_S,
                           sdr = mu_S,
                           cosp_rate =lambda_C,
                           host_exp_rate = 0.0,
                           time_to_sim = time,
                           numbsim = 1)

  cophy[[length(cophy) + 1]] <- cophy_obj

}
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
[1] 11
[1] 12
```

```
[1] 13
[1] 14
[1] 15
```

Phylogenetic summary statistics represented by phylogenetic distance.

```r
library(rlist)
library(ape)

# specify the name of the output folder
output_dir <- "Output_extinction"

# specify the number of replicates
n_rep <- 3
n_gen <- 1000

# calculate the mean phylogenetic distance among host lineages
pd_host <- rep(NA, n_rep * length(sim_names))

for (sim_pos in 1:length(sim_names)){
  for (rep_id in 1:n_rep) {
    print(sim_names[sim_pos])

    # read in the population and species info
    file_name <- paste(output_dir, "/", as.character(sim_names[sim_pos]), "/", as.character(r
    pop_spec_host <- list.load(file_name)

    # calculate a vector recording the mean phylogenetic distance among host lineages for eac
    pd_host_gens <- rep(NA, n_gen)

    for (gen_id in (n_gen - 1 + 1):n_gen) {

      # read in needed lists for the current generation
      host_spec_list <- pop_spec_host[[gen_id]]$host_spec_list

      # read in the needed phylogeny file
      file_name <- paste(output_dir, "/", as.character(sim_names[sim_pos]), "/", as.characte
      tree_host <- read.tree(file_name)

      # trim the host tree to the generation of interest (here the generation of is of concer
      if(gen_id < n_gen){
        for(prune_gen in n_gen:(gen_id+1)){
        # get the number of species at this generations
        n_spec_host <- length( pop_spec_host[[prune_gen]]$host_spec_list )

        # create a vector for the taxon names to be removed
        rm_host <- paste ('_', as.character(prune_gen), '.', as.character(1:n_spec_host), '_
        tree_host <- drop.tip(tree_host, tip = rm_host, trim.internal = F, rooted = T, colla

      }

      }

      # calculate mean phylogenetic distance for this generation
      pd_host_gens[gen_id] <- mean(cophenetic.phylo(tree_host))
    }

    pd_host[(sim_pos - 1) * n_rep + rep_id] <- mean(pd_host_gens[(n_gen - 1 + 1) : n_gen])

  }
}



# create a vector recording which treatment each value belongs to
nam <- rep(NA, length(sim_names) * n_rep)
for ( rep_id in 1:n_rep) {
  nam[(1:length(sim_names)) * n_rep - (n_rep - rep_id)] <- sim_names
}

# create a data frame
phylo_dist_host <- data.frame(treatment=nam, pd_host=pd_host)

# (do not do this for formal analysis) make pseudo-replicates to get a sense of the effects o
# nest_mod <- rbind(nest_mod, nest_mod, nest_mod, nest_mod, nest_mod, nest_mod, nest_mod, nes

# save the data frame to a spreadsheet
write.csv(phylo_dist_host, paste(output_dir, "/phylo_dist_host.csv", sep = ''))
```

Network summary statistics represented by nestedness (z-score).

```
library(ggplot2)
library(rlist)
library(permute)
library(lattice)
#install.packages("Matrix")
# remove.packages("vegan")
# install.packages("vegan")
library(vegan)
library(statnet.common)
library(network)
library(sna)
#install.packages("spam")
#install.packages("bipartite")
library(bipartite)
library(GGally)
#install.packages("https://cran.r-project.org/src/contrib/Archive/metacom/metacom_1.5.1.tar.g
#library(metacom)
library(igraph)

par(mfrow = c(1,1))
par(mar = c(2,2,1,1))

# specify the number of replicates for each treatment
n_rep <- 3

#####################################################################################
# functions
modularity_quant <- function(net){
  temp <- computeModules(net)
  return(attributes(temp)$likelihood)
}

modularity_bin <- function(net){
  inc <- graph_from_incidence_matrix(net)
  inc <- cluster_walktrap(inc)
  temp <- igraph::modularity(inc)
  return(temp)
}


# Null distribution function for quantitative nestedness - calculates the network nestedness
net.null.nest.quant = function(nulls){
  net.null.metric <- list()
  for (i in 1:length(nulls)) {
    net.null.metric[[i]] = do.call('rbind',
                                   lapply(nulls[[i]], networklevel, index = 'weighted NODF'))
  }
  #names(net.null.metric) <- webs.names
  return(net.null.metric)
}

# Null distribution function for binary nestedness - calculates the network nestedness for ea
net.null.nest.bin = function(nulls){
  net.null.metric <- list()
  for (i in 1:length(nulls)) {
    net.null.metric[[i]] = do.call('rbind',
                                   lapply(nulls[[i]], networklevel, index = 'NODF'))
  }
  #names(net.null.metric) <- webs.names
  return(net.null.metric)
}

# Null distribution function for quantitative modularity - calculates the network modularity
net.null.mod.quant = function(nulls){
  net.null.metric <- list()
  for (i in 1:length(nulls)) {
    net.null.metric[[i]] = do.call('rbind',
                                   lapply(nulls[[i]], modularity_quant))
  }
  #names(net.null.metric) <- webs.names
  return(net.null.metric)
}

# Null distribution function for binary modularity - calculates the network modularity for ea
net.null.mod.bin = function(nulls){
  net.null.metric <- list()
  for (i in 1:length(nulls)) {
    net.null.metric[[i]] = do.call('rbind',
                                   lapply(nulls[[i]], modularity_bin))
  }
  #names(net.null.metric) <- webs.names
  return(net.null.metric)
}
```

```r
net.zscore = function(obsval, nullval) {
  (obsval - mean(nullval))/sd(nullval)
}

# Function that perform z-score calculation of nestedness using the observed and null networl
nest.quant.zscore = function(obs, nulltype){
  net.nest.zscore <- list()
  for(i in 1:length(obs)){
    net.nest.zscore[[i]] = net.zscore(obs[[i]]['weighted NODF'],
                                      nulltype[[i]][ ,'weighted NODF'])
  }
  #names(net.nest.zscore) <- webs.names
  return(net.nest.zscore)
}

# Function that perform z-score calculation of nestedness using the observed and null networl
nest.bin.zscore = function(obs, nulltype){
  net.nest.zscore <- list()
  for(i in 1:length(obs)){
    net.nest.zscore[[i]] = net.zscore(obs[[i]]['NODF'],
                                      nulltype[[i]][ ,'NODF'])
  }
  #names(net.nest.zscore) <- webs.names
  return(net.nest.zscore)
}


# Function that perform z-score calculation of modularity using the observed and null networl
mod.quant.zscore = function(obs, nulltype){
  net.mod.zscore <- list()
  for(i in 1:length(obs)){
    net.mod.zscore[[i]] = net.zscore(obs[[i]],
                                      nulltype[[i]])
  }
  #names(net.nest.zscore) <- webs.names
  return(net.mod.zscore)
}

# (identical to the function above, but with a different name for convenience) Function that
mod.bin.zscore = function(obs, nulltype){
  net.mod.zscore <- list()
  for(i in 1:length(obs)){
    net.mod.zscore[[i]] = net.zscore(obs[[i]],
                                     nulltype[[i]])
  }
  #names(net.nest.zscore) <- webs.names
  return(net.mod.zscore)
}




#####################################################################################3
# print("Plotting nestedness for quantitative networks")
# # plotting nestedness for quantitative networks
# nest_sim <- NULL
# for (sim_id in sim_names){
#   print(sim_id)
#   for (rep_id in 1:n_rep) {
#
#     webs_all <- list.load(paste(output_dir, '/', sim_id, '/', rep_id, '/networks.rds', sep
#     webs <- webs_all
#
#     # if the network is too small for nestness calculation
#     while(nrow(webs[[length(webs)]]) == 1 || ncol(webs[[length(webs)]]) == 1){
#       webs <- webs[-length(webs)]
#     }
#
#     # for testing
#     print(length(webs))
#
#     webs <- webs[length(webs)]
#
#     # Calculate network metric nestedness for all generations
#     net.metrics.nest.quant <- lapply(webs, networklevel, index = 'weighted NODF')
#
#     # create nulls for nestedness
#     net.nulls.r2d <- lapply(webs, nullmodel, method = "r2dtable", N = 100)
#     # calculate nestedness for the null
#     r2d.nest <- net.null.nest.quant(net.nulls.r2d)
#     # calculate z scores
#     r2d.nest.zscore <- nest.quant.zscore(net.metrics.nest.quant, r2d.nest)
#
#     nest_sim <- c(nest_sim, mean(unlist(r2d.nest.zscore)))
#   }
# }
#
# nam <- rep(NA, length(sim_names) * n_rep)
# for ( rep_id in 1:n_rep) {
#   nam[(1:length(sim_names)) * n_rep - (n_rep - rep_id)] <- sim_names
# }
# nest <- nest_sim
# write.csv(nest, paste(output_dir, "/nest_quant.csv", sep = ''))
# data <- data.frame(nam, nest)
# data$nam <- factor(data$nam, sim_names)
# #head(data)
#
#
# # Print the plot to a pdf file
# p <- ggplot(data, aes(x = nam, y = nest)) +
#   geom_boxplot() +
#   coord_flip()
#
# pdf(paste(output_dir, "/nestedness_quant.pdf", sep = ''), width = 5, height = 25) #
# print(p)
# dev.off()




##################################################################################
print("Plotting modularity for quantitative networks")
# plotting modularity for quantitative networks
mod_sim <- NULL
for (sim_id in sim_names){
  print(sim_id)
  for (rep_id in 1:n_rep) {

    webs_all <- list.load(paste(output_dir, '/', sim_id, '/', rep_id, '/networks.rds', sep =
    webs <- webs_all

    # if the network is too small for nestness calculation, or if there is only one host spe
    while(nrow(webs[[length(webs)]]) == 1 || ncol(webs[[length(webs)]]) == 1 || sum(rowSums(
      webs <- webs[-length(webs)]
    }

    # for testing
    print(length(webs))

    webs <- webs[length(webs)]

    # Calculate network metric modularity Q for all generations
    net.metrics.mod.quant <- rep(NA, length(webs))
    for (gen_id in 1:length(webs)) {
      #print(gen_id)
      net <- webs[[gen_id]]

      temp <- modularity_quant(net)
      net.metrics.mod.quant[gen_id] <- temp
    }

    # create nulls for modularity
    net.nulls.r2d <- lapply(webs, nullmodel, method = "r2dtable", N = 100)
    # calculate nestedness for the null
    r2d.mod <- net.null.mod.quant(net.nulls.r2d)
    # calculate z scores
    r2d.mod.zscore <- mod.quant.zscore(net.metrics.mod.quant, r2d.mod)

    mod_sim <-  c(mod_sim, mean(unlist(r2d.mod.zscore), na.rm = T))
  }
}


nam <- rep(NA, length(sim_names) * n_rep)
for ( rep_id in 1:n_rep) {
  nam[(1:length(sim_names)) * n_rep - (n_rep - rep_id)] <- sim_names
}
mod <- mod_sim
write.csv(mod, paste(output_dir, "/mod_quant.csv", sep = ''))
data <- data.frame(nam, mod)
data$nam <- factor(data$nam, sim_names)

# Print the plot to a pdf file
p <- ggplot(data, aes(x = nam, y = mod)) +
  geom_boxplot() +
  coord_flip()

pdf(paste(output_dir, "/modularity_quant.pdf", sep = ''), width = 5, height = 25)
```

```
print(p)
dev.off()



###################################################################################3
# print("Plotting nestedness for binary networks")
# # plotting nestedness for binary networks
# nest_sim <- NULL
# for (sim_id in sim_names){
#   print(sim_id)
#   for (rep_id in 1:n_rep) {
#
#     webs_all <- list.load(paste(output_dir, '/', sim_id, '/', rep_id, '/networks.rds', sep
#     webs <- webs_all
#     nest_bin <- NA
#
#     while(is.na(nest_bin)){ # non-NA values have not been reached yet
#
#       # if the network is too small for nestness calculation
#       while(nrow(webs[[length(webs)]]) == 1 || ncol(webs[[length(webs)]]) == 1){
#         webs <- webs[-length(webs)]
#       }
#
#       webs_sub <- webs[length(webs)] # take a subset of networks, usually the last few gene
#
#       # Calculate network metric nestedness for all generations
#       net.metrics.nest.bin <- lapply(webs_sub, networklevel, index = 'NODF')
#
#       # create nulls for nestedness
#       net.nulls.r2d <- lapply(webs_sub, nullmodel, method = "shuffle.web", N = 100)
#       # calculate nestedness for the null
#       r2d.nest <- net.null.nest.bin(net.nulls.r2d)
#       # calculate z scores
#       r2d.nest.zscore <- nest.bin.zscore(net.metrics.nest.bin, r2d.nest)
#       nest_bin <- mean(unlist(r2d.nest.zscore))
#
#       webs <- webs[-length(webs)]
#
#     }
#
#     nest_sim <- c(nest_sim, nest_bin)
```

```
#
#     # for testing
#     print(length(webs) + 1)
#   }
# }
#
# nam <- rep(NA, length(sim_names) * n_rep)
# for ( rep_id in 1:n_rep) {
#   nam[(1:length(sim_names)) * n_rep - (n_rep - rep_id)] <- sim_names
# }
# nest <- nest_sim
# write.csv(nest, paste(output_dir, "/nest_bin.csv", sep = ''))
# data <- data.frame(nam, nest)
# data$nam <- factor(data$nam, sim_names)
# #head(data)
#
#
# # Print the plot to a pdf file
# p <- ggplot(data, aes(x = nam, y = nest)) +
#   geom_boxplot() +
#   coord_flip()
#
# pdf(paste(output_dir, "/nestedness_bin.pdf", sep = ''), width = 5, height = 25) #
# print(p)
# dev.off()



##################################################################################
# print("Plotting modularity for binary networks")
# # plotting modularity for binary networks
# mod_sim <- NULL
# for (sim_id in sim_names){
#   print(sim_id)
#   for (rep_id in 1:n_rep) {
#
#     webs_all <- list.load(paste(output_dir, '/', sim_id, '/', rep_id, '/networks.rds', sep
#     webs <- webs_all
#     mod_bin <- NA
#
#     while(is.na(mod_bin)){ # non-NA values have not been reached yet
#
#       # if the network is too small for nestness calculation
#       while(nrow(webs[[length(webs)]]) == 1 || ncol(webs[[length(webs)]]) == 1){
#         webs <- webs[-length(webs)]
#       }
#
#       webs_sub <- webs[length(webs)] # take a subset of networks, usually the last few gen
#
#       # Calculate network metric modularity Q for all generations
#       net.metrics.mod.bin <- rep(NA, length(webs_sub))
#       for (gen_id in 1:length(webs_sub)) {
#         #print(gen_id)
#         net <- webs_sub[[gen_id]]
#         temp <- modularity_bin(net)
#         net.metrics.mod.bin[gen_id] <- temp
#       }
#
#       # create nulls for modularity
#       net.nulls.r2d <- lapply(webs_sub, nullmodel, method = "shuffle.web", N = 100)
#       # calculate nestedness for the null
#       r2d.mod <- net.null.mod.bin(net.nulls.r2d)
#       # calculate z scores
#       r2d.mod.zscore <- mod.bin.zscore(net.metrics.mod.bin, r2d.mod)
#       mod_bin <- mean(unlist(r2d.mod.zscore))
#
#       webs <- webs[-length(webs)]
#
#     }
#
#     mod_sim <- c(mod_sim, mod_bin)
#
#     # for testing
#     print(length(webs) + 1)
#   }
# }
#
#
# nam <- rep(NA, length(sim_names) * n_rep)
# for ( rep_id in 1:n_rep) {
#   nam[(1:length(sim_names)) * n_rep - (n_rep - rep_id)] <- sim_names
# }
# mod <- mod_sim
# write.csv(mod, paste(output_dir, "/mod_bin.csv", sep = ''))
# data <- data.frame(nam, mod)
# data$nam <- factor(data$nam, sim_names)
#
# # Print the plot to a pdf file
# p <- ggplot(data, aes(x = nam, y = mod)) +
#   geom_boxplot() +
#   coord_flip()
#
# pdf(paste(output_dir, "/modularity_bin.pdf", sep = ''), width = 5, height = 25)
# print(p)
# dev.off()


#####################################################################################
# visualize individual webs
# gen_id <- 500
# visweb(webs[[gen_id]])
# plotweb(webs[[gen_id]], text.rot=90, col.low = "green", col.high = "blue")
#
# # creating network object for plotting
# net <- network(webs[[gen_id]], matrix.type = "bipartite",
#                directed = FALSE, ignore.eval = FALSE)
# # plotting
# col = c("actor" = "tomato", "event" = "steelblue")
#
# ggnet2(net, node.size = 5, color = "mode", palette = col, edge.color = "gray70",
#        edge.size = 1, mode = "kamadakawai",
#        label = F, label.color = "white", label.size = 1,
#        shape = c(rep(15, nrow(webs[[gen_id]])), rep(16, ncol(webs[[gen_id]]))))
```

Phylogeny-network summary statistics represented by the Mantel correlation (a measure of partner conservatism).

```
library(ape)
#install.packages("~/Desktop/model_rcpp_hpc_OneDrive/R_local_sources/phangorn_2.7.0.tar.gz",
#install.packages("phytools")
library(phytools)
#install.packages("geiger")
library(geiger)
library(rlist)
library(nlme)
#install.packages('visreg')
```

```r
library(visreg)
#install.packages("GUniFrac")
library(GUniFrac)
#install.packages("picante")
library(picante)
library(vegan)
library(scales)

sim_nam <- "aa0000FFF" # this can be anything
source('basic_functions.R')

# specify the number of replicates and the total number of generations simulated
n_rep <- 3
n_gen <- 1000

# initializae the vectors for recording calculated values
host_part_cons_w <- rep(NA, n_rep * length(sim_names))
non_part_cons_w <- rep(NA, n_rep * length(sim_names))

host_part_cons_u <- rep(NA, n_rep * length(sim_names))
non_part_cons_u <- rep(NA, n_rep * length(sim_names))

# start calculation
for (sim_pos in 1:length(sim_names)){
  sim_id <- sim_names[sim_pos]
  print(sim_names[sim_pos])

  for (rep_id in 1:n_rep) {

    # decide which generation to look at (the network must not be too small, i.e., nrow > 1 a
    gen_id <- n_gen
    networks <- list.load( paste(output_dir, '/', sim_id, "/", rep_id, "/networks.rds", sep =

    # show progress
    print(percent( ((sim_pos - 1) * n_rep + rep_id)/(length(sim_names) * n_rep), accuracy = 1

    # loop this until non-NA values are reached
    host_r_w <- NA
    non_r_w <- NA

    host_r_u <- NA
    non_r_u <- NA
    while(is.na(host_r_w) || is.na(non_r_w) || is.na(host_r_u) || is.na(non_r_u)){ # non-NA v

      # if the network is too small for nestness calculation
      while(nrow(networks[[gen_id]]) < 3 || ncol(networks[[gen_id]]) < 3){ # at least 3 cols/
        gen_id <- gen_id - 1
      }

        tree_host <- read.tree( paste(output_dir, '/', sim_id, "/", rep_id, "/host_phy", sep
        # assign(paste( "tree_host_", "full", sep = ''), tree_host)
        pop_spec_host <- list.load( paste(output_dir, '/', sim_id, "/", rep_id, "/pop_spec_ho

        tree_non <- read.tree( paste(output_dir, '/', sim_id, "/", rep_id, "/non_phy", sep =
        # assign(paste( "tree_non_", "full", sep = ''), tree_non)
        pop_spec_non <- list.load( paste(output_dir, '/', sim_id, "/", rep_id, "/pop_spec_non

        # trim the tree to the generation of interest
        if(gen_id != n_gen){
          for(prune_gen in n_gen:(gen_id + 1)){
            # get the number of species at this generations
            n_spec_host <- length( pop_spec_host[[prune_gen]]$host_spec_list )
            n_spec_non <- length( pop_spec_non[[prune_gen]]$non_spec_list )

            # create a vector for the taxon names to be removed
            rm_host <- paste ('_', as.character(prune_gen), '.', as.character(1:n_spec_host)
            tree_host <- drop.tip(tree_host, tip = rm_host, trim.internal = F, rooted = T, co

            rm_non <- paste ('_', as.character(prune_gen), '.', as.character(1:n_spec_non),
            tree_non <- drop.tip(tree_non, tip = rm_non, trim.internal = F, rooted = T, colla
          }

          # assign(paste( "tree_host_", as.character(gen_id), sep = ''), tree_host)
          # assign(paste( "tree_non_", as.character(gen_id), sep = ''), tree_non)
        }

        # prune the tree to extant species only (remove all extinct lineages)
        n_spec_host <- length( pop_spec_host[[gen_id]]$host_spec_list )
        kp_host <- paste ('_', as.character(gen_id), '.', as.character(1:n_spec_host), '_', s
        tree_host <- drop.tip(tree_host, tip = tree_host$tip.label[-match(kp_host, tree_host$

        n_spec_non <- length( pop_spec_non[[gen_id]]$non_spec_list )
        kp_non <- paste ('_', as.character(gen_id), '.', as.character(1:n_spec_non), '_', sep
        tree_non <- drop.tip(tree_non, tip = tree_non$tip.label[-match(kp_non, tree_non$tip.l
        # read in the incidence matrix representing the interaction network
        networks <- list.load( paste(output_dir, '/', sim_id, "/", rep_id, "/networks.rds", s
        network <- networks[[gen_id]]
        rownames(network) <- paste('_', substring(rownames(network), 3), '_', sep = '')
        colnames(network) <- paste('_', substring(colnames(network), 3), '_', sep = '')

        network_host <- network
        network_non <- t(network)

        # calculate UniFrac distances for hosts, i.e., dissimilarity in interaction partners
        unifracs_host <- GUniFrac(network_host, tree_non, alpha=c(0, 0.5, 1))$unifracs
        dw_host <- unifracs_host[, , "d_1"]     # Weighted UniFrac
        du_host <- unifracs_host[, , "d_UW"]        # Unweighted UniFrac
        dv_host <- unifracs_host[, , "d_VAW"]       # Variance adjusted weighted UniFrac
        d0_host <- unifracs_host[, , "d_0"]         # GUniFrac with alpha 0
        d5_host <- unifracs_host[, , "d_0.5"]       # GUniFrac with alpha 0.5

        # calculate UniFrac distances for non-hosts, i.e., dissimilarity in interaction partr
        unifracs_non <- GUniFrac(network_non, tree_host, alpha=c(0, 0.5, 1))$unifracs
        dw_non <- unifracs_non[, , "d_1"]       # Weighted UniFrac
        du_non <- unifracs_non[, , "d_UW"]      # Unweighted UniFrac
        dv_non <- unifracs_non[, , "d_VAW"]     # Variance adjusted weighted UniFrac
        d0_non <- unifracs_non[, , "d_0"]       # GUniFrac with alpha 0
        d5_non <- unifracs_non[, , "d_0.5"]     # GUniFrac with alpha 0.5

        # calculate phylogenetic distance between all pairs of hosts
        dist_host <- cophenetic.phylo(tree_host)
        temp1 <- dist_host[ match(rownames(network_host), rownames(dist_host)) , ]
        temp2 <- temp1[ , match(rownames(network_host), colnames(temp1)) ]
        dist_host <- temp2

        # calculate phylogenetic distance between all pairs of non-nons
        dist_non <- cophenetic.phylo(tree_non)
        temp1 <- dist_non[ match(rownames(network_non), rownames(dist_non)) , ]
        temp2 <- temp1[ , match(rownames(network_non), colnames(temp1)) ]
        dist_non <- temp2

        # Mantel tests for hosts and non-hosts
        # for hosts, some columns in dw_host may be NA because that host species does not par
        host_wo_non <- which(rowSums(network_host) == 0)
        if(length(host_wo_non) > 0){
          print(paste("host species", as.character(host_wo_non), "does not partner with any r
          dw_host <- dw_host[-host_wo_non, ]
          dw_host <- dw_host[, -host_wo_non]
          du_host <- du_host[-host_wo_non, ]
          du_host <- du_host[, -host_wo_non]
          dist_host <- dist_host[-host_wo_non, ]
          dist_host <- dist_host[, -host_wo_non]
        }

        host_mantel_w <- mantel(dw_host, dist_host, method="pearson", permutations=1000)
        non_mantel_w <- mantel(dw_non, dist_non, method="pearson", permutations=1000)

        host_mantel_u <- mantel(du_host, dist_host, method="pearson", permutations=1000)
        non_mantel_u <- mantel(du_non, dist_non, method="pearson", permutations=1000)

        host_r_w <- host_mantel_w$statistic
        non_r_w <- non_mantel_w$statistic

        host_r_u <- host_mantel_u$statistic
        non_r_u <- non_mantel_u$statistic

        gen_id <- gen_id - 1
    }


    host_part_cons_w[(sim_pos - 1) * n_rep + rep_id] <- host_r_w
    non_part_cons_w[(sim_pos - 1) * n_rep + rep_id] <- non_r_w

    host_part_cons_u[(sim_pos - 1) * n_rep + rep_id] <- host_r_u
    non_part_cons_u[(sim_pos - 1) * n_rep + rep_id] <- non_r_u

    # for testing
    print(gen_id + 1)

  }
}

part_cons <- data.frame(host_part_cons_w=host_part_cons_w, non_part_cons_w=non_part_cons_w, h
write.csv(part_cons, paste(output_dir, "/part_cons.csv", sep = ''))
```