Using sabinaHSBM for link prediction and network reconstruction using Hierarchical Stochastic Block Models

Example on how to use it to identify missing and spurious links with the full\_reconstruction method (with parallelized computation)

The *sabinaHSBM* package provides tools for link prediction and network reconstruction using hierarchical stochastic block models (HSBM). This document demonstrates a simple use for the full\_reconstruction method, showcasing how it handles missing and spurious links in an incomplete and error-prone network. To enhance computational efficiency during the prediction links step we leverage parallelized computation.

We use the example dataset dat2 included in the package to show key functionalities, including data preparation, link prediction, and network reconstruction.

**Important note**

The *sabinaHSBM* package depends on various R and Python libraries, as well as system-level components — especially the graph-tool Python package.

* **UNIX users** can run *sabinaHSBM* natively if their system includes:
  + R (version ≥ 4.3.3) with all required R packages (listed below)
  + Python ≥ 3.12 with the graph-tool library (version 2.59)
* For all other users — or to avoid configuration issues — a **ready-to-use Docker image** is available.  
  It includes all dependencies pre-installed, including the *sabinaHSBM* package.  
  See *Supporting Information S3* for instructions on how to use

# Loading Required Libraries

After ensuring the required system dependencies are available (see note above), we begin by loading the required R packages. Missing packages are automatically installed.

# If the package is not installed, install it from GitHub  
# if (!requireNamespace("sabinaHSBM", quietly = TRUE)) {  
# library(remotes)  
# remotes::install\_github("h-lima/sabinaHSBM")  
# }  
  
# Load the sabinaHSBM package  
#library(sabinaHSBM)  
devtools::load\_all("/home/project\_hsbm/graph\_tool/sabinaHSBM/")  
#devtools::load\_all("/media/sf\_UAM\_NexTdive/graph\_tool/sabinaHSBM/")

If the required packages are not already installed, install them from CRAN

list.of.packages <- c(  
 "dplyr",  
 "parallel",  
 "reshape2",  
 "reticulate",  
 "stringr",  
 "tidyr",  
 "ROCR",  
 "data.table"  
)  
  
new.packages <- list.of.packages[!(list.of.packages %in% installed.packages()[, "Package"])]  
if (length(new.packages) > 0) {  
 install.packages(new.packages, dependencies = TRUE)  
}  
  
for (package in list.of.packages) {  
 library(package, character.only = TRUE)  
}

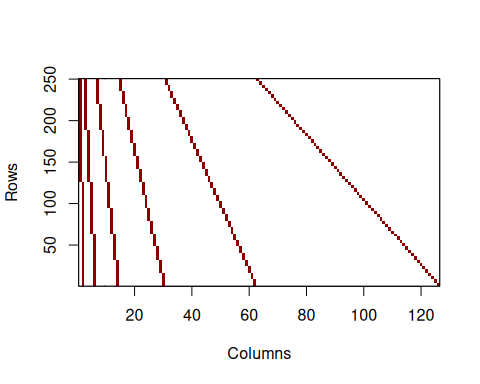
# Record starting time (optional)  
start\_time <- Sys.time()

# Load data

The dataset dat2 is a binary bipartite matrix representing a hypothetical species interactions network. Columns and rows correspond to two different types of nodes (e.g., hosts and parasites), and links (values of 1, in red) represent interaction between them while 0 (in black) represent lack of observed interaction. These links are distributed in a structured pattern, with no missing information, to focus the example on identifying spurious links.

# Load the dataset  
data(dat2, package = "sabinaHSBM")  
dat<- dat2

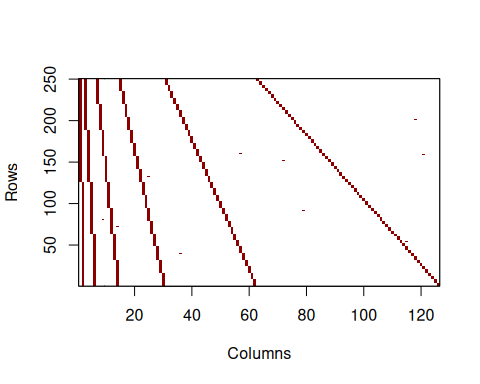
# Plot the original matrix  
plot\_interaction\_matrix(adj\_mat = dat, order\_mat = FALSE)



To simulate spurious links and address the full\_reconstruction method, we randomly add a small number of false positives (1) to the matrix. This controlled modification allows us to demonstrate how the method identifies both spurious links (observed but potentially erroneous interactions, false positives) and missing links (unobserved but likely interactions, false negatives).

# Add spurious links artificially  
set.seed(123)  
num\_spurious <- ceiling(sum(dat) \* 0.01) # Proportion of spurious links to add  
spurious\_links <- matrix(nrow = num\_spurious, ncol = 2)  
  
for (i in 1:num\_spurious) {  
 repeat {  
 random\_row <- sample(1:nrow(dat), 1)  
 random\_col <- sample(1:ncol(dat), 1)  
   
 if (dat[random\_row, random\_col] == 0) {  
 dat[random\_row, random\_col] <- 1  
 spurious\_links[i,] <- c(random\_row, random\_col)  
 break  
 }  
 }  
 }

# Plot the matrix with spurious  
plot\_interaction\_matrix(dat, order\_mat = FALSE)



# Preparing Input data for HSBM

The hsbm.input function pre-processes the dataset, creating cross-validation folds and edge lists required for modeling. Here, we use 5-fold cross-validation and the full\_reconstruction method.

# Prepare input data  
n\_folds <- 5 # Number of folds for cross-validation  
  
myInput <- hsbm.input(  
 dat, # Binary bipartite matrix of observed links  
 n\_folds = n\_folds,  
 add\_spurious = FALSE # Simulate spurious links  
)

## [1] "Actual cross-validation rate is 0.200"  
## [2] "Actual cross-validation rate is 0.200"  
## [3] "Actual cross-validation rate is 0.200"  
## [4] "Actual cross-validation rate is 0.200"  
## [5] "Actual cross-validation rate is 0.200"

For the purposes of this example, artificial spurious links are labeled with the “spurious\_edge” tag using the code below. This labeling allows for tracking these links throughout the analysis and network reconstruction process, although in typical use, such labeling would not be necessary.

myInput$edgelists <- lapply(myInput$edgelists,  
 function(x){  
 x$x <- 1  
 return(x)  
 })  
  
for(i in 1:length(myInput$edgelists)){  
 el <- myInput$edgelists[[i]]  
 rows <- as.numeric(el[, 1]) + 1  
 cols <- as.numeric(el[, 2]) - nrow(myInput$data) + 1  
 rows\_cols <- paste0(rows, "\_", cols)  
 spurious\_rows\_cols <- paste0(spurious\_links[, 1], "\_", spurious\_links[, 2])  
 spurious\_loc <- which(rows\_cols %in% spurious\_rows\_cols)  
 el[spurious\_loc, ]$x <- 1  
 el[spurious\_loc, ]$edge\_type <- "spurious\_edge"  
 myInput$edgelists[[i]] <- el  
}

# Summarizes network characteristics  
summary(myInput)

## n\_rows n\_cols n\_links rows\_single\_link cols\_single\_link possible\_links  
## 1 250 126 1515 0 0 31500

# Predict link probabilities

The hsbm.predict function applies the HSBM to predict probabilities of all links (observed and unobserved links). This step is crucial to identify spurious and missing links within the data, which are often present in incomplete or error-prone networks. The function works directly with the processed input created by hsbm.input. This step can be computationally intensive when working with large datasets or numerous folds. To improve performance, we use parallelized computation, distributing tasks across multiple cores.

# Define the number of cores to use  
#nCores <- detectCores() - 1  
nCores <- 2  
  
# Generate HSBM predictions  
myPred <- hsbm.predict(  
 myInput, # Input data processed by hsbm.input()  
 iter = 10000, # Number of iterations  
 wait= 10000, # Number of iterations for MCMC equilibration  
 method = "full\_reconstruction", # Method for link prediction  
 save\_blocks = TRUE, # Save group assignments   
 save\_pickle = FALSE, # Save results as pickle files,  
 save\_plots = FALSE, # Save hierarchical edge bundling plots  
 n\_cores = nCores  
)

Predicted link probabilities and group assignments are stored for each fold. Below, we extract the link probabilities (p) and groups assignments of links for fold 1.

# View probabilities for fold 1  
probabilities\_fold1 <- myPred$probs[[1]]  
head(probabilities\_fold1)

## v1 v2 p v1\_names v2\_names edge\_type  
## 1 0 250 1.0000000 sp1 SPa documented  
## 2 0 252 1.0000000 sp1 SPc documented  
## 3 0 256 1.0000000 sp1 SPg documented  
## 4 0 264 0.9983998 sp1 SPo documented  
## 5 0 280 1.0000000 sp1 SPeb documented  
## 6 0 312 1.0000000 sp1 SPkc documented

The group assignments provide the hierarchical clustering structure of nodes for each fold. Let’s extract and examine the group assignments for fold 1:

# View the group/block assignments for fold 1  
groups\_fold1 <- myPred$groups[[1]]  
  
# Filter one type of nodes (e.g., hosts, in columns)  
vnames <- colnames(myPred$data)  
groups\_fold1 <- groups\_fold1 %>% filter(names %in% vnames)  
g\_cols <- grep("^G", names(groups\_fold1))  
groups\_fold1[g\_cols] <- lapply(groups\_fold1[g\_cols], sort)  
  
print(head(groups\_fold1))

## nodes G1 G2 G3 G4 G5 names  
## 1 250 12 7 1 4 2 SPa  
## 2 251 12 7 1 4 2 SPb  
## 3 252 12 7 1 4 2 SPc  
## 4 253 12 7 1 4 2 SPd  
## 5 254 12 7 1 4 2 SPe  
## 6 255 12 7 1 4 2 SPf

Hierarchical group assignments provide insight into how nodes (e.g., hosts) are organized across multiple levels. At the first level (G1) nodes are divided into specific groups, reflecting fine-scale patterns. Moving to higher levels (G2, G3, G4) these groups (or communities) are progressively aggregated, revealing broader patterns and relationships or communities.

Next, we will explore how the model assigns specific probabilities to different types of links (“documented,” “held out,” and “spurious\_edge”) for each fold to evaluate HSBM ability to correctly identify them. Documented links, which are real observations (true positives), should have probabilities close to 1. Held-out links, which are real observations but transformed to 0s during training for validation, should also show high probabilities. In contrast, spurious edges, artificially generated (initially unobserved links transformed to 1s), should have probabilities close to 0, as the model should recognize them as false links (false negatives).

for(i in 1:5){  
 cat("Fold ", i, "\n")  
 cat("\tDocumented\n\t")  
 print(dplyr::filter(myPred$probs[[i]], edge\_type == "documented", p < 0.95))  
 cat("\tHeld out\n\t")  
 print(dplyr::filter(myPred$probs[[i]], edge\_type == "held\_out", p < 0.95))  
 cat("\tSpurious edge\n\t")  
 print(dplyr::filter(myPred$probs[[i]], edge\_type == "spurious\_edge", p < 0.95))  
}

## Fold 1   
## Documented  
## [1] v1 v2 p v1\_names v2\_names edge\_type  
## <0 rows> (or 0-length row.names)  
## Held out  
## [1] v1 v2 p v1\_names v2\_names edge\_type  
## <0 rows> (or 0-length row.names)  
## Spurious edge  
## v1 v2 p v1\_names v2\_names edge\_type  
## 1 49 367 0.5955596 sp50 SPne spurious\_edge  
## 2 90 306 0.6180618 sp91 SPec spurious\_edge  
## 3 98 321 0.7212721 sp99 SPtc spurious\_edge  
## 4 117 274 0.6987699 sp118 SPy spurious\_edge  
## 5 136 342 0.2120212 sp137 SPod spurious\_edge  
## 6 194 291 0.8047805 sp195 SPpb spurious\_edge  
## 7 42 350 0.0000000 sp43 SPwd spurious\_edge  
## 8 91 370 0.0000000 sp92 SPqe spurious\_edge  
## 9 210 285 0.0000000 sp211 SPjb spurious\_edge  
## Fold 2   
## Documented  
## [1] v1 v2 p v1\_names v2\_names edge\_type  
## <0 rows> (or 0-length row.names)  
## Held out  
## [1] v1 v2 p v1\_names v2\_names edge\_type  
## <0 rows> (or 0-length row.names)  
## Spurious edge  
## v1 v2 p v1\_names v2\_names edge\_type  
## 1 49 367 0.57245725 sp50 SPne spurious\_edge  
## 2 89 340 0.35343534 sp90 SPmd spurious\_edge  
## 3 90 306 0.68296830 sp91 SPec spurious\_edge  
## 4 117 274 0.34583458 sp118 SPy spurious\_edge  
## 5 194 291 0.05480548 sp195 SPpb spurious\_edge  
## 6 42 350 0.00000000 sp43 SPwd spurious\_edge  
## 7 91 370 0.00000000 sp92 SPqe spurious\_edge  
## 8 98 321 0.00000000 sp99 SPtc spurious\_edge  
## 9 136 342 0.00000000 sp137 SPod spurious\_edge  
## Fold 3   
## Documented  
## v1 v2 p v1\_names v2\_names edge\_type  
## 1 217 367 0.9435944 sp218 SPne documented  
## Held out  
## [1] v1 v2 p v1\_names v2\_names edge\_type  
## <0 rows> (or 0-length row.names)  
## Spurious edge  
## v1 v2 p v1\_names v2\_names edge\_type  
## 1 136 342 0.7690769 sp137 SPod spurious\_edge  
## 2 42 350 0.0000000 sp43 SPwd spurious\_edge  
## 3 98 321 0.0000000 sp99 SPtc spurious\_edge  
## 4 210 285 0.0000000 sp211 SPjb spurious\_edge  
## Fold 4   
## Documented  
## [1] v1 v2 p v1\_names v2\_names edge\_type  
## <0 rows> (or 0-length row.names)  
## Held out  
## [1] v1 v2 p v1\_names v2\_names edge\_type  
## <0 rows> (or 0-length row.names)  
## Spurious edge  
## v1 v2 p v1\_names v2\_names edge\_type  
## 1 98 321 0.06020602 sp99 SPtc spurious\_edge  
## 2 136 342 0.69126913 sp137 SPod spurious\_edge  
## 3 169 258 0.93499350 sp170 SPi spurious\_edge  
## 4 178 263 0.82748275 sp179 SPn spurious\_edge  
## 5 196 364 0.58315832 sp197 SPke spurious\_edge  
## 6 49 367 0.00000000 sp50 SPne spurious\_edge  
## 7 90 306 0.00000000 sp91 SPec spurious\_edge  
## 8 91 370 0.00000000 sp92 SPqe spurious\_edge  
## 9 210 285 0.00000000 sp211 SPjb spurious\_edge  
## Fold 5   
## Documented  
## [1] v1 v2 p v1\_names v2\_names edge\_type  
## <0 rows> (or 0-length row.names)  
## Held out  
## [1] v1 v2 p v1\_names v2\_names edge\_type  
## <0 rows> (or 0-length row.names)  
## Spurious edge  
## v1 v2 p v1\_names v2\_names edge\_type  
## 1 49 367 0.02810281 sp50 SPne spurious\_edge  
## 2 90 306 0.64596460 sp91 SPec spurious\_edge  
## 3 98 321 0.25072507 sp99 SPtc spurious\_edge  
## 4 117 274 0.61286129 sp118 SPy spurious\_edge  
## 5 158 328 0.47844784 sp159 SPad spurious\_edge  
## 6 169 258 0.86578658 sp170 SPi spurious\_edge  
## 7 178 263 0.94489449 sp179 SPn spurious\_edge  
## 8 194 291 0.62226223 sp195 SPpb spurious\_edge  
## 9 42 350 0.00000000 sp43 SPwd spurious\_edge  
## 10 91 370 0.00000000 sp92 SPqe spurious\_edge  
## 11 136 342 0.00000000 sp137 SPod spurious\_edge  
## 12 210 285 0.00000000 sp211 SPjb spurious\_edge

# Network Reconstruction

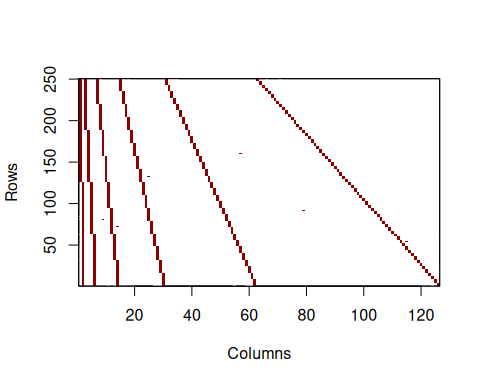
The hsbm.reconstructed function generates a reconstructed binary interaction matrix by combining predictions from all folds. The predicted matrix is binarized using a user-specified threshold.

# Network reconstruction  
myReconst <- hsbm.reconstructed(  
 myPred, # Predictions processed by hsbm.predict  
 rm\_documented=FALSE, # Remove documented links during validation  
 na\_treatment = "ignore\_na", # Handle NA values in predictions  
 threshold = 0.5, # Binarization threshold  
 new\_matrix\_method = "average\_thresholded", # Combine fold predictions  
 spurious\_edges = TRUE,  
)

## Warning in get\_hsbm\_results(hsbm\_out, input\_names = TRUE, na\_treatment =  
## na\_treatment): Predictions obtained for 13.61% of the links. Consider  
## increasing the number of iterations.

This output includes the final averaged probability matrix, the final reconstructed binary matrix, and evaluation metrics. These results highlight the predicted interactions, showcasing the method’s ability to detect spurious and missing links effectively. Let’s explore some of the key outputs.

# Visualize the reconstructed binary matrix  
plot\_interaction\_matrix(myReconst$new\_mat, order\_mat = FALSE)



# View the reconstruction summary and evaluation metrics  
summary(myReconst)

## $`Reconstructed Network Metrics`  
## obs\_links unobs\_links pred\_links kept\_links spurious\_links missing\_links  
## 1 1515 29985 1509 1509 6 0  
##   
## $`Evaluation Metrics`  
## mean\_RLRR mean\_auc mean\_aucpr mean\_yPRC mean\_prec mean\_sens mean\_spec  
## 1 0.9973597 0.9987763 1.048951 0.04809524 1 0.9966997 1  
## mean\_ACC mean\_ERR mean\_tss  
## 1 0.9998413 0.0001587302 0.9966997

The summary provides the number of spurious and missing links, as well as key evaluation metrics, such as the retained link recovery rate (RLRR).

Let’s now examine the top 10 predicted links that are most likely to be spurious (false positives) by visualizing the probabilities of “documented” links.

# Visualize the top most likely spurious links  
top\_links\_spurious <- top\_links(myReconst,   
 n = 10,   
 edge\_type = "documented")  
print(top\_links\_spurious)

## v1\_names v2\_names p sd  
## 1 sp43 SPwd 0.2000000 0.4335897  
## 2 sp92 SPqe 0.2000000 0.4335897  
## 3 sp211 SPjb 0.2000000 0.4335897  
## 4 sp99 SPtc 0.2064406 0.3046635  
## 5 sp137 SPod 0.3344734 0.3242977  
## 6 sp50 SPne 0.4392239 0.3590875  
## 7 sp91 SPec 0.5893989 0.3622572  
## 8 sp195 SPpb 0.6963696 0.3894361  
## 9 sp118 SPy 0.7314931 0.2698057  
## 10 sp90 SPmd 0.8632263 0.2801915

This document demonstrates the use of the *sabinaHSBM* package for network reconstruction. By applying the full reconstruction method, we showcased how spurious and missing links can be effectively identified and addressed in complex networks.

# Computing characteristics

# Show processing time and computer characteristics (optional)  
end\_time <- Sys.time()  
cat("The processing time of this script took: ",  
 round(as.numeric(difftime(end\_time, start\_time, units = "mins", 1))), "minutes\n")

## The processing time of this script took: 97 minutes

This analysis was performed on a Dynabook with the following characteristics:

* Processor (CPU): Intel Core i7-1165G7 @ 2.80GHz (11th Gen)
* Memory (RAM): 32 GB
* Operating System: Windows 11 Pro, Version 24H2
* R Version: 4.3.3