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 steps we use 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:

There are **two ways** to use the package:

- **UNIX users (native installation)** can run *sabinaHSBM* locally **if** their system includes:
  - R (version $\leq$ 4.0.4) with all required R packages (listed below)
  - Python $\leq$ 3.9.2 with the `graph-tool` library (version $\leq$ 2.45)
- **All other users**, or those who prefer to avoid manual setup, can use the **ready-to-use Docker image**, which includes everything needed:
  - All R and Python dependencies
  - The *sabinaHSBM* package pre-installed and ready to use

Loading Required Libraries

Note:

The following instructions are **only required for UNIX users** running *sabinaHSBM natively* (outside Docker).

These users must ensure their system includes the required dependencies before proceeding.

```
# 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)
```

Install required R packages if not already available:

```
list.of.packages <- c(
  "dplyr",
  "parallel",
  "reshape2",
  "reticulate",
```

```r
  "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)
}
```

If you're using the **Docker image** (see *Supporting Information S3*) **you do not need to install any packages**, simply start the container and load the package in your R session with:

```r
library(sabinaHSBM)
```

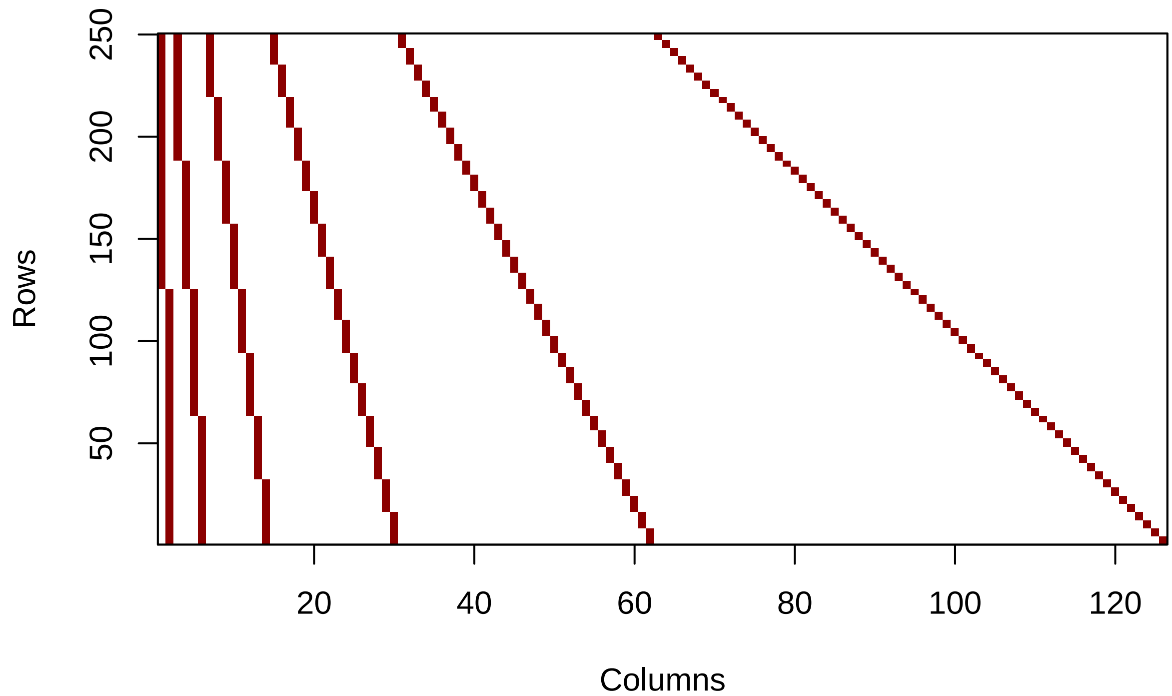
```r
# Record starting time (optional)
start_time <- Sys.time()
```

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 interactions between them while `0` (in black) represent lack of an observed interaction. These links are distributed in a structured pattern, with no missing information, to focus the example on identifying spurious links.

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

```r
# Plot the original matrix
plot_interaction_matrix(adj_mat = dat, order_mat = FALSE)
```



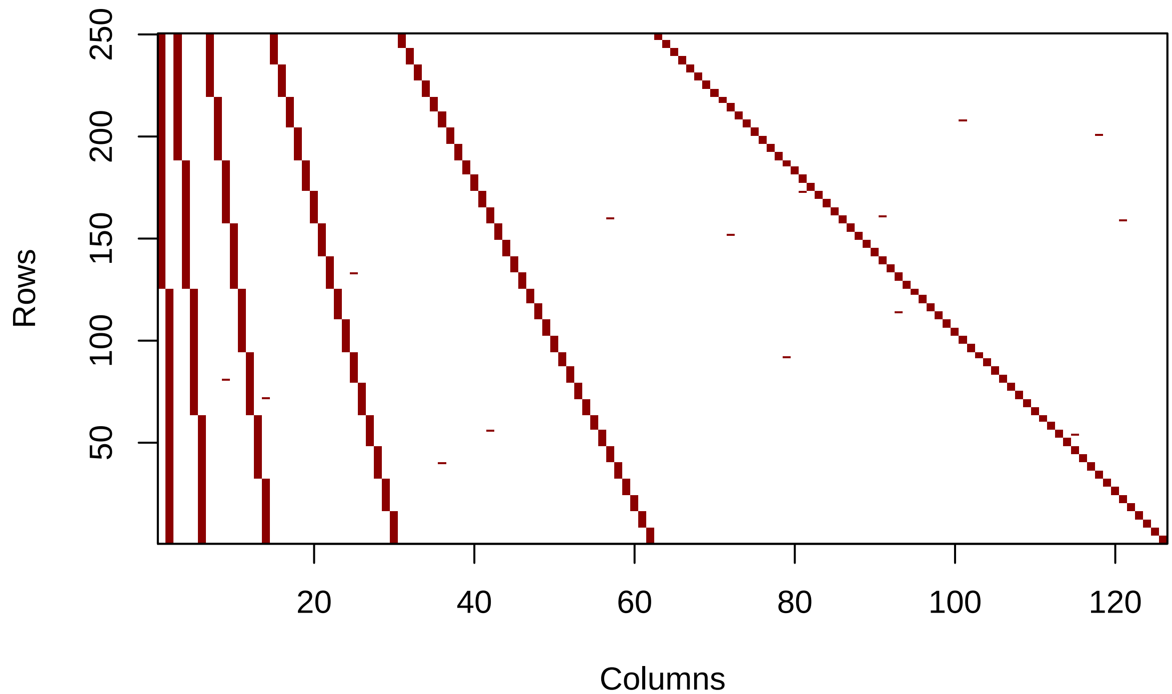
To simulate spurious links and address the `full_reconstruction` method, we randomly add a small number of false positives (1) to the matrix. This controlled modification allows us to demonstrate how the method identifies both spurious links (observed but potentially erroneous interactions, false positives)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

```r
                                  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
}
```

```r
# 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.

```r
# Define the number of cores to use
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
  rnd_seed = 123,       # Sets seed in python environment for reproducibility
  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.

```r
# View probabilities for fold 1
probabilities_fold1 <- myPred$probs[[1]]
head(probabilities_fold1)
```

```
##   v1  v2 p v1_names v2_names  edge_type
## 1  0 312 1      sp1     SPkc documented
## 2  0 280 1      sp1     SPeb documented
```

```
## 3   0 256 1       sp1       SPg documented
## 4   0 250 1       sp1       SPa documented
## 5   0 264 1       sp1       SPo documented
## 6   0 252 1       sp1       SPc 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:

```r
# View the group/block assignments for fold 1
groups_fold1 <- myPred$groups[[1]]

# Filter one type of nodes (nodes 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 G6 names
## 1   250 24  7  1  0  2  3   SPa
## 2   251 24  7  1  0  2  3   SPb
## 3   252 24  7  1  0  2  3   SPc
## 4   253 24  7  1  0  2  3   SPd
## 5   254 24  7  1  0  2  3   SPe
## 6   255 24  7  1  0  2  3   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 (unless it could be a missing link), as the model should recognize them as false links (false negatives).

```r
fold_res <- list()
for(i in 1:5){
    doc <- dplyr::filter(myPred$probs[[i]], edge_type == "documented")
    held <- dplyr::filter(myPred$probs[[i]], edge_type == "held_out")
    spur <- dplyr::filter(myPred$probs[[i]], edge_type == "spurious_edge")
    fold_data <- c(i,
                   nrow(doc), sum(doc$p < 0.95),
                   nrow(held), sum(held$p < 0.95),
                   nrow(spur), sum(spur$p < 0.95))
    fold_res[[i]] <- fold_data
}

res_summary <- do.call(rbind, fold_res)
colnames(res_summary) <- c("Fold", "Nr doc", "Doc p <0.95",
                           "Nr held", "Held p < 0.95",
                           "Nr spur", "Spur p < 0.95")
```

```r
kable(res_summary) # We use kable function from kable package to prettify output here
```

Fold	Nr doc	Doc p <0.95	Nr held	Held p < 0.95	Nr spur	Spur p < 0.95
1	1197	0	303	0	15	12
2	1199	0	301	0	15	8
3	1199	0	301	0	15	9
4	1204	0	296	0	15	7
5	1201	0	299	0	15	9

Network Reconstruction

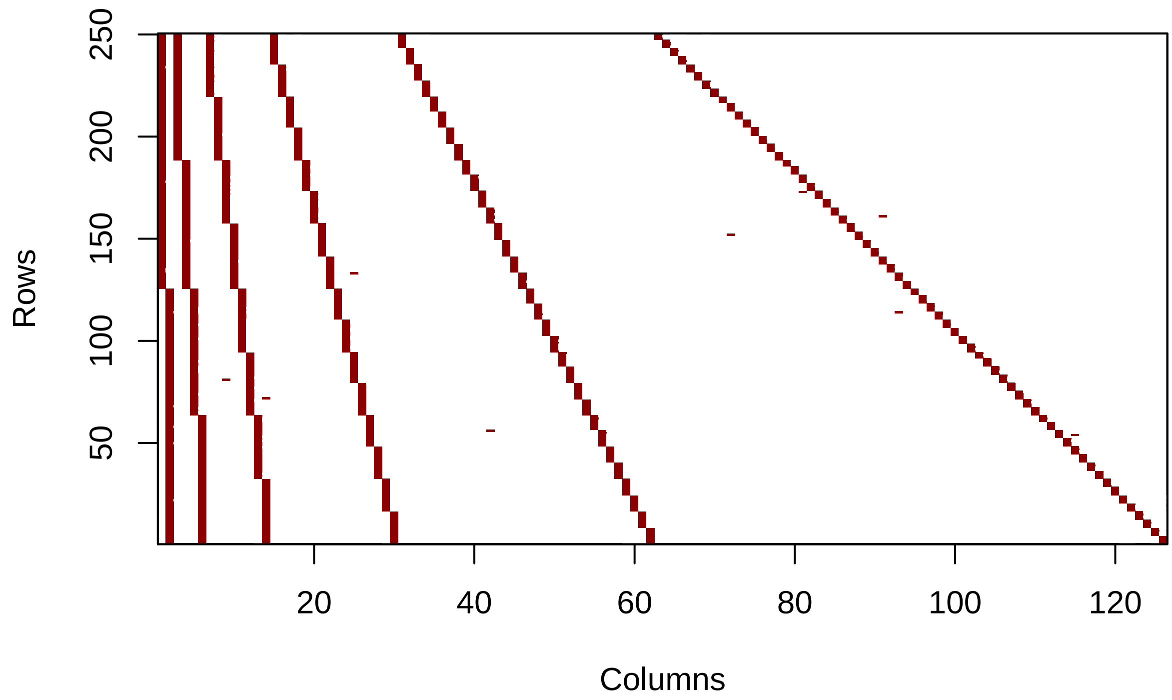
The `hsbm.reconstructed` function generates a reconstructed binary interaction matrix by combining predictions from all folds. The predicted matrix is transformed to binary values using a user-specified threshold.

```r
# 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 12.47% 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.

```r
# Visualize the reconstructed binary matrix
plot_interaction_matrix(myReconst$new_mat, order_mat = FALSE)
```




```r
# 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 0.9966997 0.9993878   1.007331 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.

```r
# 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      sp50     SPne 0.2478648 0.4068467
## 2      sp91     SPec 0.2593059 0.2586287
```

```
## 3      sp43     SPwd 0.2688669 0.1349554
## 4     sp159     SPad 0.2776678 0.1339075
## 5      sp92     SPqe 0.3441944 0.3660432
## 6     sp211     SPjb 0.3773177 0.3227095
## 7     sp137     SPod 0.5739374 0.1972130
## 8     sp118      SPy 0.6493649 0.3697671
## 9      sp99     SPtc 0.7076508 0.4069580
## 10    sp195     SPpb 0.7633563 0.2655090
```

This example was tailored to find spurious links and the model didn't identify any missing links at the threshold of 0.5 (but it did correctly identify 100% of the held-out links as missing). Nevertheless, we show how we can rank the most likely missing links according to the model.

```
# Visualize the top most likely spurious links
top_links_missing <- top_links(myReconst,
                               n = 10,
                               edge_type = "undocumented")
print(top_links_missing)
```

```
##    v1_names v2_names           p          sd
## 1     sp152      SPl 0.006500650 0.000000000
## 2     sp141      SPl 0.006000600 0.000000000
## 3     sp150      SPl 0.005800580 0.000000000
## 4     sp237     SPte 0.005200520 0.000000000
## 5     sp127     SPwb 0.005100510 0.000000000
## 6     sp155     SPqd 0.004600460 0.000000000
## 7     sp144      SPz 0.004500450 0.000000000
## 8      sp53      SPg 0.004300430 0.000000000
## 9      sp90     SPob 0.004025403 0.003458571
## 10      sp6     SPtc 0.004000400 0.000000000
```

If the task is to do network reconstruction by inserting the most plausible missing links, we can also use the probabilities from the `full_reconstruction` method as 'scores' in a supervised binary classification. This is achieved by setting a discrimination threshold algorithm in `hsbm.reconstructed()` as it is done in Supporting Information S1.

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:  64 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