Using *sabinaHSBM* for link prediction and network reconstruction with "conditional_missing" method

This guide shows a simple use case based on the "conditional_missing" method in *sabinaHSBM* to identify missing links (false negatives) in an incomplete network.

We use the artifically created dataset `dat` included in the package to show key functionalities, including data preparation, link prediction, and network reconstruction.

Requirements and installation

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

Option A: Docker

Non-native UNIX users or those who prefer to avoid manual dependency installation can use the provided ready-to-use Docker image which includes R, Python, all dependencies, and *sabinaHSBM* pre-installed (see Supporting Information S1 for Docker container setup).

Once the container is running, simply load the package in your R session with:

```
library(sabinaHSBM)
```

Option B: Native UNIX installation

Native UNIX users can run *sabinaHSBM* locally **if** their system includes:

- R (version $\geq$ 4.0.4) with all required R packages (listed below)
- Python $\geq$ 3.12.3 with the `graph-tool` library (version $\geq$ 2.45)

```
install.packages("remotes")
# If the package is not installed, install sabinaHSBM from GitHub
if (!requireNamespace("sabinaHSBM", quietly = TRUE)) {
    remotes::install_github("anonbuild/sabinaHSBM")
}

# Load sabinaHSBM
library(sabinaHSBM)
```

Install and load required R packages if not already available:

```
list.of.packages <- c(
  "dplyr",
  "reshape2",
  "reticulate",
  "stringr",
  "tidyr",
  "ROCR"
)
```

```r
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)
}
```

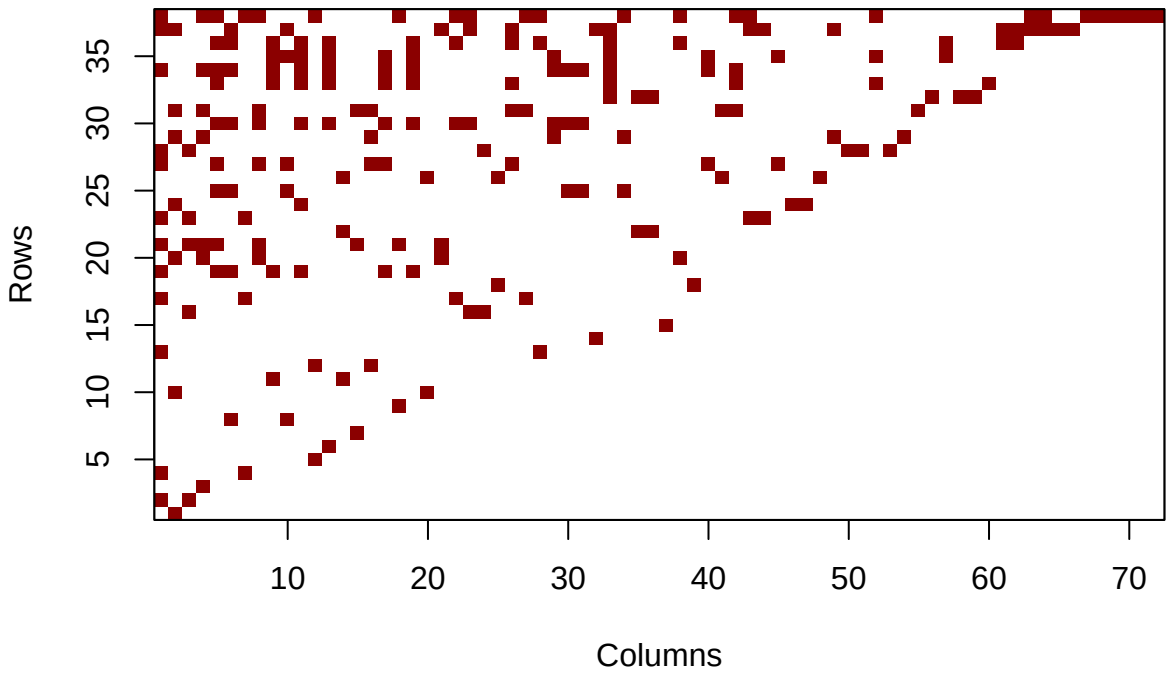
```r
# Record starting time (optional)
start_time <- Sys.time()
```

Load example data

The dataset `dat` is a binary 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 an interaction between them while `0` (in white) represent lack of an observed interaction.

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

```r
# Plot the simulated matrix
plot_interaction_matrix(dat)
```



Preparing Input data for HSBM

The `hsbm.input` function cleans the dataset, and creates cross-validation folds by temporarily holding out subsets of observed links. Internally, it ensures that each fold masks a representative sample. In our example, it randomly splits the 218 observed links into ten similar-sized subsets (because here we use 10-fold cross-validation), holding out one subset per fold to test the model's ability to recover missing links. If the network is unipartite we must set `is_bipartite = FALSE`.

```
# Prepare input object
myInput <- hsbm.input(
  dat,                # Binary matrix of observed links
  n_folds = 10,       # Number of folds for cross-validation
  is_bipartite = TRUE # The matrix represents a bipartite network
)
```

```
# Summarizes network characteristics
summary(myInput)
```

```
##   n_rows n_cols n_links rows_single_link cols_single_link possible_links
## 1     38     72     218               8               22           2736
```

Predicting missing links with "conditional_missing"

The `hsbm.predict` works directly with the processed input created by `hsbm.input`. It applies HSBM to each fold's training network and computes posterior probabilities for links and group assignments for nodes. The `conditional_missing` method focuses exclusively on predicting the conditional posterior probability of unobserved links (`0`s) given the network's learned block structure, useful to identify missing links (i.e., unobserved interactions likely to exist) in partially incomplete networks, without casting doubt on any documented interactions. Computation can be intensive, but parallelization across folds is supported with the `n_cores` argument.

```
# Predict missing links using HSBM
myPred <- hsbm.predict(
  myInput,           # Input data processed by hsbm.input()
  iter = 10000,       # Number of iterations
  wait = 1000,       # Number of iterations for MCMC equilibration
  rnd_seed = 123,       # Sets seed in python environment for reproducibility
  method = "conditional_missing",  # Prediction method
  save_blocks = TRUE,  # Save group assignments
  save_pickle = FALSE, # Save results as pickle files,
  save_plots = FALSE,  # Save hierarchical edge bundling plots
  n_cores = 3 # Number of cores to use
)
```

Predicted probabilities and group assignments are stored for each fold. Below, we extract some probabilities (p) for fold 1. The lower the p, the less likely the link exists and the lower its priority; conversely, higher p values are the most promising missing interactions. A 'reconstructed' edge type refers to an undocumented link.

```
# View probabilities for fold 1
probabilities_fold1 <- myPred$probs[[1]]
print(probabilities_fold1[sample(1:nrow(probabilities_fold1), 10), ])
```

```
##      v1  v2           p v1_names v2_names     edge_type
## 665   3  89 0.0059247814     row4    col52 reconstructed
## 1128 37  94 0.0023905452    row38    col57 reconstructed
```

```
## 2581 37  47 1.0000000000    row38    col10    documented
## 1260 34  85 0.0083410904    row35    col48 reconstructed
## 231   9  53 0.0027616900    row10    col16 reconstructed
## 1208 31 104 0.0028790449    row32    col67 reconstructed
## 962   4  64 0.0134569510     row5    col27 reconstructed
## 2080 18  84 0.0028092943    row19    col47 reconstructed
## 2014 32  88 0.0009332101    row33    col51 reconstructed
## 2063 18  66 0.0156622432    row19    col29 reconstructed
```

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

```
# Check groups in each fold
print(lapply(myPred$groups,
             function(x){
                 g_cols <- grep("^G", names(x))
                 return(apply(x[g_cols], 2, function(y) length(unique(y))))
             }))
```

```
## [[1]]
## G1 G2 G3 G4
##  4  2  1  1
##
## [[2]]
## G1 G2 G3 G4
##  4  2  1  1
##
## [[3]]
## G1 G2 G3 G4
##  4  2  1  1
##
## [[4]]
## G1 G2 G3 G4
##  4  2  1  1
##
## [[5]]
## G1 G2 G3 G4
##  4  2  1  1
##
## [[6]]
## G1 G2 G3 G4
##  4  2  1  1
##
## [[7]]
## G1 G2 G3 G4
##  4  2  1  1
##
## [[8]]
## G1 G2 G3 G4
##  4  2  1  1
##
## [[9]]
## G1 G2 G3 G4
##  4  2  1  1
##
```

```
## [[10]]
## G1 G2 G3 G4
##  4  2  1  1
```

Hierarchical group assignments provide insight into how nodes 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 are progressively aggregated, revealing broader patterns and relationships or communities.

Network Reconstruction

The `hsbm.reconstructed` function generates a reconstructed binary interaction matrix by combining the probabilities across all folds and transforming them into a new binary (0/1) matrix/network from a user-specified threshold.

```
# Network reconstruction
myReconst <- hsbm.reconstructed(
  myPred,                # Predictions processed by hsbm.predict
  rm_documented = TRUE,  # Use of documented entries during validation
  threshold = "prc_closest_topright", # Binarization threshold
  consistency_matrix = "average_thresholded" # Combine fold predictions
)
```

The `myReconst` object includes the final averaged probability matrix, the final reconstructed binary matrix, and evaluation metrics. Let's explore the model overall performance and network reconstruction details.

```
# View the reconstructed network summary and evaluation metrics
summary(myReconst)
```

```
## $`Reconstructed Network Metrics`
##   obs_links unobs_links pred_links kept_links spurious_links missing_links
## 1       218        2518        252        218              0            34
##
## $`Evaluation Metrics`
##   mean_RLF mean_auc mean_aucpr   mean_yPRC mean_prec mean_sens mean_spec
## 1    0.875 0.942772  0.2477377 0.001308816 0.1386388     0.875 0.8865766
##    mean_ACC  mean_ERR  mean_tss
## 1 0.8865655 0.1134345 0.7615766
```

The summary above provides the number of missing links, as well as key evaluation metrics, such as the recovery link fraction (RLF). These results highlight the predicted interactions, showcasing the method's ability to detect missing links effectively.

The `top-links` function identifies the undocumented links most likely to be missing links according to HSBM.

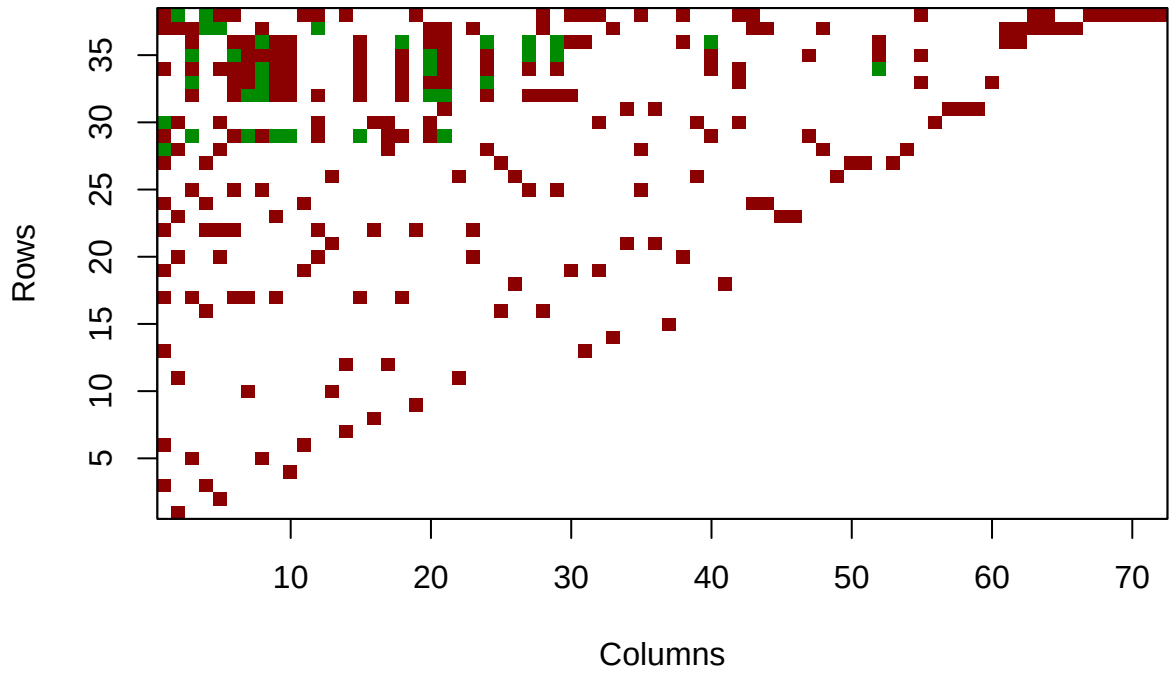
```
# View the top potential missing links
top_links_df <- top_links(myReconst,
                          n = 10,
                          edge_type = "undocumented") # Type of edge to rank
print(top_links_df)
```

```
##    v1_names v2_names         p         sd
## 1      row5     col7 0.4892555 0.05452399
## 2      row2    col29 0.3419086 0.05616368
## 3      row5     col4 0.3204528 0.04828993
## 4      row4     col6 0.3092431 0.05545125
```

```
## 5      row7     col4 0.2699492 0.04293651
## 6      row6    col11 0.2595792 0.03380236
## 7      row6    col12 0.2527598 0.03038593
## 8      row3     col9 0.2426134 0.08706284
## 9      row1    col24 0.2377940 0.04477313
## 10     row3    col14 0.2316924 0.06776324
```

We can visualize the reconstructed matrix highlighting the predicted missing links (colored green).

```
# Plot the matrix with spurious
cols <- c("white", "red4")
new_missing <- which(myReconst$data == 0 & myReconst$new_mat == 1)
dat_missing <- myReconst$data
# Add indicator and color for missing links
if(length(new_missing)!=0){
    dat_missing[new_missing] <- 2
    cols <- c(cols, "green4")
}
plot_interaction_matrix(dat_missing, col = cols)
```



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

Computing characteristics

```
# Show processing time and computer characteristics
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:  33 minutes
```

This analysis was performed on a Dell Inspiron notebook with the following characteristics:

- Processor (CPU): Intel Core i7-7500U @ 3.5GHz
- Memory (RAM): 16 GB
- Operating System: Ubuntu 24.04 (run on a docker image)
- R Version: 4.3.3