Chapter 16

Constructing and Analyzing Microbiome Networks in R

Mehdi Layeghifard, David M. Hwang, and David S. Guttman

Abstract

Microbiomes are complex microbial communities whose structure and function are heavily influenced by microbe–microbe and microbe–host interactions mediated by a range of mechanisms, all of which have been implicated in the modulation of disease progression and clinical outcome. Therefore, understanding the microbiome as a whole, including both the complex interplay among microbial taxa and interactions with their hosts, is essential for understanding the spectrum of roles played by microbiomes in host health, development, dysbiosis, and polymicrobial infections. Network theory, in the form of systems-oriented, graph-theoretical approaches, is an exciting holistic methodology that can facilitate microbiome analysis and enhance our understanding of the complex ecological and evolutionary processes involved. Using network theory, one can model and analyze a microbiome and all its complex interactions in a single network. Here, we describe in detail and step by step, the process of building, analyzing and visualizing microbiome networks from operational taxonomic unit (OTU) tables in R and RStudio, using several different approaches and extensively commented code snippets.

Key words Graph theory, igraph, Microbial co-occurrence, Microbiome, Network, OTU table, R, RStudio

1 Introduction

Microbiomes are complex microbial communities whose structure and function are heavily influenced by microbe–microbe and microbe–host interactions. These interactions are mediated by a range of mechanisms, encompassing direct cell-to-cell contact and interspecies signaling, to indirect metabolite sensing, all of which have been implicated in the modulation of disease progression and clinical outcome [1]. An example of complex microbial interactions involved in disease aggravation is polymicrobial synergism, which describes cases where infections by multiple interacting species of bacteria are more severe than single-agent infections. Polymicrobial synergism is reported to result in increased levels of antibiotic resistance, biofilm development, tissue damage and adaptation to the environment [2, 3]. Therefore, understanding the microbiome as a whole, including both the complex interplay among microbial taxa and interactions with their hosts, is essential for understanding the spectrum of roles played by microbiomes in host health, development, dysbiosis, and polymicrobial infections.

While the widespread adoption of next-generation sequencing technologies has dramatically increased the scope and scale of microbiome studies, the analytical methodologies used to study microbe–microbe and host–microbe interactions are surprisingly limited [4]. Network theory, in the form of systems-oriented, graph-theoretical approaches, is an exciting holistic methodology that can facilitate microbiome analysis and enhance our understanding of the complex ecological and evolutionary processes involved. Using network theory, one can model and analyze a microbiome and all its complex interactions in a single network. An interesting aspect of network theory is that the architectural features of networks appear to be universal to most complex systems, such as microbiomes, molecular interaction networks, computer networks, microcircuits, and social networks [5]. This universality paves the way for using expertise developed in well-studied nonbiological systems to unravel the interwoven relationships that shape microbial interactions.

The first network model described mathematically is the random network introduced in 1960 by Paul Erdős and Alfred Rényi [6]. This model assumes a network of randomly interconnected nodes, in which nodes' degrees will follow a Poisson distribution and most nodes have a number of connections comparable to the network's average degree. Most natural or artificial networks, however, show a power-law degree distribution, where a few nodes have a very large number of connections, while other nodes have no or few connections. These networks are often called scale-free networks [7]. There are also small-world networks, which describe a model in which most nodes are accessible to every other node through a relatively short path. Finally, regular networks are highly ordered nonrandom networks, where all the nodes have exactly the same degree [8].

A wide range of methods, with varying levels of efficiency and accuracy, have been used to construct networks based on microbiome data. The simplest methods are (dis)similarity- or distance-based techniques. The most popular methods, however, are correlation-based techniques, where significant pairwise associations between operational taxonomic units (OTUs, a grouping of organisms circumscribed by a specified level of DNA sequence similarity at a marker gene) are detected using a correlation coefficient such as Pearson's correlation coefficient or Spearman's nonparametric rank correlation coefficient. However, the use of correlation coefficients to detect dependencies between members of a microbiome suffers from limitations such as detecting spurious correlations due to compositionality [9], and being severely underpowered owing to the relatively low number of samples.

The concerns over correlation-based analyses have led to the development of methods that are robust to compositionality. SparCC (Sparse Correlations for Compositional data), for example, is a new technique that uses linear Pearson's correlations between the log-transformed components to infer associations in compositional data [10]. SPIEC-EASI (SParse InversE Covariance Estimation for Ecological Association Inference) is another statistical method for the inference of microbial ecological networks that combines data transformations developed for compositional data analysis with a graphical model inference framework with the assumption that the underlying ecological association network is sparse [11].

An alternative approach uses probabilistic graphical models (PGMs), which provides a probability theory framework based on discrete data structures in computer science, to measure uncertainty in high dimensional data. PGMs use probability theory and graph theory in combination to tackle both uncertainty and complexity in data, simultaneously. PGMs use graphs as the foundation for both measuring joint probability distributions and representing sets of conditional dependence within data in a compact fashion (see [12] for a detailed review of various network construction methods applicable to microbiome data). One method, *EBICglasso*, estimates sparse undirected graphical models for continuous data with multivariate Gaussian distribution through the use of L1 (*lasso*) regularization before using an extended Bayesian information criteria (EBIC) to select the most fitting model. *lasso* regularization is a regression-analysis method that enforces a sparsity constraint on the data that can lead to simpler and more interpretable models less prone to overfitting. *lasso* essentially makes the data smaller (i.e., sparse) and performs both variable selection and regularization in order to enhance the prediction accuracy and interpretability of the statistical model by selecting only a subset of the provided covariates for use in the final model [13]. Given a collection of graphical models for the data, information criteria enable us to estimate the relative quality of each model or tune parameters of penalization methods such as the graphical lasso. Thus, EBIC provides a means for model selection and optimization.

A microbiome network usually consists of clusters (also known as modules) of closely associated microbes (i.e., groups of coexisting or co-evolving microbes), as well as individual microbes that are central to the network's structure (i.e., keystone taxa). Clusters not only represent the local interaction patterns in the network, but also contribute to the network's structure and connectivity, and can have biological, taxonomic, evolutionary, or functional importance. Topological clustering methods are the most popular techniques used to detect clusters in the networks. For example, the "walktrap" algorithm is a bottom-up approach that assumes short random walks of three to five steps through interconnected nodes are more likely to stay within a cluster due to the higher level of interconnectedness within clusters [14]. The Markov clustering (MCL) algorithm, on the other hand, is a stochastic approach that tries to simulate a flow within the network structure, strengthening the flow where nodes are highly interconnected and weakening it in other regions (van [15]). The inherent clustering structure of the network, which influences the flow process, will be eventually revealed when the flow process stabilizes. MCL has found great popularity in network biology and has been used to identify families within protein networks, detect orthologous groups, predict protein complexes from protein interaction networks, and find gene clusters based on expression profiles.

Two popular approaches have been used to detect keystone taxa from microbiome networks: centrality indices and link-analysis methods. Keystone taxa found by node centrality indices are expected to have more links, can reach all the other taxa more quickly, and control the flow between other taxa. Examples of node centrality indices are degree centrality [16], node- and edge-betweenness [16, 17], and closeness [16]. Link-analysis algorithms, on the other hand, are iterative and interactive data-analysis techniques used to evaluate connections between nodes. The PageRank algorithm [18] is a well-known link-analysis method that is based on the assumption that keystone taxa are likely to be more connected to other taxa when compared to non-keystone taxa.

Here, we present a comprehensive protocol for performing network analysis on microbiome data. We will begin with a brief description of how to obtain and set up R and RStudio software and the required packages, followed by explaining how to import and preprocess a microbiome OTU table. Next, we will explain how to build microbiome co-occurrence networks using several different approaches. Then, we will show how to infer clusters of coexisting or co-evolving microbes within the microbiome network before explaining how to find most central microbes (i.e., keystone taxa) within communities. Finally, we will describe how to simulate different types of network as well as how to estimate various network metrics, before discussing network visualization. A microbiome network with clusters, hubs, and other features highlighted is represented in Fig. 1a. More detailed foundational information and descriptions of the microbiome network methods used here are available in [12].

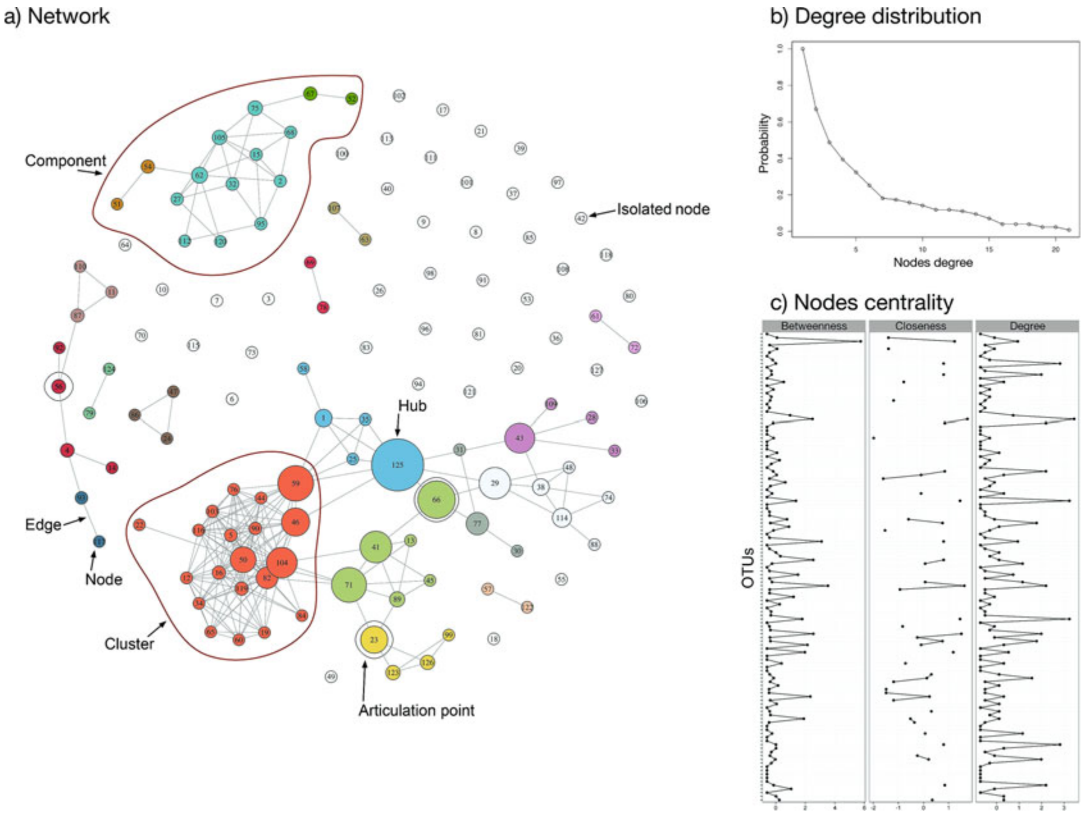

Fig. 1 A microbiome network and its properties. (**a**) the SparCC network constructed from American Gut microbiome project OTU table is shown here. Various network properties discussed in this method are highlighted in this figure. Clusters are shown with different colors, while the isolated nodes are shown in white. The size of the nodes corresponds to their betweenness centrality score. (**b**) The degree distribution of the same network is plotted using the code snippet provided in Subheading 3.9, **step 10**. (**c**) Centrality measures of the network are plotted using the *centralityPlot* function of qgraph package (Subheading 3.9, **step 10**)

2 Materials

2.1 OTU Table

Operational taxonomic unit (OTU) tables are usually in the form of a $n \times m$ matrix, where n is number of samples and m is the number of OTUs, or vice versa. OTU tables can usually be resolved at different taxonomic levels using microbiome-processing tools such as QIIME. Here, the OTU table from the American Gut project [19], which is included in the SpiecEasi package, will be used to construct microbiome networks. The American Gut data come from two sampling rounds (containing 304 and 254 human samples, respectively) and comprise several thousand OTUs. The data used here is a smaller subsample (containing 289 samples and 127 OTUs) created by Kurtz et al. [11] for demo purposes after several preprocessing and filtering steps. These steps are detailed here [11].

2.2 Obtaining R and RStudio

The method described here relies upon R, a free software environment for statistical computing and graphics, and RStudio, an integrated development environment (IDE) for R. RStudio includes a console and a syntax-highlighting editor that supports direct code execution, as well as tools for plotting, command history, debugging, and workspace management. R and RStudio are available for all major operating systems, and can be downloaded from https://www.r-project.org/ and https://www.rstudio.com/products/rstudio/, respectively. In the code snippets throughout this chapter, lines starting with "#" are comments provided to make the code more legible to the user. Moreover, independent lines of code or commands are accommodated on single lines. Blocks of code (i.e., lines of code encapsulated within round or curly brackets), however, are naturally broken down into several lines for improved readability.

2.3 Obtaining Required Libraries

igraph and qgraph are two collections of open source and free network analysis tools that can be used in R to construct, simulate, analyze, and visualize networks. Moreover, the vegan package will be used to calculate pairwise (dis)similarity/distance between OTUs as well as to permute the OTU table. The MCL package is needed for cluster detection and SpiecEasi will be used to construct microbiome networks. These packages can be installed in R by copying the following commands into RStudio, or by entering the package names in the dropdown menu *Tools / Install Packages.*

```
# Install Required packages
install.packages("igraph")
install.packages("qgraph")
install.packages("vegan")
install.packages("MCL")
# Install SpiecEasi package
install.packages("devtools")
library(devtools)
install_github("zdk123/SpiecEasi")
```

Here, R version 3.4.2 was used on Ubuntu Linux (Ubuntu 17.10; Platform: x86_64-pc-linux-gnu) with SpiecEasi_0.1.3, devtools_1.13.4, MCL_1.0, vegan_2.4-5, qgraph_1.4.4, igraph_1.1.2, and BiocInstaller_1.28.0 as installed packages.

Depending on the operating-system setup and the software already installed on the system, some packages might not be installed due to lacking dependencies. In that case, we should call the *install.packages* function with an extra argument to make sure the dependencies are also installed.

```
install.packages("igraph", dependencies = TRUE)
```

All the packages discussed in this method come with manual or tutorials, which we encourage the users to read carefully for more information about the available methods, their applications and detailed real-life or toy examples. While typing a command in RStudio or R console, pressing the "tab" key will suggest options. This is especially useful, if user forgets the arguments that could be supplied to a function.

3 Methods

3.1 Load and Process OTU Table

1. First, we need to load the required packages into the R environment.

```
library(igraph)
library(qgraph)
library(vegan)
library(MCL)
library(SpiecEasi)
```

2. We will use an OTU table from the American Gut project as an example dataset. First, load this file into R, and then reformat the row and column names to OTU_x and Sample_x, respectively, where *x* is the number of that row or column, which will help with readability. You can view the data in RStudio after loading the file and changing the row and column names by selecting the file name (amgut1.filt) in the **Data** window.

```
# Load OTU table
data("amgut1.filt")
# Change row and column names to a more readable format
colnames(amgut1.filt) <- sapply(1:ncol(amgut1.filt),
                         function(x) paste("OTU", x, sep = "_"))
rownames(amgut1.filt) <- sapply(1:nrow--(amgut1.filt),
                         function(x) paste("Sample", x, sep = "_"))
```

3. Alternatively, any user-provided OTU table can be loaded in R environment using the following command, given that the file includes header and row names and entries are separated by ",". *See* **Note 1** to learn how to get help on built-in functions.

```
user.table <- read.table(infile,
                         header = T,
                         row.names = 1,
                         sep = ",")
```

4. Values in an OTU table are usually reported as absolute abundances. For some of the downstream processing, however, we will need to make samples comparable to each other by converting the absolute abundance values to relative abundances.

See **Notes 2** and **3** for more information on how to deal with missing data and filter poor samples, respectively.

```
otu.relative <- amgut1.filt / rowSums(amgut1.filt)
```

3.2 Dissimilarity-Based Network

1. To build a microbiome network using dissimilarity-based approaches, we will need to first compute a pairwise dissimilarity matrix from the OTU table. This can be performed by calling the *vegdist* function from vegan package using a variety of distance and dissimilarity indices, including "manhattan," "euclidean," "bray," and "jaccard." Here, we use the Bray–-Curtis dissimilarity index, which is a statistic used to quantify the compositional dissimilarity between two different samples, based on the counts in each sample. The Bray–Curtis dissimilarity is bounded between 0 and 1, where 0 means identical composition of OTUs, and 1 means lack of any shared OTUs. Given that *vegdist* by default calculates distances between the rows while we want distances between the columns (i.e., OTUs), we need to transpose our input OTU table (*t(otu.relative)*; *see* **Note 4**).

```
# Create dissimilarity matrix
distances <- vegdist(t(otu.relative),
                      method = "bray")
```

2. In order to build a network, the dissimilarity matrix needs to be converted to an adjacency matrix (via first converting the distance object to a matrix using *as.matrix* function). An adjacency matrix A is an $n \times n$ binary matrix (i.e., only containing 1s and 0s), where n is the number of OTUs and entry $A_{i,j}$ (i.e., entry at row i and column j) indicates a link between OTU_i and OTU_j, only if the value at that entry is 1. This conversion can be performed by specifying a threshold. For example, the following code snippet will convert the dissimilarity matrix to an adjacency matrix with threshold set at 0.6, so that two OTUs with a dissimilarity index smaller than or equal to 0.6 will be connected to each other. *See* **Notes 5** and **6** to learn how to import and export networks. By setting mode and diagonal parameters (*mode* and *diag* below) to "undirected" and FALSE, respectively, we make sure to get an undirected network without loops.

```
# Convert distance object to a matrix
diss.mat <- as.matrix(distances)
diss.cutoff <- 0.6
diss.adj <- ifelse(diss.mat <= diss.cutoff, 1, 0)
# Construct microbiome network from adjacency matrix
diss.net <- graph.adjacency(diss.adj,
                            mode = "undirected",
                            diag = FALSE)
```

3.3 Correlation-Based Network

1. An alternative approach to construct a microbiome network is to use pairwise correlation coefficients (i.e., Pearson, Kendall, or Spearman) calculated between the OTUs. The *cor* function, by default, computes the correlations between the columns of the input matrix.

```
cor.matrix <- cor(otu.relative, method = "pearson")
```

2. Similar to dissimilarity-based methods, we need to convert the correlation matrix into an adjacency matrix. Unlike dissimilarity-based methods, however, we compare the absolute values to a threshold (0.3 in the example below), because we are interested in both positive and negative correlations.

```
# Convert correlation matrix to binary adjacency matrix
cor.cutoff <- 0.3
cor.adj <- ifelse(abs(cor.matrix) >= cor.cutoff, 1, 0)
# Construct microbiome network from adjacency matrix
cor.net <- graph.adjacency(cor.adj,
                           mode = "undirected",
                           diag = FALSE)
```

3. In order to avoid including false positives (i.e., spurious correlations) in the network, one can permute the OTU table many times (100 times in the example below) and calculate a *p*-value for each possible pairwise interaction to test the validity of the detected interaction. A small *p*-value (e.g., ≤ 0.01 used below) indicates strong evidence against the null hypothesis that the detected interactions are spurious or random. The following function will take the OTU table (designated as MB within the function), correlation method, number of permutations and the desired significance level as input to generate the underlying microbiome network. After applying the Benjamini-Hochberg correction for multiple tests, it will then convert the corrected *p*-values to an adjacency matrix, from which it will construct the microbiome network. The permutation parameters can be adjusted by the user via changing the arguments of the *permatfull* function.

```
# Execute this command after running Function 1
cor.net.2 <- build.cor.net(amgut1.filt,
                           method = 'pearson',
                           num_perms = 100,
                           sig_level = 0.01)


##### Function 1: Construct microbiome network using
permutation
build.cor.net <- function(MB, method, num_perms, sig_le-
vel) {
    taxa <- dim(MB)[2]
    MB.mat <- array(0, dim = c(taxa, taxa, num_perms + 1))
    # Perform permutation
```

```
  MBperm <- permatswap(MB, "quasiswap", times = num_perms)
  # Convert to relative abundance
  MB.relative <- MB / rowSums(MB)
  MB.mat[,,1] <- as.matrix(cor(MB.relative, method=method))
  for(p in 2:num_perms) {
  MBperm.relative <- MBperm$perm[[p-1]]
            / rowSums(MBperm$perm[[p-1]])
  MB.mat[, , p] <- as.matrix(cor(MBperm.relative, method =
  method))
  }
  # Get p-values
  pvals <- sapply(1:taxa,
               function(i) sapply(1:taxa, function(j)
          sum(MB.mat[i, j, 1] > MB.mat[i, j, 2:num_perms])))
  pvals <- pvals / num_perms
  # p-value correction
  pvals_BH <- array(p.adjust(pvals, method = "BH"),
                  dim=c(nrow(pvals), ncol(pvals)))
  # Build adjacency matrix
  adj.mat <- ifelse(pvals_BH >= (1 - sig_level), 1, 0)
  # Add names to rows & cols
  rownames(adj.mat) <- colnames(MB)
  colnames(adj.mat) <- colnames(MB)
  # Build and return the network
  graph <- graph.adjacency(adj.mat, mode = "undirected", diag =
FALSE)
}
```

3.4 Graphical Model Networks

1. We can use the *EBICglasso* function of qgraph to build a microbiome network by computing the underlying sparse Gaussian graphical model of our OTU table, using graphical lasso based on extended Bayesian information criterion (EBIC). We first compute a correlation or partial correlation matrix from the OTU table. The graphical lasso technique will then be used to find the graph with the best EBIC. Next, we will build the qgraph network and convert it to an igraph network (*see* **Note 7**).

```
# Compute (partial) correlations
ebic.cor <- cor_auto(amgut1.filt)
# Identify graph with the best EBIC
ebic.graph <- EBICglasso(ebic.cor, ncol(amgut1.filt), 0.5)
# Build the network
ebic.qgnet <- qgraph(ebic.graph, DoNotPlot = TRUE)
# Convert to igraph network
ebic.net <- as.igraph(ebic.qgnet, attributes = TRUE)
```

2. We can also use the *FDRnetwork* function of qgraph to find the OTU table's underlying graphical model using local false discovery rate. Similar to *EBICglasso*, we first compute a correlation or partial correlation matrix. Next, we build the graphical model using one of the three available methods: "lfdr" for the local false discovery rate, "pval" for the p-value, and "qval" for the q-value. Q-values are p-values that have been adjusted for the False Discovery Rate (FDR; the proportion of false positives expected to result from a test). For comparison, while a p-value ≤ 0.01 means that less than or equal to 1% of all tests will result in false positives, a q-value $\leq 1\%$ indicates that less than or equal to 1% of only significant results will lead to false positives. Finally, we will build the qgraph network and convert it to an igraph network.

```
# Compute (partial) correlations
fdr.cor <- cor_auto(amgut1.filt)
# Identify graphical model
fdr.graph <- FDRnetwork(fdr.cor, cutoff = 0.01, method =
"pval")
# Build the network
fdr.qgnet <- qgraph(fdr.graph, DoNotPlot = TRUE)
# Convert to igraph network
fdr.net <- as.igraph(fdr.qgnet, attributes = TRUE)
```

3.5 SparCC and SPIEC-EASI Networks

1. SparCC networks can be constructed by feeding the OTU table (with absolute abundance values) to the *sparcc* function of the SpiecEasi package followed by converting the correlation matrix to an adjacency matrix using a threshold (*sparcc.cutoff <- 0.3* below).

```
# SparCC network
sparcc.matrix <- sparcc(amgut1.filt)
sparcc.cutoff <- 0.3
sparcc.adj <- ifelse(abs(sparcc.matrix$Cor) >= sparcc.
cutoff, 1, 0)
# Add OTU names to rows and columns
rownames(sparcc.adj) <- colnames(amgut1.filt)
colnames(sparcc.adj) <- colnames(amgut1.filt)
# Build network from adjacency
sparcc.net <- graph.adjacency(sparcc.adj,
                              mode = "undirected",
                              diag = FALSE)
```

2. SPIEC-EASI networks are constructed using the *spiec.easi* function of the SpiecEasi package. The resulting object contains a matrix called *refit*, which is a sparse adjacency matrix that can be directly used to build the microbiome network.

```
# SPIEC-EASI network
SpiecEasi.matrix <- spiec.easi(amgut1.filt,
                               method = 'glasso',
                               lambda.min.ratio = 1e-2,
                               nlambda = 20,
                               icov.select.params = list(rep.num = 50))
# Add OTU names to rows and columns
rownames(SpiecEasi.matrix$refit) <- colnames(amgut1.filt)
# Build network from adjacency
SpiecEasi.net <- graph.adjacency(SpiecEasi.matrix$refit,
                                 mode = "undirected",
                                 diag = FALSE)
```

3.6 Hub Detection

1. Hubs are nodes in the network that have a significantly larger number of links compared to the other nodes in the network. A hub in a microbiome network can be considered as an equivalent to a keystone species in the microbial community. Using centrality indices (closeness and betweenness below) to find keystone species will output a vector containing values between 0 and 1 for every node in the network. Link-analysis methods (page_rank and hub_score below), on the other hand, will output an object in which there is a vector containing the values for the nodes.

```
# Use sparcc.net for the rest of the method
net <- sparcc.net
# Hub detection
net.cn <- closeness(net)
net.bn <- betweenness(net)
net.pr <- page_rank(net)$vector
net.hs <- hub_score(net)$vector
```

2. These centrality vectors can be sorted to select taxa with highest probability of being keystone species. In the following, nodes are sorted based on their *hub_score* measurements (*net.hs* below) and the top 5 ($n = 5$ below) are chosen.

```
# Sort the species based on hubbiness score
net.hs.sort <- sort(net.hs, decreasing = TRUE)
# Choose the top 5 keystone species
net.hs.top5 <- head(net.hs.sort, n = 5)
```

3.7 Cluster Detection

1. Two cluster-detection methods from the igraph package and one from the MCL package are used here. It should be noted that igraph methods output an object containing various information on the detected clusters, including but not limited to the membership of each cluster (*membership* function below).

While igraph methods work directly on the network object, MCL should be applied to the adjacency matrix (*adj* below). *See* **Note 8** for more detail on alternative methods.

```
# Get clusters
wt <- walktrap.community(net)
ml <- multilevel.community(net)
# Get membership of walktrap clusters
membership(wt)
# Get clusters using MCL method
adj <- as_adjacency_matrix(net)
mc <- mcl(adj, addLoops = TRUE)
```

2. The clusters detected by various methods can be compared to each other using igraph's *compare* function. In addition, customized vectors of known or expected cluster memberships can be created in order to compare with the results of the clustering methods. Here, we divided the nodes into five clusters by random sampling (*sample* function below). This, however, could be replaced by a user-provided list. In case of identical clusters, the output will be 0. Identified clusters (*wt* below) can be plotted as a dendrogram.

```
# Compare clusters detected by different methods
compare(membership(wt), membership(ml))
compare(membership(wt), mc$Cluster)
# Create customized membership for comparison
expected.cls <- sample(1:5, vcount(net), replace = T) %>%
as_membership
compare(expected.cls, membership(wt))
# Plot clusters as dendrogram
plot_dendrogram(wt)
```

3. One measure of the strength of division of a network into clusters or modules is network modularity. High modularity indicates that the network has dense connections within certain groups of nodes and sparse connections between these groups. The modularity of a graph with respect to a given membership vector can be used to estimate how separated different clusters of taxa are from each other.

```
# Calculate modularity
modularity(net, membership(wt))
```

3.8 Network Simulation

Network simulation is usually used for various comparative or analytical reasons. The code snippet below will generate regular, random, small-world, and scale-free networks, respectively. The variable *num.nodes* indicates the number of nodes in the simulated network. *k* represents the degree of each node in the regular network. *p* stands for the probability of drawing an edge between two arbitrary nodes in the random network and the rewiring probability in the small-world network. *dim* and *nei* in the small-world network represent, respectively, the dimensions of the starting lattice and the neighborhood within which the node of the lattice will be connected. All these functions default to undirected networks. *See* **Note 9** on how to simulate fictional OTU tables.

```
# Simulate networks
Num.nodes <- 50
regular.net <- k.regular.game(num.nodes, k = 4)
random.net <- erdos.renyi.game(num.nodes, p = 0.037)
smallworld.net <- sample_smallworld(dim = 1, num.nodes, nei =
2, p = 0.2)
scalefree.net <- barabasi.game(num.nodes)
```

3.9 Network Features

1. Basic features of a network, such as vectors of nodes or edges, names of the nodes, and number of nodes or edges can be extracted using the following commands.

```
# Network features
nodes <- V(net)
edges <- V(net)
node.names <- V(net)$name
num.nodes <- vcount(net)
num.edges <- ecount(net)
```

2. Transitivity, also known as the clustering coefficient, measures the probability that the adjacent nodes of a certain node are themselves connected. Using local type will generate a score for every node in the network, whereas using global type will produce one transitivity score for the whole network.

```
clustering_coeff <- transitivity(net, type = "global")
```

3. If we are interested in knowing which nodes are directly connected to any given node in the network, we can use the *neighbors* function. Moreover, we can see if two nodes share any neighboring nodes using the *intersection* function. OTU nodes 1 and 25 are labeled as 1 and 25 in Fig. 1a.

```
# Obtain the neighbors of nodes 1 and 25
otu1_neighbors <- neighbors(net, "OTU_1")
otu25_neighbors <- neighbors(net, "OTU_25")
# Find neighbors shared by nodes 1 and 25
intersection(otu1_neighbors, otu25_neighbors)
```

4. All the edges incident to one or multiple nodes (i.e., all edges connecting that node or nodes to other nodes) can be obtained using *incident* and *incident_edges* functions, respectively. The number of links connecting any given node to the network is that node's degree. Indegree of a node is the number of links ending at that node and outdegree is the number of links originating from the node. In undirected networks, indegree and outdegree are the same, hence the mode is set to "all."

```
# Edges incident to OTU_1
otu1.edges <- incident(net, "OTU_1", mode = "all")
# Edges incident to OTU_1 and OTU_25
otus.edges <- incident_edges(net, c("OTU_1", "OTU_25"),
mode = "all")
# Extracting/printing the incident edges separately
otus.edges$"OTU_1"
otus.edges$"OTU_25"
```

5. The average nearest neighbor degree (ANND) of a given node (or a set of nodes) can be calculated using the *knn* function. ANND is a measure of the dependencies between degrees of neighbor nodes. This allows us to test if the correlation between degrees of neighbor nodes is positive and the nodes of high degree have a preference to connect to other nodes of high degree or the correlation is negative and the nodes of high degree have a connection preference for nodes of low degree. The following code snippet calculates and prints the average nearest neighbor degree for all the nodes in the network (hence, *vids* = *V(net)*).

```
net.knn <- knn(net, vids = V(net))
net.knn$knn
```

6. To find all nodes reachable, directly or indirectly, from a given node (e.g., OTU_1 below) we can use the *subcomponent* function which outputs a list of connections.

```
sub.node1 <- subcomponent(net, v = "OTU_1", mode = "all")
```

7. Isolated nodes that are not connected to any other node in the network can be removed as follows.

```
clean.net <- delete.vertices(net, which(degree(net, mode = "all")
== 0))
```

8. Sometimes, a network is consisted of multiple disconnected components. These components can be obtained and printed as follows.

```
# Network components
net.comps <- components(net)
```

```
# Print components membership
net.comps$membership
# Print components sizes
net.comps$csize
# Print number of components
net.comps$no
```

9. Then, the largest or any other component can be used to induce (i.e., extract) a subnetwork from the microbiome network using *induced_subgraph* function. In fact, any set of nodes can be selected via R's standard subsetting techniques (shown below to extract components) and used to induce a subnetwork. All the methods applicable to a network can also be applied to the subnetworks.

```
# Largest component
largest.comp <- V(net)[which.max(net.comps$csize) == net.
comps$membership]
# Second component
second.comp <- V(net)[net.comps$membership == 2]
# The component containing OTU_1
otu1.comp <- V(net)[net.comps$membership ==
                      which(names(net.comps$membership) ==
"OTU_1")]
# Largest component subnetwork
largest.subnet <- induced_subgraph(net, largest.comp)
# Subnetwork for the component containing OTU_1
otu1.subnet <- induced_subgraph(net, otu1.comp)
```

10. The degree distribution of a network's nodes can be obtained and plotted to gain a better grasp of node connectivity (Fig. 1b). The *centralityPlot* function of the qgraph package can also be used to plot nodes' degree, closeness and betweenness measures for a network (or several networks), side by side, for comparison purposes (Fig. 1c). It should be noted that qgraph functions can be applied to networks produced by igraph without any modification. Here, the degree distribution is plotted as the cumulative sum of degrees (*cumulative* = *T*). To disable it, one should set the cumulative parameter to *False* (*cumulative* = *F*).

```
# Degrees
deg <- degree(net, mode = "all")
# Degree distribution
deg.dist <- degree_distribution(net, mode = "all", cumulative = T)
# Plot degree distribution
plot(deg.dist, xlab = "Nodes degree", ylab = "Probability")
lines(deg.dist)
# qgraph method
centralityPlot(net)
```

11. Real-life random or regular networks are rare. Small-world and scale-free networks, on the other hand, are quite common and, sometimes, it is important to know the type of the network at hand. The *fit_power_law* of igraph package tries to fit a power law function to the degree distribution and outputs a p-value, among other statistics, to indicate if the test rejects the hypothesis (p-value < 0.05) that network's degree distribution is drawn from the fitted power-law distribution. One notable member of scale-free family of network models is the hierarchical network model. Unlike other scale-free networks that predict an inverse relationship between the average clustering coefficient and the number of nodes, hierarchical networks show no relationship between the size of the network and its average clustering coefficient. Usually, the power-law behavior starts showing only above a threshold value, which if provided (*xmin = 10*, below), allows to fit only the tail of the distribution. The *smallworldness* function of qgraph package outputs a vector of statistics, first of which is a *smallworldness* score. This function measures the transitivity and the average shortest path length of the input network before computing the average of the same measures on a number of random networks (10 random networks here, $B = 10$). The small-worldness score is then computed as the transitivity of the input network over its average shortest path length after both being normalized by the same measures obtained from the random networks. If this score is higher than 1 (or higher than 3, to be more stringent), the network will be considered of small-world type.

```
# Scalefreeness: Fit a power_law to the network
deg <- degree(net, mode = "in")
pl <- fit_power_law(deg, xmin = 10)
pl$KS.p
# Smallworldness
sw <- smallworldness(net, B = 10)
sw[1]
```

12. The intersection and union of edges of two networks can be obtained, in order to compare two networks constructed by two different methods from the same data or two networks built from different time points. Here, we use the subnetwork containing OTU_1 (see above) as the second network.

```
intsect.edges <- intersection(net, otu1.subnet)
union.edges <- union(net, otu1.subnet)
```

13. A pairwise Jaccard similarity matrix can be calculated for some or all network nodes in order to know how similar some taxa

are in terms of their connection patterns in the network. Here, a similarity matrix is calculated for all the nodes.

```
node.similarity <- similarity(net, vids = V(net), mode = "all",
method = "jaccard")
```

14. Some nodes in the network are so central to the whole or a part of network that their removal will break the network into more components. These nodes are called articulation points or cut vertices and can be identified as follows.

```
# Find articulation points
AP <- articulation.points(net)
```

3.10 Network Visualization

1. The simplest way to visualize the network is to use igraph's built-in plot function with the network object as either the only parameter or together with the object containing the identified clusters (*wt* below). *See* **Note 10** on how to use igraph demo function.

```
# Simple plotting
plot(net)
plot(wt, net)
```

2. The visualization can be customized by feeding various parameters to the plot function. Here, for example, we customized nodes color, size, shape, frame color, label size, and label color. Similar customization can be applied to edge attributes as shown below with edge color. We also specified a layout for the network (Fig. 1a; **Note 11**).

```
# Customized plotting
plot(net,
     main = "Microbiome Network",
     vertex.color = "white",
     vertex.size = 12,
     vertex.shape = "circle",
     vertex.frame.color = "green",
     Vertex.label.size = 1,
     Vertex.label.color = "black",
     edge.color = "grey",
     layout = layout.fruchterman.reingold)
```

3. When the node names are long or there is a possibility of node labels overlapping due to the larger number of nodes, we can plot the network with node numbers and provide the node labels in the legend (*see* **Note 12** on how to handle large datasets). The following function, in addition to moving the node labels to the legend, will also scale the size of the nodes

based on their hub score (either calculated by one of the igraph hub detection methods or provided by user) and saves the plot as an image. The size or quality of the image can be customized by modifying the line that defines the image properties. This function takes four arguments; microbiome network, the hub scores, a name for the output file, and a title for the plot. Similar to previous section, node and edge attributes can be customized by changing the arguments fed to the *plot* function within Function 2.

```
# Function 2: Plot network with node size scaled to hubbiness
plot.net <- function(net, scores, outfile, title) {
    # Convert node label from names to numerical IDs.
    features <- V(net)$name
    col_ids <- seq(1, length(features))
    V(net)$name <- col_ids
    node.names <- features[V(net)]

    # Nodes' color.
    V(net)$color <- "white"

    # Define output image file.
    outfile <- paste(outfile, "jpg", sep=".")
    # Image properties.
   jpeg(outfile,width=4800,height=9200,res=300,quality=100)
   par(oma = c(4, 1, 1, 1))

   # Main plot function.
   plot(net, vertex.size = (scores*5)+4, vertex.label.cex = 1)
   title(title, cex.main = 4)

   # Plot legend containing OTU names.
   labels = paste(as.character(V(net)), node.names, sep = ") ")
   legend("bottom", legend = labels, xpd = TRUE, ncol = 5, cex = 1.2)
   dev.off()
}

  # Execute this command after running Function 2
  plot.net(net, net.hs, outfile = "network1", title = "My
  Network")
```

4. The next function will take two more arguments: membership of the detected clusters and articulation points (*cls* and *AP* below, respectively). The nodes will be colored based on their cluster membership and articulation points will be highlighted with a halo. Isolated nodes that are not member of any cluster will be colored white. We also added a custom layout using qgraph package in order to separate the overlapping nodes.

This layout can be further customized by changing the parameters fed into the pertinent function below (i.e., *qgraph.layout.fruchtermanreingold* function).

```
# Function 3: Plot network with clusters and node size scaled to
hubbiness
plot.net.cls <- function(net, scores, cls, AP, outfile, title) {
      # Get size of clusters to find isolated nodes.
      cls_sizes <- sapply(groups(cls), length)
      # Randomly choosing node colors. Users can provide their own
      vector of colors.
      colors <- sample(colours(), length(cls))
      # Nodes in clusters will be color coded. Isolated nodes will be
      white.
        V(net)$color <- sapply(membership(cls),
                              function(x) {ifelse(cls_sizes[x]>1,
          colors[x], "white")})

      # Convert node label from names to numerical IDs.
      node.names <- V(net)$name
      col_ids <- seq(1, length(node.names))
      V(net)$name <- col_ids

      # To draw a halo around articulation points.
      AP <- lapply(names(AP), function(x) x)
      marks <- lapply(1:length(AP), function(x) which(node.names ==
      AP[[x]]))

      # Define output image file.
      outfile <- paste(outfile, "jpg", sep=".")
      # Image properties.
      jpeg(outfile, width = 4800, height = 9200, res = 300, quality =
      100)
      par(oma = c(4, 1, 1, 1))

      # Customized layout to avoid nodes overlapping.
      e <- get.edgelist(net)
      class(e) <- "numeric"
      l <- qgraph.layout.fruchtermanreingold(e, vcount=vcount(net),

      area=8*(vcount(net)^2),

      repulse.rad=(vcount(net)^3.1))
          # Main plot function.
          plot(net, vertex.size = (scores*5)+4, vertex.label.cex=0.9,
               vertex.label.color = "black",
               mark.border="black",
               mark.groups = marks,
        mark.col = "white",
        mark.expand = 10,
        mark.shape = 1,
        layout=l)
        title(title, cex.main=4)

        # Plot legend containing OTU names.
        labels = paste(as.character(V(net)), node.names, sep =
        ") ")
        legend("bottom", legend = labels, xpd = TRUE, ncol =
        5, cex = 1.2)
            dev.off()
        }

# Execute this command after running Function 3
plot.net.cls(net, net.hs, wt, AP,
             outfile = "network2", title = "My Network")
```

4 Notes

1. The full description of a given function, the parameters it accepts, and the output it produces can be obtained by typing the question mark before the function's name. For example, to get more information on *walktrap.community* function, we can run the following command.

```
?walktrap.community
```

2. Many functions can handle missing values in data automatically. Yet, it is always a good practice to take care of missing values explicitly. For example, the following command removes all taxa with missing values, provided that the missing values are represented as NAs.

```
otu.table <- otu.table[ , colSums(is.na(otu.table)) == 0]
```

3. Although optional, it is usually considered good practice to drop poor samples with fewer of observations. This line of code drops samples that have less than 1% of the observations of the largest sample in the OTU table.

```
otu.table <- otu.table[, colSums(abs(otu.table)) >
                        floor(max(colSums(otu.table))/100)]
```

4. Among R functions that accept matrices as input, some apply the operation in a column-wise fashion (i.e., *cor* function) while others work in a row-wise fashion (i.e., *vegdist* function). Therefore, it is critical to provide the input matrix in the correct

format by transposing the input accordingly, prior to or while feeding the input to the desired function.

```
otu.t <- t(otu)
```

5. In cases where user already has access to a microbiome network, it could be imported into R, if the file format is known. The correct file format should be selected from the list below (e.g., *format = "graphml"*).

```
imported_net <- read_graph(infile,
                           format = c("edgelist", "pajek",
                           "ncol", "lgl", "graphml", "di-
                           macs", "graphdb", "gml", "dl"))
```

6. Similarly, the microbiome network constructed in R can be exported into a file to be imported and used in other software. The desired file format can be selected from the list below (e.g., *format = "graphml"*).

```
write_graph(net, outfile,
            format = c("edgelist", "pajek", "ncol", "lgl",
                       "graphml", "dimacs", "gml", "dot",
"leda"))
```

7. Even though the qgraph network could be easily plotted or analyzed using available methods in the qgraph package, for the sake of consistency as well as the greater repertoire of analytical tools in igraph package, we recommend converting to an igraph network. It should be noted that sometimes qgraph to igraph object conversion might throw some warnings due to the differences between graph attributes that could be transferred between the two objects. The network topology, however, will be accurately transferred.
8. The igraph package comes with several built-in methods to detect clusters within microbiome networks including edge betweenness, fast greedy, leading eigenvectors, Louvain multilevel, spin-glass, and label propagation. Although these methods use various algorithms, their main function, which is cluster detection, remains the same. igraph's manual provides detailed information on how to use each of these methods.
9. Sometimes one needs to create and use a toy OTU table for testing purposes. The following code snippet produces an OTU table with 20 OTUs and 50 samples with abundance values randomly chosen from between 1 and 100, before row and column names are assigned.

```
# Create a simulated OTU table followed by adding row and
column names
otu.table <- matrix(sample(1:100, 1000, replace = TRUE),
                    nrow = 20,
                    ncol = 50)
rownames(otu.table) <- paste("OTU", 1:nrow(otu.table), sep
= "")
colnames(otu.table) <- paste("Sample", 1:ncol(otu.table),
sep = "")
```

10. igraph comes with a demo function that can be used to get a solid grasp of methods available in the package (e.g., *centrality* function below). The *igraph_demo* function can be called without any arguments to see what demos are available, before choosing one of the available demos. Moreover, this function can be run interactively.

```
# Get available demos
igraph_demo()
igraph_demo("centrality")
# Run interactively
if (interactive()) {
    igraph_demo("centrality")
}
```

11. Every time we plot a network, the positions of the nodes are recalculated, even when using the same layout. Therefore, to have a more visually stable and comparable representation, we should either use a fixed set of coordinates for all the nodes or choose a desired layout and assign it as a fixed attribute of the network. The fixed set of coordinates should be in the form of a $n \times m$ matrix, where n is the number of nodes and m is the x and y coordinates for each node.

```
# Assign custom (random) coordinates to layout
net$layout <- array(1:40, dim = c(20, 2))
# Assign a layout as a fixed attribute of the network
net$layout <- layout.fruchterman.reingold(net)
```

12. When working with large datasets containing thousands of OTUs, we can convert the adjacency matrix to a sparse matrix using *Matrix* function for a faster and more memory-efficient analysis.

```
sparse.adj <- Matrix(adj, sparse=TRUE)
```

References

1. Magalhaes AP, Azevedo NF, Pereira MO, Lopes SP (2016) The cystic fibrosis microbiome in an ecological perspective and its impact in antibiotic therapy. Appl Microbiol Biotechnol 100:1163–1181. https://doi.org/10.1007/s00253-015-7177-x
2. Dalton T, Dowd SE, Wolcott RD, Sun Y, Watters C, Griswold JA, Rumbaugh KP (2011) An in vivo polymicrobial biofilm wound infection model to study interspecies interactions. PLoS One 6:e27317. https://doi.org/10.1371/journal.pone.0027317
3. Murray JL, Connell JL, Stacy A, Turner KH, Whiteley M (2014) Mechanisms of synergy in polymicrobial infections. J Microbiol 52:188–199. https://doi.org/10.1007/s12275-014-4067-3
4. Legendre P, Legendre L (2012) Numerical ecology. Developments in environmental modelling, vol 24, 3rd English ed. Elsevier, Amsterdam
5. Barabási AL, Oltvai ZN (2004) Network biology: understanding the cell's functional organization. Nat Rev Genet 5:101–113. https://doi.org/10.1038/nrg1272
6. P. Erdős AR On the evolution of random graphs. In: Publication of the Mathematical Institute of the Hungarian Academy of Sciences; 1960
7. Barabasi AL, Albert R (1999) Emergence of scaling in random networks. Science 286:509–512
8. Watts DJ, Strogatz SH (1998) Collective dynamics of 'small-world' networks. Nature 393:440–442. https://doi.org/10.1038/30918
9. Chen EZ, Li H (2016) A two-part mixed-effects model for analyzing longitudinal microbiome compositional data. Bioinformatics (Oxford, England) 32:2611–2617. https://doi.org/10.1093/bioinformatics/btw308
10. Friedman J, Alm EJ (2012) Inferring correlation networks from genomic survey data. PLoS Comput Biol 8:e1002687. https://doi.org/10.1371/journal.pcbi.1002687
11. Kurtz ZD, Muller CL, Miraldi ER, Littman DR, Blaser MJ, Bonneau RA (2015) Sparse and compositionally robust inference of microbial ecological networks. PLoS Comput Biol 11:e1004226. https://doi.org/10.1371/journal.pcbi.1004226
12. Layeghifard M, Hwang DM, Guttman DS (2017) Disentangling Interactions in the Microbiome: A Network Perspective. Trends Microbiol 25:217–228. https://doi.org/10.1016/j.tim.2016.11.008
13. Tibshirani R (2018) Regression shrinkage and selection via the lasso: a retrospective. J Roy Stat Soc Ser B (Stat Method) 73:273–282. https://doi.org/10.1111/j.1467-9868.2011.00771.x
14. Pons P, Latapy M (2018) Computing communities in large networks using random walks. In: Yolum GT, Gürgen F, Özturan C (eds) Computer and information sciences—ISCIS 2005. Lect notes comput sci, vol 3733. SpringerLink, Berlin, Heidelberg, pp 284–293. https://doi.org/10.1007/11569596_31
15. Dongen SV (2008) Graph Clustering Via a Discrete Uncoupling Process. SIAM J Matrix Anal Appl 30:121–141. https://doi.org/10.1137/040608635
16. Freeman LC (1978) Centrality in social networks conceptual clarification. Soc Netw 1:215–239. https://doi.org/10.1016/0378-8733(78)90021-7
17. Brandes U (2001) A faster algorithm for betweenness centrality. J Math Sociol 25:163–177. https://doi.org/10.1080/0022250X.2001.9990249
18. Brin S, Page L (1998) The anatomy of a large-scale hypertextual web search engine. In: Comput netw ISDN syst, vol 1–7. Elsevier Science Publishers, Brisbane, Australia, pp 107–117. https://doi.org/10.1016/S0169-7552(98)00110-X
19. McDonald D, Birmingham A, Knight R (2015) Context and the human microbiome. Microbiome 3:52. https://doi.org/10.1186/s40168-015-0117-2