GeneBridge: Inferring the Evolutionary Rooting of Orthologous Genes

Dalmolin Systems Biology Group

Table of contents

1 GeneBridge

2 Lesson overview

Target Audience: Researchers, graduate students, and biologists interested in evolutionary analysis.

Level: Intermediate

Prerequisites To be able to follow this course, learners should have knowledge in:

1. Basic knowledge of the R programming language (loading libraries, handling data frames).
2. Familiarity with fundamental concepts in evolutionary biology (e.g., genes, orthologs, phylogenetic trees).
3. Being comfortable working with RStudio or a similar R environment.

Description This repository contains the materials for the course "GeneBridge: Inferring the Evolutionary Rooting of Orthologous Genes," organized by Dalmolin's Systems Biology Group, based at the Bioinformatics Multidisciplinary Environment (BioME) at UFRN. This workshop is divided into three parts: (1) Performing the Rooting Analysis, (2) Plotting Rooted Genes, and (3) Plotting Abundances.

Learning Outcomes: By the end of the course, learners will be able to:

1. **Understand** the theoretical basis of evolutionary rooting and the Bridge algorithm. [Understanding]
2. **Apply** the GeneBridge package to infer the evolutionary root for a set of orthologous genes. [Applying]
3. **Analyze** and **interpret** the output tables and rooting results. [Analysing]
4. **Create** and **customize** plots to visualize rooted genes on a phylogenetic tree. [Creating]
5. **Generate** and **evaluate** plots of gene abundances across different clades. [Evaluating]

Time estimation: 180 minutes

Requirements: Participants must have a laptop with a recent version of R and RStudio installed. Specific R package dependencies (e.g., GeneBridge) are detailed in the repository's setup instructions.

Acknowledgements:

- Bioinformatics Multidisciplinary Environment (BioME - IMD/UFRN)
- Postgraduate Program in Bioinformatics (PPg-Bioinfo - UFRN)

3 Gene Rooting

3.1 Importing Packages

```
library(GeneBridge)
library(geneplast.data)
library(readr)
library(dplyr)
library(purrr)
library(biomaRt)
library(magrittr)
library(KEGGREST)
library(ape)
library(tidyverse)
library(data.table)
library(stringi)
library(AnnotationHub)
library(sourcetools)
library(here)
```

3.2 Defining functions for later use

The first one searches for the respective protein ID for each gene in the input list. The second returns the interactions between these proteins, and the last one filters the interactions by a combined confidence score greater than 0.4.

```
# get IDs from STRING DB
get_string_ids <- function(genes_hgnc, species_id = "9606") {

    req <- RCurl::postForm(
    "https://string-db.org/api/tsv/get_string_ids",
    identifiers = paste(genes_hgnc, collapse = "%0D"),
    echo_query = "1",
```

```
    species = species_id,
    .opts = list(ssl.verifypeer = FALSE)
  )

  map_ids <- read.table(text = req, sep = " ", header = TRUE, quote = "") %>%
    dplyr::select(-queryIndex) %>%
    unique()

  map_ids$stringId <- substring(map_ids$stringId, 6, 1000)

  return(map_ids)
}

 # Get STRING interactions
 get_network_interaction <- function(map_ids, protein_id, species_id =
 ↪  "9606") {

   identifiers <- map_ids %>% pull(protein_id) %>% na.omit %>%
 ↪  paste0(collapse="%0d")

   req2 <- RCurl::postForm(
     "https://string-db.org/api/tsv/network",
     identifiers = identifiers,
     required_core = "0",
     species = species_id,
     .opts = list(ssl.verifypeer = FALSE)
   )

   int_network <- read.table(text = req2, sep = "   ", header = TRUE)

   int_network <- unique(int_network)

   return(int_network)
 }

 ## Recomputing scores
combine_scores <- function(dat, evidences = "all", confLevel = 0.4) {
  if(evidences[1] == "all"){
    edat<-dat[,-c(1,2,ncol(dat))]
  } else {
    if(!all(evidences%in%colnames(dat))){
      stop("NOTE: one or more 'evidences' not listed in 'dat' colnames!")
    }
```

```
    edat<-dat[,evidences]
  }
  if (any(edat > 1)) {
    edat <- edat/1000
  }
  edat<-1-edat
  sc<- apply(X = edat, MARGIN = 1, FUN = function(x) 1-prod(x))
  dat <- cbind(dat[,c(1,2)],combined_score = sc)
  idx <- dat$combined_score >= confLevel
  dat <-dat[idx,]
  return(dat)
}
```

3.3 Loading gene list and orthology data

We load the orthology data through **AnnotationHub**, this R package provides a central place where genomic files (VCF, bed, wig) and other resources from standard locations (e.g., UCSC, Ensembl) can be accessed. This way, we have access to the input files for the GeneBridge algorithm.

```
# Load the Gene Set Table
sensorial_genes <- read.csv(here("data/sensorial_genes.csv"))

# Query Phylotree and OG data
ah <- AnnotationHub()
meta <- query(ah, "geneplast")
load(meta[["AH83116"]])

head(sensorial_genes)
head(cogdata)
```

3.4 1. Pre-processing

3.4.1 1.1. Mapping

For the next analyses, we need to cross-reference information between our genes of interest (Gene IDs from the **`sensorial_genes`** table) and Protein IDs (from the **`cogdata`** table). The STRINGdb API is used to map Gene IDs to Protein IDs, allowing the filtering of genes of

interest in the cogdata table. The final goal is to obtain a filtered set of sensory genes with their respective pathways and COG IDs.

```
map_ids <- get_string_ids(sensorial_genes$gene_symbol)

# Subsetting cogs of interest - Sensorial Genes
gene_cogs <- cogdata %>%
  filter(ssp_id %in% map_ids$ncbiTaxonId) %>%
  filter(protein_id %in% map_ids[["stringId"]]) %>%
  group_by(protein_id) %>%
  summarise(n = n(), cog_id = paste(cog_id, collapse = " / "))

head(map_ids)

#map_ids |>
#  vroom::vroom_write(file = here("data/map_ids.csv"), delim = ",")
```

3.4.2 1.2. Resolving duplicate COGs

Due to evolutionary events such as gene duplication, some genes may be associated with more than one Cluster of Orthologous Groups (COG). To ensure the functionality of the algorithm, it is necessary to resolve these cases, prioritizing COGs according to the following criteria:

1. Priority by COG type:

- KOGs have higher priority.
- COGs have higher priority than NOGs.

2. Cases with COGs starting with the same letter:

- Are resolved manually, based on the annotated function of the COG and the scientific question of the study.

The code below implements this resolution and integrates the corrections into the main table.

```
gene_cogs %>% filter(n > 1)

# Resolving main proteins
gene_cogs_resolved <- tribble(
  ~protein_id, ~cog_id,
"ENSP00000332500", "NOG274749", #NOG274749 / NOG274749
"ENSP00000409316", "NOG282909", #NOG282909 / NOG282909 / NOG282909
"ENSP00000480090", "KOG3599"    #KOG3599 / KOG3272
```

```
)

# Removing unresolved cases and adding manual assignments
gene_cogs %<>%
  filter(n == 1) %>%
 dplyr:: select(-n) %>%
  bind_rows(gene_cogs_resolved)

#gene_cogs |>
#  vroom::vroom_write(file = here("data/gene_cogs.csv"), delim = ",")
```

3.5 3. Processing

The objective of this step is to perform the rooting of the genes of interest using the **GeneBridge** package. For this, we use the `newBridge`, `runBridge`, and `runPermutation` functions, which produce statistical results associated with the selected COGs in a phylogenetic tree.

3.5.1 3.1. Necessary Inputs

1. **`ogdata`**:
   - Dataset containing three main columns:
     - `Protein ID`: Protein identifiers.
     - `COG ID`: Clusters of interest.
     - `Specie ID`: Species identifiers.
   - In the example, the `cogdata` object is being used.
2. **`phyloTree`**:
   - Phylogenetic tree containing 476 eukaryotes, representing the evolutionary structure among the analyzed species.
3. **`ogids`**:
   - List of **COGs of interest**. This set is derived from the `gene_cogs` table and includes the COGs associated with the proteins after the previous processing.
4. **`refsp`**:
   - Reference species for rooting. In the example, we use `9606` (human).

The *getBridge* function extracts the results generated by GeneBridge in table format. The *res* table contains the statistical results of the rooting.

```
## Run GeneBridge
cogs_of_interest <- gene_cogs %>% pull(cog_id) %>% unique

ogr <- newBridge(ogdata=cogdata, phyloTree=phyloTree, ogids =
 ↪  cogs_of_interest, refsp="9606")

ogr <- runBridge(ogr, penalty = 2, threshold = 0.5, verbose = TRUE)

ogr <- runPermutation(ogr, nPermutations=1000, verbose=FALSE)

res <- getBridge(ogr, what="results")

saveRDS(ogr, file = here("data/ogr.RData"))
```

3.6 4. Post-processing

After performing the rooting with **GeneBridge**, it is necessary to adjust the data to improve the visualization and interpretation of the results. In this step, we add the names of the clades to the identified roots, using an external table that relates the root identifiers to the clade names.

```
# naming the rooted clades
CLADE_NAMES <-
 ↪  "https://raw.githubusercontent.com/dalmolingroup/neurotransmissionevolution/ctenophora_be

lca_names <- vroom::vroom(CLADE_NAMES)

groot_df <- res %>%
  tibble::rownames_to_column("cog_id") %>%
  dplyr::select(cog_id, root = Root) %>%
  left_join(lca_names) %>%
  inner_join(gene_cogs)

head(groot_df)

#groot_df |>
#  vroom::vroom_write(file = here("data/groot_df.csv"), delim = ",")
```

3.6.1 4.1. Protein-Protein Interaction Network

The construction of a protein-protein interaction (PPI) network is an essential step to identify the functional relationships between proteins. In this process, we use the **STRINGdb** API, a database that catalogs interactions between proteins based on various sources, including experimental assays, co-expression, and evidence extracted from scientific publications.

The STRINGdb API offers methods to: - Obtain protein interactions for a list of proteins. - Select specific sources of evidence. - Calculate and combine scores based on the selected evidence.

More information about the API can be found in the STRING API documentation.

```
# Get proteins interaction
string_edgelist <- get_network_interaction(groot_df)

# Recomputing scores
string_edgelist <- combine_scores(string_edgelist,
                                  evidences = c("ascore", "escore",
                                   ↪  "dscore"),
                                  confLevel = 0.7)

colnames(string_edgelist) <- c("stringId_A", "stringId_B", "combined_score")

# Remove the species id
string_edgelist$stringId_A <- substring(string_edgelist$stringId_A, 6, 1000)
string_edgelist$stringId_B <- substring(string_edgelist$stringId_B, 6, 1000)

# How many edgelist proteins are absent in gene_ids? (should return 0)
setdiff(
  string_edgelist %$% c(stringId_A, stringId_B),
  map_ids %>% pull(stringId)
)

head(string_edgelist)
```

For the construction of the graph, in addition to the interactions between the proteins, it is necessary that each node is annotated with additional information that will be used in the analysis, such as: - Protein name. - Clade where it is rooted. - Metabolic pathway in which it participates.

```
## Create anotation table
nodelist <- data.frame(node = unique(c(string_edgelist$stringId_A,
 ↪  string_edgelist$stringId_B)))
```

```
merged_paths <- merge(nodelist, groot_df, by.x = "node", by.y = "protein_id")

pivotada <- sensorial_genes %>%
  dplyr::select(gene_symbol, pathway_name) %>%
  dplyr::mutate(n = 1) %>%
  tidyr::pivot_wider(
    id_cols = gene_symbol,
    names_from = pathway_name,
    values_from = n,
    values_fn = list(n = length),
    values_fill = list(n = 0),
  )

source_statements <-
  colnames(pivotada)[2:length(pivotada)]

nodelist <-
  nodelist %>%
  left_join(merged_paths, by = c("node" = "node")) %>%
  left_join(map_ids, by = c("node" = "stringId")) %>%
  left_join(pivotada, by = c("queryItem" = "gene_symbol"))

head(nodelist)
```

In addition to the graph structure, we can calculate metrics such as the number of connections (degree) of each node.

```
# Network Metrics
connected_nodes <- rle(sort(c(string_edgelist[,1], string_edgelist[,2])))
connected_nodes <- data.frame(count=connected_nodes$lengths,
 ↪  node=connected_nodes$values)
connected_nodes <- left_join(nodelist, connected_nodes, by = c("node" =
 ↪  "node"))
connected_nodes <- dplyr::select(connected_nodes, queryItem, root,
 ↪  clade_name, count)

head(connected_nodes)
```

```
#nodelist |>
#  vroom::vroom_write(file = here("data/nodelist.csv"), delim = ",")
```

```
#string_edgelist |>
#  vroom::vroom_write(file = here("data/string_edgelist.csv"), delim = ",")
#merged_paths |>
#  vroom::vroom_write(file = here("data/merged_paths.csv"), delim = ",")
```

4 Plotting Roots

4.1 Import libraries

```
library(ggplot2)
library(ggraph)
library(dplyr)
library(tidyr)
library(igraph)
library(purrr)
library(vroom)
library(paletteer)
library(easylayout)
library(UpSetR)
library(tinter)
library(here)
library(dplyr)
```

4.2 Define functions

```
# Set colors
color_mappings <- c(
  "Olfactory transduction"   = "#8dd3c7",
  "Taste transduction"      = "#72874EFF",
  "Phototransduction"       = "#fb8072"
)

subset_graph_by_root <-
  function(geneplast_result, root_number, graph) {
    filtered <- geneplast_result %>%
      filter(root >= root_number) %>%
      pull(node)
```

```
    induced_subgraph(graph, which(V(graph)$name %in% filtered))
  }

adjust_color_by_root <- function(geneplast_result, root_number, graph) {
  filtered <- geneplast_result %>%
    filter(root == root_number) %>%
    pull(node)

  V(graph)$color <- ifelse(V(graph)$name %in% filtered, "black", "gray")
  return(graph)
}

# Configure graph colors by genes incrementation
subset_and_adjust_color_by_root <- function(geneplast_result, root_number,
 ↪  graph) {
  subgraph <- subset_graph_by_root(geneplast_result, root_number, graph)
  adjusted_graph <- adjust_color_by_root(geneplast_result, root_number,
 ↪  subgraph)
  return(adjusted_graph)
}

plot_network <- function(graph, title, nodelist, xlims, ylims, legend =
 ↪  "none") {

  # Generate color map
  source_statements <-
    colnames(nodelist)[10:length(nodelist)]

  color_mappings <- c(
    "Olfactory transduction"   = "#8dd3c7",
    "Taste transduction"      = "#72874EFF",
    "Phototransduction"       = "#fb8072"
  )

  vertices <- igraph::as_data_frame(graph, "vertices")

  ggraph:: ggraph(graph,
                  "manual",
                  x = V(graph)$x,
                  y = V(graph)$y) +
    ggraph::geom_edge_link0(edge_width = 1, color = "#90909020") +
    ggraph::geom_node_point(ggplot2::aes(color = I(V(graph)$color)), size =
 ↪  2) +
```

```
    scatterpie::geom_scatterpie(
      aes(x=x, y=y, r=18),
      cols = source_statements,
      data = vertices[rownames(vertices) %in% V(graph)$name[V(graph)$color ==
       ↪  "black"],],
      colour = NA,
      pie_scale = 1
    ) +
    geom_node_text(aes(label = ifelse(V(graph)$color == "black",
     ↪  V(graph)$queryItem, NA)),
                   nudge_x = 1, nudge_y = 1, size = 0.5, colour = "black") +
    ggplot2::scale_fill_manual(values = color_mappings, drop = FALSE) +
    ggplot2::coord_fixed() +
    ggplot2::scale_x_continuous(limits = xlims) +
    ggplot2::scale_y_continuous(limits = ylims) +
    ggplot2::theme_void() +
    ggplot2::theme(
      legend.position = legend,
      legend.key.size = ggplot2::unit(0.5, 'cm'),
      legend.key.height = ggplot2::unit(0.5, 'cm'),
      legend.key.width = ggplot2::unit(0.5, 'cm'),
      legend.title = ggplot2::element_text(size=6),
      legend.text = ggplot2::element_text(size=6),
      panel.border = ggplot2::element_rect(
        colour = "#161616",
        fill = NA,
        linewidth = 1
      ),
      plot.title = ggplot2::element_text(size = 8, face = "bold")
    ) +
    ggplot2::guides(
      color = "none",
      fill = "none"
    ) +
    ggplot2::labs(fill = "Source:", title = title)
}
```

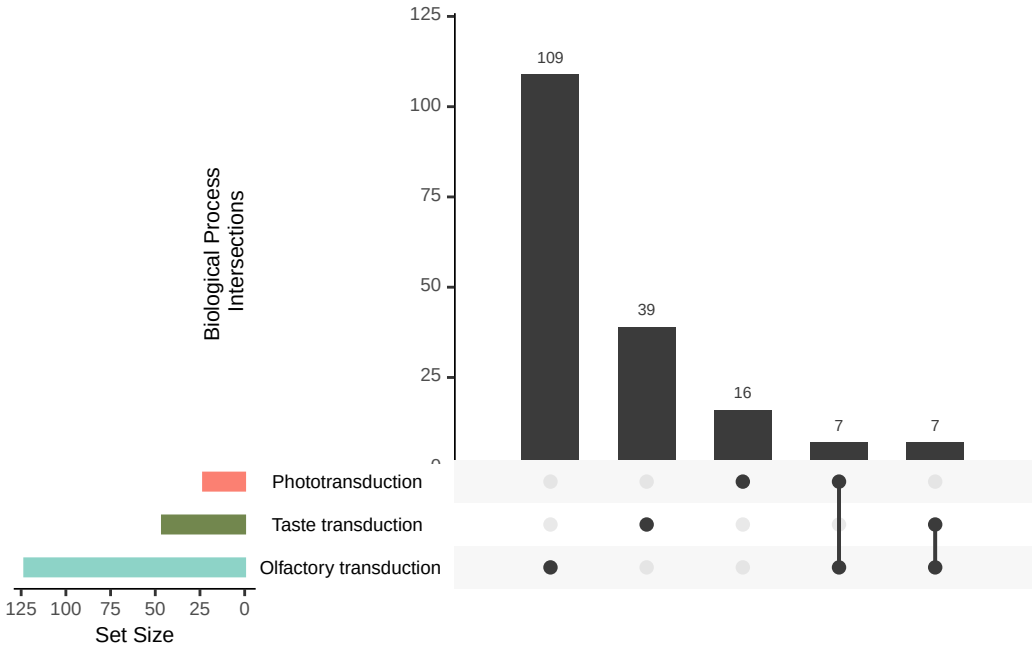
4.3 Loading necessary tables

```
#Load data (need to save tables from first qmd)
nodelist <- vroom::vroom(file = here("data/nodelist.csv"), delim = ",")
string_edgelist <- vroom::vroom(file = here("data/string_edgelist.csv"),
 ↪  delim = ",")
merged_paths <- vroom::vroom(file = here("data/merged_paths.csv"), delim =
 ↪  ",")
```

4.4 1. Visualization with UpSet Plot

The *UpSet Plot* is a useful tool to visualize the distribution and concatenation of genes between different metabolic pathways. It allows identifying how genes are shared or exclusive among the analyzed categories.

```
upset(dplyr::select(as.data.frame(nodelist),
                            "Olfactory transduction",
                            "Taste transduction",
                            "Phototransduction"),
                    nsets = 50, nintersects = NA,
      sets.bar.color = c("#8dd3c7", "#72874EFF", "#fb8072"),
      mainbar.y.label = "Biological Process \nIntersections",
      sets.x.label = "Set Size")
```



4.5 2. Visualization of the Protein-Protein Interaction Network

The visualization of the interaction network is essential to understand the functional connections between proteins. Here, we use the **easylayout** package, developed by Danilo Imparato, to generate an efficient layout. This package organizes the network nodes in x and y coordinates, allowing a structured and clear visualization. Subsequently, the graph will be plotted with **ggraph**.

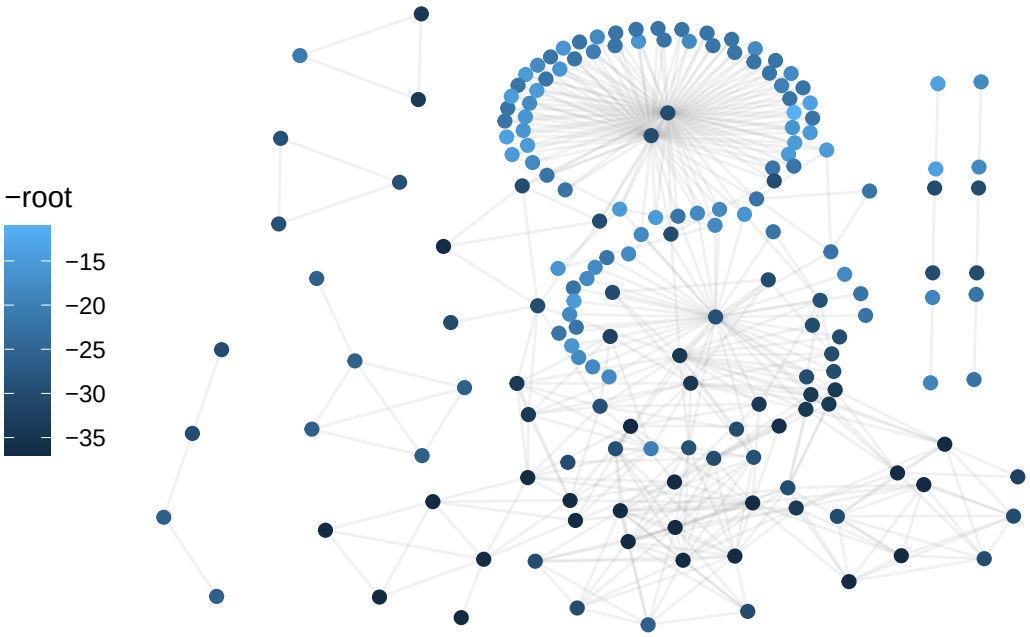
```
## Graph Build
#graph <-
#  graph_from_data_frame(string_edgelist, directed = FALSE, vertices =
 ↪  nodelist)

#layout <- easylayout::easylayout(graph)
#V(graph)$x <- layout[, 1]
#V(graph)$y <- layout[, 2]

#save(graph, file = "../data/graph_layout")
```

4.5.1 2.1. Visualization of the Ancestry of Each Node

The analysis of the ancestry of each node in the network provides an evolutionary view of the analyzed proteins. Here, we use **ggraph** to plot the graph with the positions previously saved by *easylayout*.

The nodes are colored according to the distance from the last common ancestor (LCA) of the analyzed clades and the human (*Human-LCA*). The darker shade indicates older clades in relation to humans, while light shades of blue represent newer clades, closer to the *Human-LCA*.

```
load(here("data/graph_layout"))

ggraph(graph, "manual", x = V(graph)$x, y = V(graph)$y) +
  geom_edge_link0(color = "#90909020") +
  geom_node_point(aes(color = -root), size = 2) +
  theme_void() +
  theme(legend.position = "left")
```



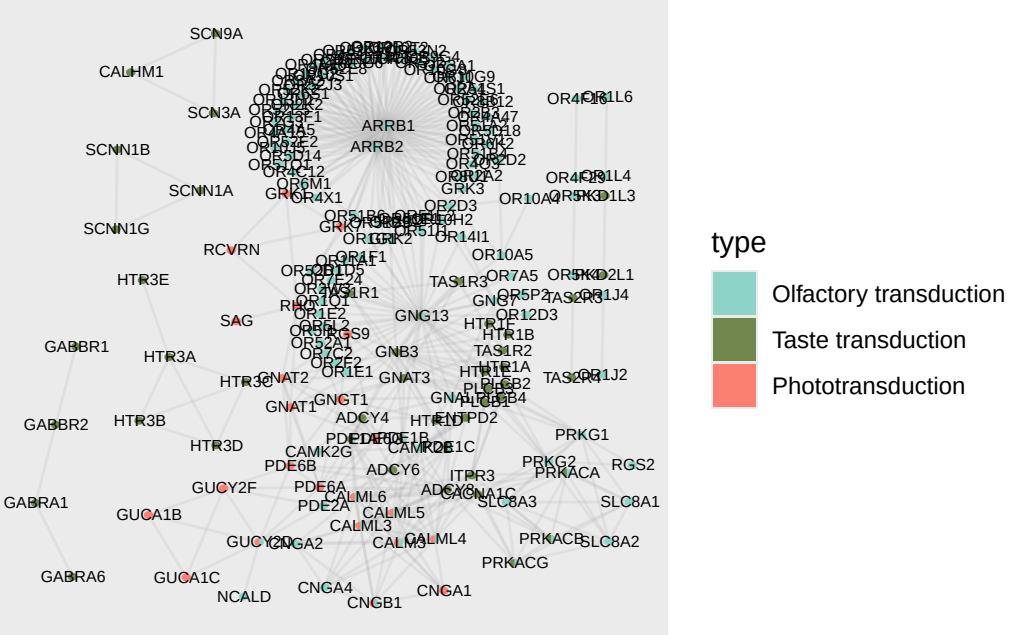
4.5.2 2.2. Visualization of the Protein-Protein Interaction Network in Human

To better understand the relationship between human proteins, we plot the interaction network where the nodes represent the human genes associated with their biological processes.

4.5.2.1 Description of the graph elements:

1. **Nodes (Circles):** The colors of the nodes are divided according to the biological processes assigned to each gene. The use of pie charts allows the visualization of genes that participate in multiple processes.
2. **Edges (Lines):** Represent the protein interactions based on data from STRINGdb.
3. **Gene labels:** Each node is annotated with the corresponding gene symbol, strategically positioned for easy reading.

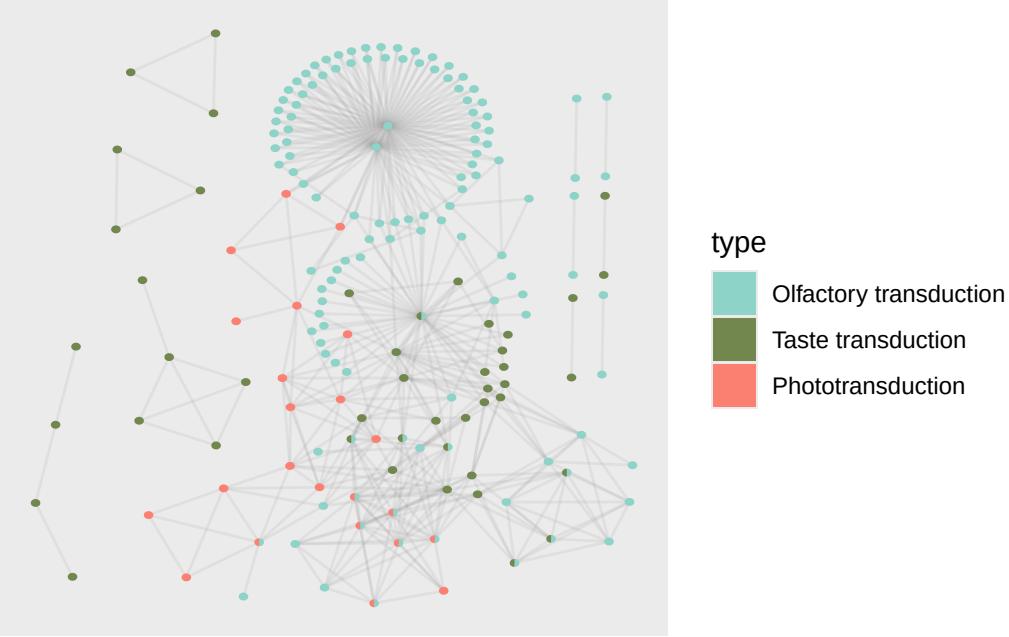
```
## Plotting Human PPI Network
#ppi_labaled <-
ggraph::ggraph(graph,
               "manual",
               x = V(graph)$x,
               y = V(graph)$y) +
  ggraph:: geom_edge_link0(edge_width = 0.5, color = "#90909020") +
  scatterpie::geom_scatterpie(
    cols = colnames(nodelist[10:12]),
    data = igraph::as_data_frame(graph, "vertices"),
    colour = NA,
    pie_scale = 0.40
  ) +
  geom_node_text(aes(label = nodelist$queryItem), colour = "black", nudge_x =
   ↪  0.8, nudge_y = 0.8, size = 2) +
  ggplot2::scale_fill_manual(values = color_mappings, drop = FALSE)
```



type

- Olfactory transduction
- Taste transduction
- Phototransduction

```
#ppi <-
  ggraph::ggraph(graph,
                 "manual",
                 x = V(graph)$x,
                 y = V(graph)$y) +
  ggraph:: geom_edge_link0(edge_width = 0.5, color = "#90909020") +
  scatterpie::geom_scatterpie(
    cols = colnames(nodelist[10:12]),
    data = igraph::as_data_frame(graph, "vertices"),
    colour = NA,
    pie_scale = 0.40
  ) +
  ggplot2::scale_fill_manual(values = color_mappings, drop = FALSE)
```



4.5.3 2.3. Visualization of the Protein-Protein Interaction Network in Each Clade

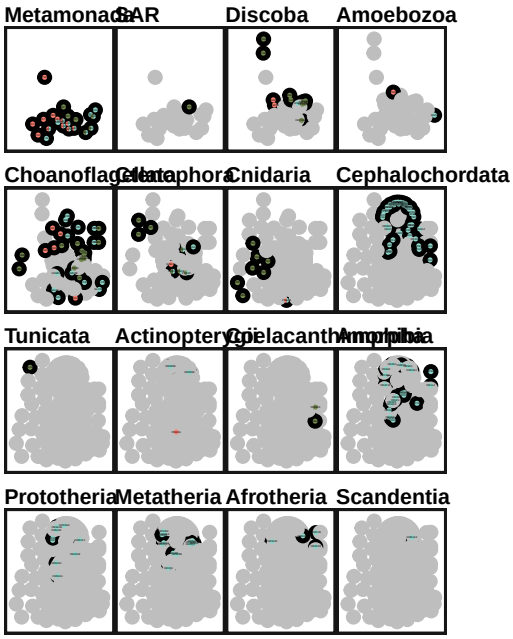
In this section, we visualize the genes that are statistically rooted in each clade. The arrangement of the genes allows us to observe the increment of orthologous genes as a function of the complexity and antiquity of the biological system.

4.5.3.1 Visualization features:

1. **Evolution of the graphs:** The graphs are organized from left to right and from top to bottom, allowing the analysis of the evolutionary progression.
2. **Node coloring:** The color of the nodes indicates the level of ancestry, as previously highlighted, where darker tones represent older clades and lighter tones indicate evolutionary proximity to humans.
3. **Organisms of interest:** In addition to visualizing all clades, it is possible to generate graphs focused only on certain groups, such as *Metamonada*, *Choanoflagellata*, *Cephalochordata*, and *Amphibia*.

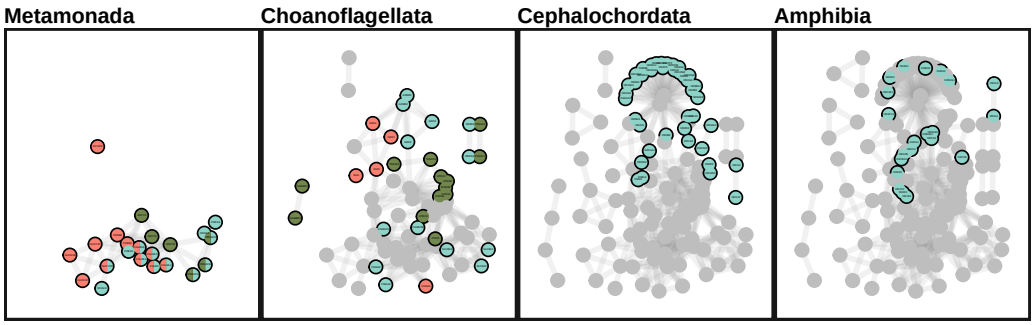
With these visualizations, it is possible to identify patterns of gene evolution in different clades and perform detailed comparisons with organisms of specific interest.

```r
geneplast_roots <- merged_paths[order(merged_paths$root), ]


buffer <- c(-50, 50)
xlims <- ceiling(range(V(graph)$x)) + buffer
ylims <- ceiling(range(V(graph)$y)) + buffer

roots <- unique(geneplast_roots$root) %>%
  set_names(unique(geneplast_roots$clade_name))

# Subset graphs by LCAs
subsets <-
  map(roots, ~ subset_and_adjust_color_by_root(geneplast_roots, .x, graph))

# Plot titles
titles <- names(roots)

plots <-
  map2(
    subsets,
    titles,
    plot_network,
    nodelist = nodelist,
    xlims = xlims,
    ylims = ylims,
    legend = "right"
  ) %>%
  discard(is.null)

#net_all_roots <-
patchwork::wrap_plots(
  rev(plots),
  nrow = 4,
  ncol = 4
)
```




```
#ggsave(file = "../data/network_rooting.svg", plot=net_all_roots, width=10,
 ↪  height=8)
```

```
patchwork::wrap_plots(
  plots$Metamonada, plots$Choanoflagellata, plots$Cephalochordata,
 ↪  plots$Amphibia,
  ncol = 4
)
```


Metamonada
Choanoflagellata
Cephalochordata
Amphibia

5 Plotting Abundance

5.1 Importing Packages

```
library(here)
library(readr)
library(magrittr)
library(ggplot2)
library(hrbrthemes)
library(tinter)
library(dplyr)
library(tidyr)
library(AnnotationHub)
```

5.2 Defining functions

These functions are intended to organize the annotation of the vertices of our nodelist and calculate the cumulative number of genes per clade and per biological process.

```
calculate_cumulative_genes <- function(nodelist) {

    # Get all possible categories of clade_name
  all_clades <- node_annotation %>%
    arrange(desc(root)) %>%
    dplyr:: select(clade_name) %>%
    unique()

  # Define the columns of interest
  process_columns <- c("queryItem", "root", "clade_name",
                       "Olfactory transduction",
                       "Taste transduction",
                       "Phototransduction")
```

```
  # Calculate the cumulative sum grouping by clade_name
  cumulative_genes <- nodelist %>%
    arrange(desc(root)) %>%
    dplyr::select(all_of(process_columns)) %>%
    group_by(clade_name, root) %>%
    summarise(count_genes = n(), .groups = "drop") %>%
    arrange(desc(root)) %>%
    mutate(cumulative_sum = cumsum(count_genes)) %>%
    right_join(all_clades, by = "clade_name") %>%
    fill(cumulative_sum, .direction = "down")

  return(cumulative_genes)
}

calculate_cumulative_bp <- function(nodelist) {

  # Get all possible categories of clade_name
  all_clades <- node_annotation %>%
    arrange(desc(root)) %>%
    dplyr:: select(clade_name) %>%
    unique()

   # Define the columns of interest
  process_columns <- c("queryItem", "root", "clade_name",
                       "Olfactory transduction",
                       "Taste transduction",
                       "Phototransduction")

  # Calculate the cumulative sum for each biological process
  cumulative_bp <- nodelist %>%
    dplyr::select(all_of(process_columns)) %>%
    distinct(root, queryItem, .keep_all = TRUE) %>%
    mutate(across(all_of(process_columns[-c(1:3)]), ~ as.numeric(.))) %>%
    group_by(root, clade_name) %>%
    summarise(across(all_of(process_columns[-c(1:3)]),
                     ~ sum(. , na.rm = TRUE)),
              .groups = "drop") %>%
    arrange(desc(root)) %>%
    mutate(across(all_of(process_columns[-c(1:3)]), ~ cumsum(.))) %>%
    right_join(all_clades, by = "clade_name") %>%
    fill(everything(), .direction = "down")

  return(cumulative_bp)
```

```
}
```

5.3 Defining aesthetic parameters for ggplot

Assigning default colors for the metabolic pathways in the analysis and defining the other aesthetic standards for plotting.

```
# Plotting colors and labels
annotation_colors <- c(
  "Olfactory transduction"   = "#8dd3c7",
  "Taste transduction"      = "#72874EFF",
  "Phototransduction"       = "#fb8072"
)

annotation_labels <- c(
  "Olfactory transduction"   = "Olfactory transduction",
  "Taste transduction"      = "Taste transduction",
  "Phototransduction"       = "Phototransduction"
)

# This vertical line indicates the first metazoan (Amphimedon queenslandica /
 ↪  Ctenophora)
choanoflagellata_line <- geom_vline(
  xintercept = "Sphaeroforma arctica",
  color      = "#FF0000",
  linetype   = "11",
  alpha      = 1,
  linewidth  = 0.25
)

# Plotting
theme_main <- theme(
  panel.spacing      = unit(2.5, "pt"),
  strip.background   = element_blank(),
  panel.grid.major.x = element_blank(),
  panel.grid.major.y = element_line(linewidth = 0.25, linetype = "dotted",
   ↪  color = "#E0E0E0"),
  strip.text.x       = element_text(size = 9, angle = 90, hjust = 0, vjust =
   ↪  0.5, color = "#757575"),
  strip.text.y       = element_text(size = 10, angle = 0, hjust = 0, vjust =
   ↪  0.5, color = "#757575"),
```

```
  axis.title         = element_text(size = 15, color = "#424242"),
  axis.ticks.x       = element_blank(),
  axis.text.x        = element_blank(),
  axis.text.y        = element_text(size = 5.5),
  legend.position    = "none"
)

theme_supplementary <- theme(
  panel.grid.major.x = element_line(color = "#E0E0E0", linewidth = 0.25,
   ↪  linetype = "dotted"),
  panel.grid.major.y = element_blank(),
  strip.text.y       = element_text(size = 7, angle = 0, hjust = 0, vjust =
   ↪  0.5, color = "#757575"),
  strip.text.x       = element_text(size = 7, angle = 90, hjust = 0, vjust =
   ↪  0.5, color = "#757575"),
  axis.title         = element_text(size = 12, color = "#424242"),
  axis.ticks         = element_line(colour = "grey20"),
  axis.text.y        = element_text(size = 6, angle = 0, hjust = 1, vjust =
   ↪  0.5, color = "#757575"),
  axis.text.x        = element_text(size = 6)
)

theme_average <- theme(
  panel.spacing      = unit(1, "pt"),
  axis.title         = element_text(color = "#424242"),
  axis.text          = element_text(color = "#757575"),
  axis.text.x        = element_text(size = 7, angle = -45, vjust = 0, hjust =
   ↪  0),
  axis.text.y        = element_text(size = 5),
  strip.background   = element_blank(),
  strip.text         = element_text(color = "#757575"),
  strip.text.y       = element_text(angle = 0, hjust = 0, vjust = 0.5)
)

theme_big <- theme(
  panel.spacing      = unit(0.5, "pt"),
  panel.grid.major.x = element_line(linewidth = 0.1, linetype = "dashed"),
  panel.grid.major.y = element_blank(),
  strip.background   = element_blank(),
  strip.text.x       = element_text(size = 8, angle = 90, hjust = 0.5, vjust
   ↪  = 0),
  strip.text.y       = element_text(size = 8, angle = 0, hjust = 0, vjust =
   ↪  0.5),
```

```
  axis.text.x        = element_text(size = 6, angle = 90, vjust = 0, hjust =
   ↪  0),
  axis.text.y        = element_text(size = 4.5),
  axis.ticks         = element_line(size = 0.1)
)

tick_function <- function(x) {
  seq(x[2], 0, length.out = 3) %>% head(-1) %>% tail(-1) %>% { ceiling(./5)*5
   ↪  }
}
```

5.4 Load necessary tables

To proceed with the analysis, it is necessary to load a table containing the *taxid* information for each species. This information will be used later to map the *taxid* of the species with the *taxid* of each COG.

5.4.1 Table Description

- **File Name**: `string_eukaryotes.rda`
- **Content**: Taxonomic information of eukaryotic species.

The table loads the following main fields: - **TaxID** (*taxonomic identifier*): A unique identifier associated with each species. - Species name

```
# Query Phylotree and OG data
ah <- AnnotationHub()
meta <- query(ah, "geneplast")
load(meta[["AH83116"]])

# TODO: save tables from previous qmds
nodelist <- vroom::vroom(file = here("data/nodelist.csv"), delim = ",")
gene_cogs <-  vroom::vroom(file = here("data/gene_cogs.csv"), delim = ",")
sensorial_genes <- read.csv(here("data/sensorial_genes.csv"))
map_ids <- vroom::vroom(here("data/map_ids.csv"))
groot_df <- vroom::vroom(here("data/groot_df.csv"))

ogr <- readRDS(here("data/ogr.RData"))

load(here("data/string_eukaryotes.rda"))
```

```r
head(string_eukaryotes)
```

5.5 Visualization: Rooting Pattern

This visualization allows analyzing the cumulative pattern of gene emergence over evolutionary time. In addition, it also shows the growth of biological processes separated by each metabolic pathway.

5.5.1 Graph Characteristics

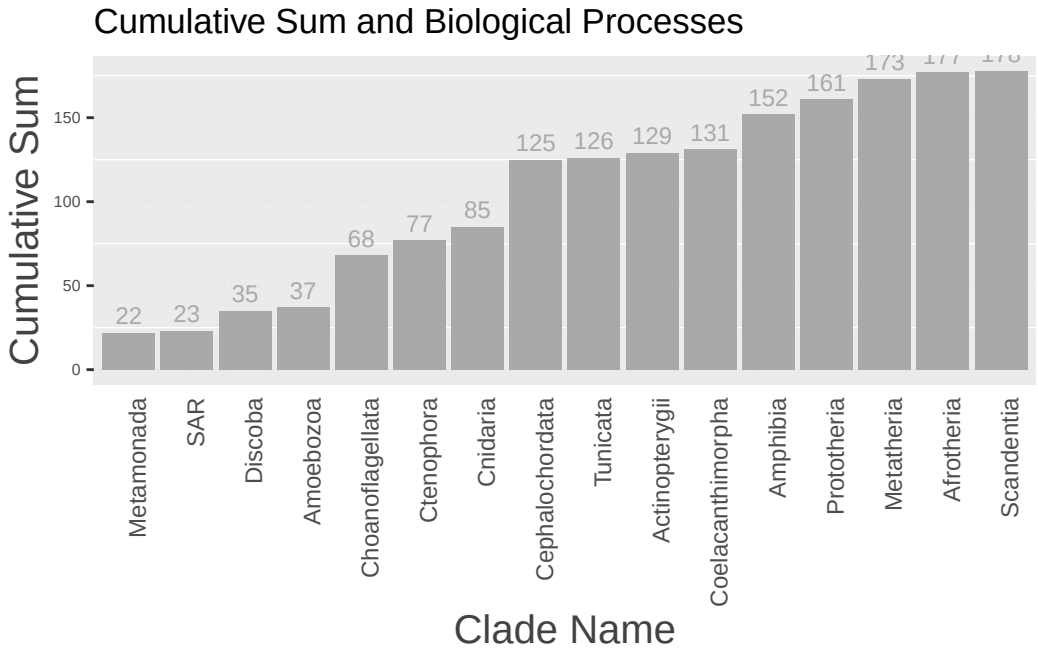
1. **Bar Chart (Cumulative Pattern)**:
    - Represents the cumulative sum of genes associated with each clade.
    - Each bar displays the total accumulated genes in a given clade, allowing the identification of the diversification pattern.
2. **Combined Chart (Bars and Lines)**:
    - The bars indicate the cumulative sum of genes per clade.
    - The lines represent the growth of different biological processes (*Process*), separated by metabolic pathways, allowing the comparison between the cumulative emergence of genes and the development of biological functions.

5.5.2 Interpretation

- **Cumulative Pattern**: The bar chart shows how genes have emerged over time, cumulatively. This view helps to identify moments of greater genetic diversification.
- **Comparison by Biological Processes**: The combined chart allows observing which biological processes stand out at different moments of evolution and how they are related to the emergence of new genes.

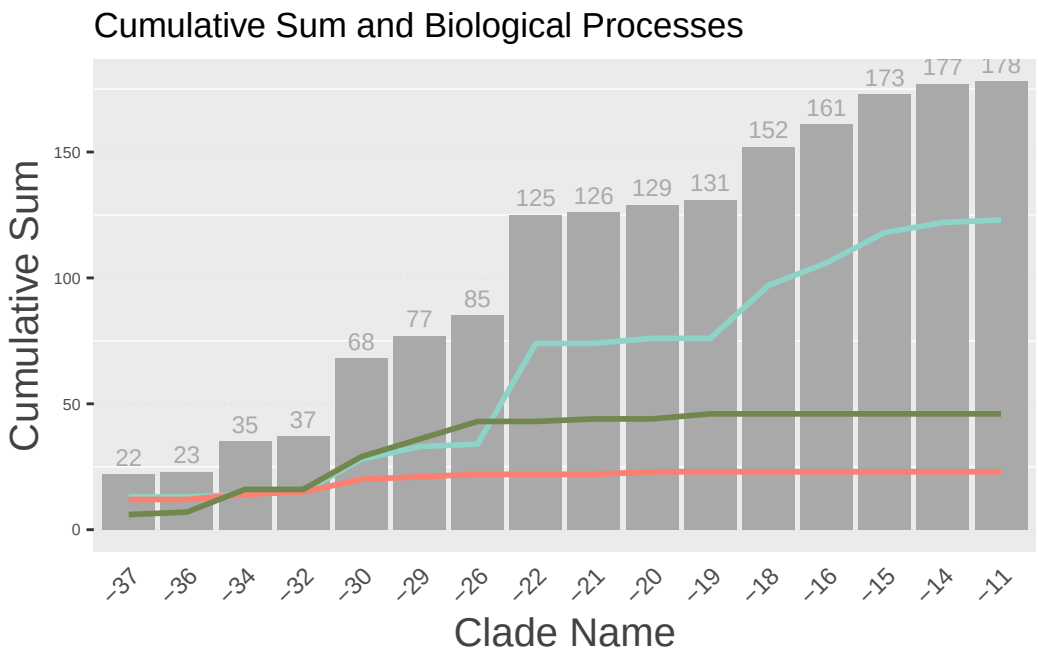
```r
# Mapping roots and proteins info
node_annotation <- nodelist %>%
  inner_join(gene_cogs, by = c("node" = "protein_id", "cog_id")) %>%
  inner_join(sensorial_genes, by = c("queryItem" = "gene_symbol")) %>%
  distinct(queryItem, cog_id, pathway_name, root, clade_name)

cumulative_genes <- calculate_cumulative_genes(nodelist)
cumulative_bp <- calculate_cumulative_bp(nodelist)
```

```r
cumulative_data <- left_join(cumulative_genes, cumulative_bp)


long_data <- cumulative_data %>%
  pivot_longer(cols = 5:7,
               names_to = "Process",
               values_to = "Value")

#a <-
ggplot() +
  # Bar chart for cumulative_sum
  geom_bar(data = cumulative_data,
           aes(x = factor(clade_name, levels = clade_name), y =
            ↪  cumulative_sum),
           stat = "identity", fill = "darkgray", colour = NA) +
  geom_text(data = cumulative_data,
            aes(x = factor(clade_name, levels = clade_name), y =
             ↪  cumulative_sum, label = cumulative_sum),
            vjust = -0.5, size = 3, color = "darkgray") +
  scale_color_manual(values = annotation_colors) +

  labs(x = "Clade Name", y = "Cumulative Sum",
       title = "Cumulative Sum and Biological Processes",
       fill = "Cumulative Sum",
       color = "Biological Processes") +

  theme_main +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```




```
#b <-
ggplot() +
  # Bar chart for cumulative_sum
  geom_bar(data = cumulative_data,
           aes(x = factor(-root), y = cumulative_sum),
           stat = "identity", fill = "darkgray", colour = NA) +
  geom_text(data = cumulative_data,
            aes(x = factor(-root), y = cumulative_sum, label =
             ↪  cumulative_sum),
            vjust = -0.5, size = 3, color = "darkgray") +

  # Line chart for biological processes
  geom_line(data = long_data,
            aes(x = factor(-root), y = Value, color = Process, group =
             ↪  Process),
            size = 1) +

  # Use the defined color palette
  scale_color_manual(values = annotation_colors) +

  labs(x = "Clade Name", y = "Cumulative Sum",
       title = "Cumulative Sum and Biological Processes",
       fill = "Cumulative Sum",
```

```
      color = "Biological Processes") +

  theme_main +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```




```
#ggsave(file = "", plot=a, width=15, height=8)
#ggsave(file = "", plot=b, width=15, height=8)
```

5.6 Calculation of COG Abundance by Function in Species

This analysis is based on the premise that similar COGs share the same biological function. Thus, it is possible to calculate the average abundance of functions associated with COGs in each species, using the number of COGs present in the species identified by the *TaxID*.

5.6.1 Calculation Steps

1. **Species Mapping** Each species was mapped to its respective clade (hierarchical level) based on the *TaxID* information.

- The `ogr@spbranches` table was used to associate species (*ssp_id*) with their *branches* (*lca*).
- The data was organized to include the order of the *TaxIDs*.

2. **COG Annotation**

- The relationship between proteins and genes was enriched with functional information, such as COG identifiers and metabolic pathway names (*pathway_name*).
- Duplicates were removed to ensure that only unique information was considered.

3. **Integration of Species Information**

- A mapping was created between species (*TaxID*) and their respective clades to add the relevant taxonomic information.
- The species were ordered based on their clades, to facilitate hierarchical analysis.

4. **Calculation of Average Abundance by Function**

- For each species and metabolic function (*pathway_name*), the average abundance of COGs was calculated.
- Additional information, such as species names (*ncbi_name*) and clades (*clade_name*), was integrated into the result.

5. **Abundance Adjustment (Capping)**

- To avoid extreme values, the average abundance was adjusted:
  - Values above a limit (mean + 3 standard deviations) were truncated to 100.
  - The adjusted values were calculated separately by metabolic function.

```
lca_spp <- ogr@spbranches %>%
  rename("taxid" = ssp_id, "species" = ssp_name, "lca" = "branch") %>%
  mutate(taxid_order = row_number()) %>%
  dplyr::select(lca, taxid, taxid_order)

clade_taxids <- lca_spp
clade_names <-
 ↪  vroom::vroom("https://raw.githubusercontent.com/dalmolingroup/neurotransmissionevolution/

cog_annotation <- map_ids %>%
  left_join(groot_df, by = c("stringId" = "protein_id")) %>%
  left_join(sensorial_genes, by = c("queryItem" = "gene_symbol")) %>%
  distinct(queryItem, cog_id, pathway_name) %>%
  dplyr::select(cog_id, pathway_name) %>%
  unique() %>%
  na.omit()
```

```r
cog_abundance_by_taxid <- cogdata %>%
  filter(cog_id %in% nodelist[["cog_id"]]) %>%
  count(ssp_id, cog_id, name = "abundance") %>%
  left_join(cog_annotation, by = "cog_id")

# Mapping species to clade info
ordered_species <- string_eukaryotes %>%
  dplyr::select(taxid, ncbi_name) %>%
  left_join(clade_taxids, by = "taxid") %>%
  left_join(clade_names, by = c("lca" = "root")) %>%
  na.omit() %>% unique() %>%
  arrange(desc(lca)) %>%
  dplyr::select(-taxid_order)

avg_abundance_by_function <- cog_abundance_by_taxid %>%
  group_by(ssp_id, pathway_name) %>%
  summarise(avg_abundance = mean(abundance)) %>%
  ungroup() %>%
  # Adding species and clade info
  left_join(ordered_species %>% mutate(taxid = as.double(taxid)), by =
  ↪  c("ssp_id" = "taxid")) %>%
  unique() %>%
  arrange(desc(lca)) %>%
  mutate(ncbi_name = factor(ncbi_name, levels = unique(ncbi_name)),
         clade_name = factor(clade_name, levels = unique(clade_name))) %>%
  na.omit()

capped_abundance_by_function <- avg_abundance_by_function %>%
  # mutate(capped_abundance = ifelse(abundance >= 100, 100, abundance)) %>%
  group_by(pathway_name) %>%
  mutate(
    # max_abundance = max(abundance[lca <= 29])
    max_abundance = avg_abundance[lca <= 29] %>% { mean(.) + 3*sd(.) }
    ,abundance     = ifelse(avg_abundance >= max_abundance,
 ↪  pmin(max_abundance, 100), pmin(avg_abundance, 100)))

# List of signatures
signatures <- unique(node_annotation$pathway_name)

roots_seq <- node_annotation %>%
  arrange(desc(root)) %>%
  dplyr:: select(root, clade_name) %>%
  unique()
```

```
roots_seq$clade_name <- factor(roots_seq$clade_name, levels =
 ↪  roots_seq$clade_name)
```

5.6.2 Visualization

In the following graphs, it is possible to observe the average abundance of COGs in each clade. Each bar represents a species within the respective clade, while the colors indicate the different metabolic functions associated with the orthologous groups.

5.6.2.1 Graph of Average COG Abundance by Clade

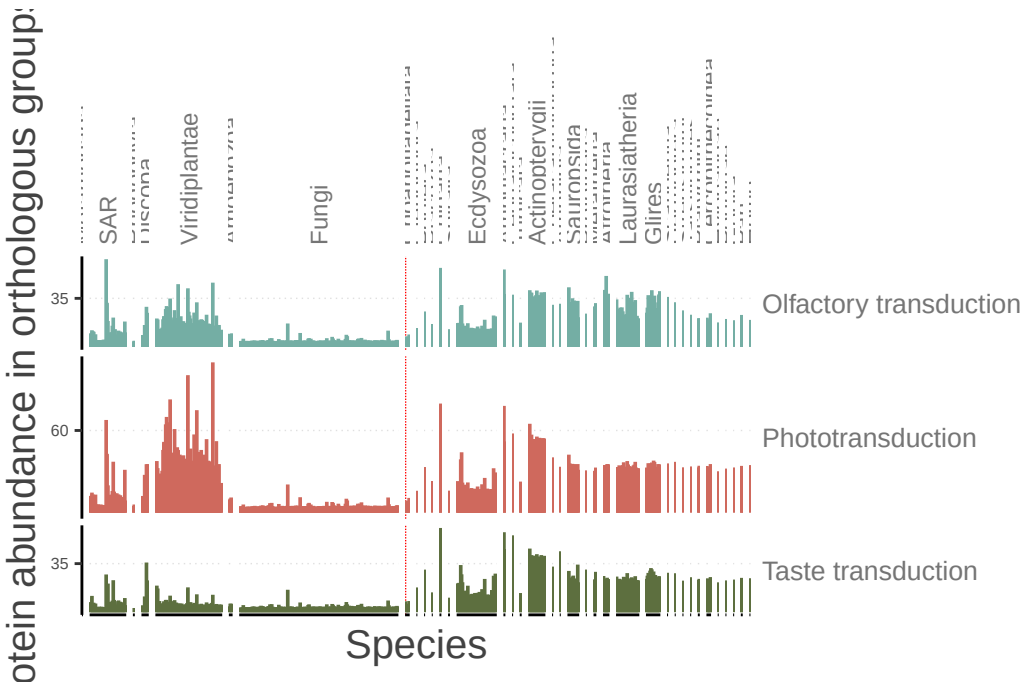
- The **X** axis represents the species.
- The **Y** axis indicates the average abundance of proteins in orthologous groups.
- The bars are colored according to the metabolic function (*pathway_name*).

5.6.2.2 Graph with Adjusted Abundance (Capped)

- To avoid distortions caused by extreme values, the abundance was adjusted to a maximum limit (mean + 3 standard deviations).
- This graph provides a more balanced visualization, especially for metabolic functions with very high values.

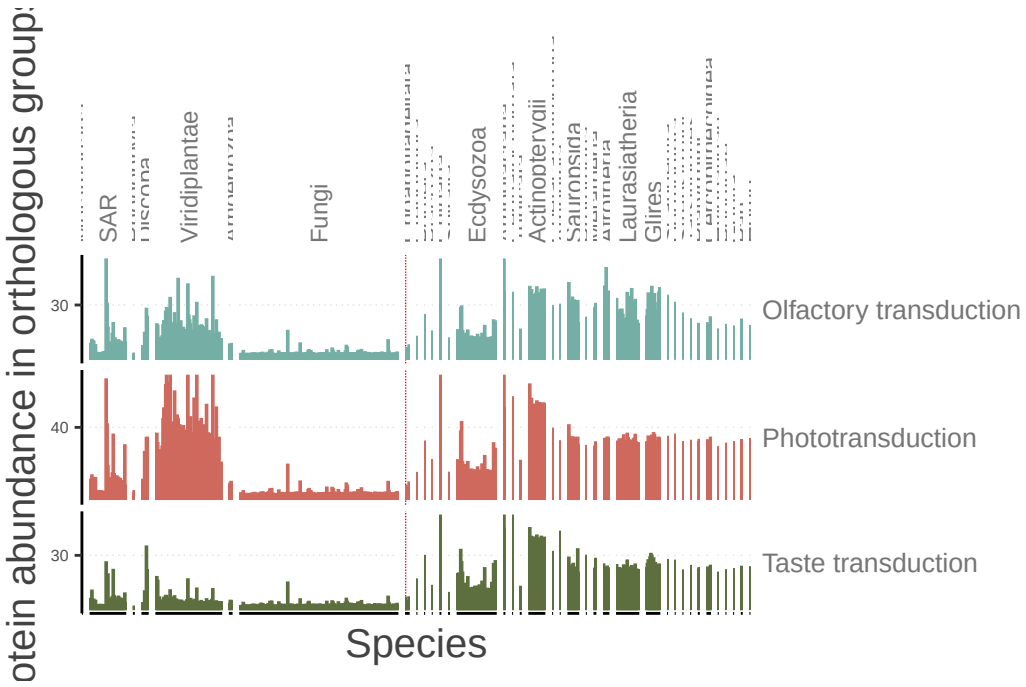
```
# Plotting by species
ggplot(avg_abundance_by_function) +
  # Geoms  ----------------
choanoflagellata_line +
  geom_bar(
    aes(x = ncbi_name, y = avg_abundance, fill = pathway_name, color =
     ↪  after_scale(darken(fill, 0.1)))
    ,stat = "identity"
  ) +
  # Labels  ---------------
 xlab("Species") +
  ylab("Average protein abundance in orthologous groups") +
  #ylab("Average protein abundance in orthologous groups") +
  # Scales ----------------
scale_y_continuous(breaks = tick_function, minor_breaks = NULL) +
  scale_fill_manual(values = annotation_colors %>% darken(0.1)) +
  # Styling ---------------
```

```
facet_grid(
  pathway_name ~ clade_name
  ,scales   = "free"
  ,space    = "free"
  ,labeller = labeller(annotation = annotation_labels)
) +
  theme_classic() +
  theme_main
```




```
# Plotting by species capped
ggplot(capped_abundance_by_function) +
  # Geoms  ----------------
choanoflagellata_line +
  geom_bar(
    aes(x = ncbi_name, y = abundance, fill = pathway_name, color =
     ↪  after_scale(darken(fill, 0.1)))
    ,stat = "identity"
  ) +
  # Labels  ---------------
  xlab("Species") +
  ylab("Average protein abundance in orthologous groups") +
  #ylab("Average protein abundance in orthologous groups") +
```

```
  # Scales ----------------
  scale_y_continuous(breaks = tick_function, minor_breaks = NULL) +
  scale_fill_manual(values = annotation_colors %>% darken(0.1)) +
  # Styling ---------------
facet_grid(
  pathway_name ~ clade_name
  ,scales   = "free"
  ,space    = "free"
  ,labeller = labeller(annotation = annotation_labels)
) +
  theme_classic() +
  theme_main
```



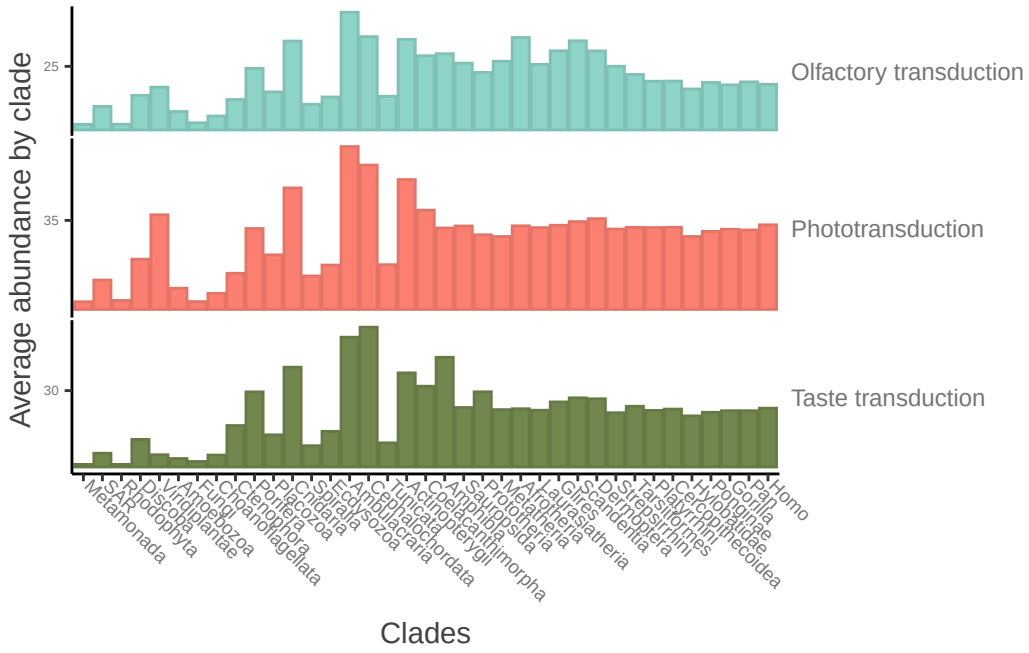
5.6.2.3 Simplified Visualization: Abundance by Clade

For a less detailed analysis, it is possible to visualize the average abundance of COGs only at the clade level, ignoring the differences between individual species. This graph provides a more direct overview, ideal for identifying global trends.

- The **X** axis represents the clades.
- The **Y** axis shows the average abundance of proteins in orthologous groups per clade.

- The bars are grouped by clade and colored according to the metabolic functions (*pathway_name*).

```
# Ploting by clade
ggplot(avg_abundance_by_function) +
  geom_bar(
    aes(x = clade_name, y = avg_abundance, fill = pathway_name, color =
     ↪  after_scale(darken(fill, 0.1)))
    ,stat = "summary"
    ,fun  = "mean"
  ) +
  scale_y_continuous(breaks = tick_function, minor_breaks = NULL) +
  scale_fill_manual(values = annotation_colors, guide = "none") +
  facet_grid(
    pathway_name ~ .
    ,scales   = "free"
    ,space    = "free_y"
    ,labeller = labeller(annotation = sub("\n", "", annotation_labels))
  ) +
  xlab("Clades") +
  ylab("Average abundance by clade") +
  theme_classic() +
  theme_average
```



6 OrthoGuide: Pre-computed Gene Rooting Data

7 OrthoGuide: Pre-computed Gene Rooting Data

OrthoGuide is available at https://dalmolingroup.imd.ufrn.br/orthoguide/ and provides immediate access to high-quality, pre-computed rooting data for 8 species. Instead of running complex analyses, users can query this database and get the information they need in seconds, along with a few exploratory plots.

7.1 How OrthoGuide Works

All data in the database is generated using the GeneBridge R package (v0.99.2). The gene-to-COG (Clusters of Orthologous Groups) information was obtained from STRING-db version 11.0.