GeneBridge

1 GeneBridge

1.1 Importando Pacotes

```
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)
```

1.2 Definindo funções para usar posteriormente

A primeira delas, busca o respectivo id de proteína para cada gene da lista de entrada. A segunda, retorna as interações entre essas proteínas e a última filtra as interações pelo score de confiança combinado maior que 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 = "\t", 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 = "\t", 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)
}
```

1.3 Carregando lista de genes e os dados de ortologia

Carregamos os dados de ortologia através do **AnnotationHub**, esse pacote do R fornece um local central onde arquivos genômicos (VCF, bed, wig) e outros recursos de locais padrões (por exemplo, UCSC, Ensembl) podem ser acessados. Dessa forma, temos acesso aos arquivos de entrada para o algoritmo do GeneBridge.

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

# Query Phylotree and OG data
```

```
ah <- AnnotationHub()
meta <- query(ah, "geneplast")
load(meta[["AH83116"]])

head(sensorial_genes)
head(cogdata)
```

1.4 1.Pré-processamento

1.4.1 1.1.Mapeamento

Para as próximas análises, precisamos cruzar informações entre nossos genes de interesse (Gene IDs da tabela `sensorial_genes`) e Protein IDs (da tabela `cogdata`). A API do STRINGdb é usada para mapear os Gene IDs para os Protein IDs, permitindo a filtragem dos genes de interesse na tabela cogdata. O objetivo final é obter um conjunto filtrado de genes sensoriais com seus respectivos pathways e 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 = ",")
```

1.4.2 1.2.Resolverndo COGs duplicados

Devido a eventos evolutivos, como duplicação gênica, alguns genes podem ser associados a mais de um Cluster of Orthologous Groups (COG). Para garantir a funcionalidade do algoritmo, é necessário resolver esses casos, priorizando COGs de acordo com os seguintes critérios:

1. Prioridade por tipo de COG:

- KOGs têm maior prioridade.
- COGs têm maior prioridade do que NOGs.

2. Casos com COGs iniciando pela mesma letra:

- São resolvidos manualmente, com base na função anotada do COG e na questão científica do estudo.

O código abaixo implementa essa resolução e integra as correções à tabela principal.

```
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 = ",")
```

1.5 3.Processamento

O objetivo desta etapa é realizar o enraizamento dos genes de interesse utilizando o pacote **GeneBridge**. Para isso, utilizamos as funções `newBridge`, `runBridge` e `runPermutation`, que produzem resultados estatísticos associados aos COGs selecionados em uma árvore filogenética.

1.5.1 3.1.Inputs necessários

1. `ogdata`:
   - Dataset contendo três colunas principais:
     - `Protein ID`: Identificadores das proteínas.
     - `COG ID`: Clusters de interesse.
     - `Specie ID`: Identificadores das espécies.
   - No exemplo, está sendo utilizado o objeto `cogdata`.
2. `phyloTree`:
   - Árvore filogenética contendo 476 eucariotos, representando a estrutura evolutiva entre as espécies analisadas.
3. `ogids`:
   - Lista dos **COGs de interesse**. Esse conjunto é derivado da tabela `gene_cogs` e inclui os COGs associados às proteínas após o processamento anterior.
4. `refsp`:
   - Espécie de referência para o enraizamento. No exemplo, utilizamos `9606` (humano).

A função *getBridge* extrai os resultados gerados pelo GeneBridge em formato de tabela. A tabela *res* contém os resultados estastisticos do enraizamento.

```
## 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 = "../data/ogr.RData")
```

1.6 4.Pós-Processamento

Após realizar o enraizamento com o **GeneBridge**, é necessário ajustar os dados para melhorar a visualização e a interpretação dos resultados. Nessa etapa, adicionamos os nomes dos clados às raízes identificadas, utilizando uma tabela externa que relaciona os identificadores das raízes aos nomes dos clados.

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

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 = ",")
```

1.6.1 4.1.Rede de Interação Proteína-Proteína

A construção de uma rede de interação proteína-proteína (PPI) é uma etapa essencial para identificar as relações funcionais entre proteínas. Neste processo, utilizamos a API do **STRINGdb**, um banco de dados que cataloga interações entre proteínas com base em diversas fontes, incluindo ensaios experimentais, co-expressão, e evidências extraídas de publicações científicas.

A API do STRINGdb oferece métodos para: - Obter interações proteicas para uma lista de proteínas. - Selecionar fontes específicas de evidências. - Calcular e combinar escores baseados nas evidências selecionadas.

Mais informações sobre a API podem ser encontradas na documentação STRING API.

```
# 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 o 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)
```

Para a construção do grafo, além das interações entre as proteínas, é necessário que cada nó seja anotado com informações adicionais que serão usadas na análise, como: - Nome da proteína. - Clado onde está enraizado. - Via metabólica em que participa.

```
## 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)
```

Além da estrutura do grafo, podemos calcular métricas como o número de conexões (grau) de cada nó.

```
# 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 = ",")
```

1.7 Importar bibliotecas

```
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)
```

1.8 Definir funções

```
# 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 collors 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)
}
```

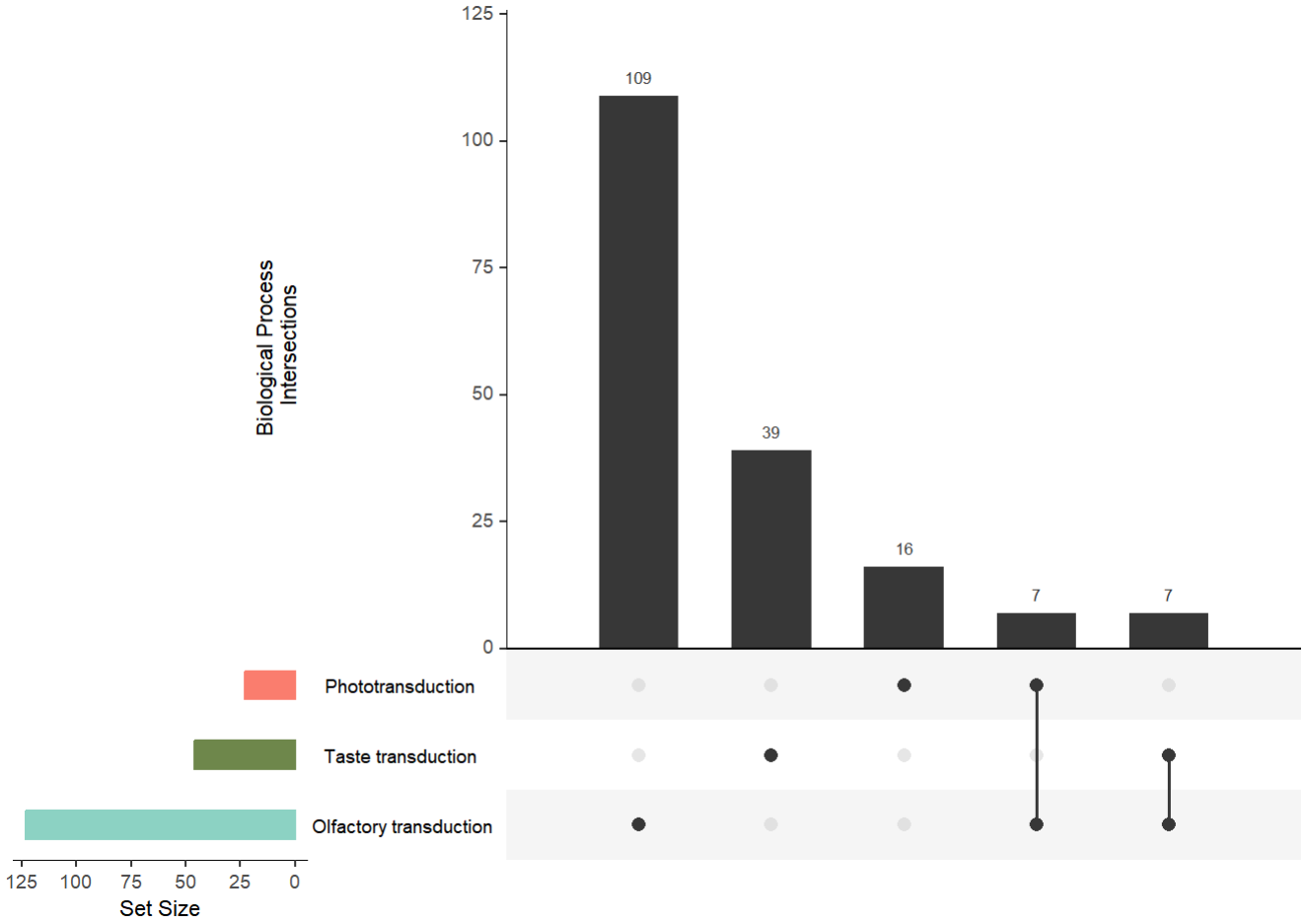
1.9 Carregando tabelas necessárias

```
#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 = ",")
```

1.10 1.Visualização com UpSet Plot

O *UpSet Plot* é uma ferramenta útil para visualizar a distribuição e concatenação de genes entre diferentes vias metabólicas. Ele permite identificar como os genes estão compartilhados ou exclusivos entre as categorias analisadas.

```
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")
```



1.11 2.Visualização da Rede de Interação Proteína-Proteína

A visualização da rede de interação é essencial para compreender as conexões funcionais entre proteínas. Aqui, utilizamos o pacote **easylayout**, desenvolvido por Danilo Imparato, para gerar um layout eficiente. Este pacote organiza os nós da rede em coordenadas x e y, permitindo uma visualização estruturada e clara. Posteriormente, o grafo será plotado com o **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")
```

1.11.1 2.1.Visualização da Ancestralidade de Cada Nó

A análise da ancestralidade de cada nó na rede fornece uma visão evolutiva sobre as proteínas analisadas. Aqui, utilizamos o **ggraph** para plotar o grafo com as posições previamente salvas pelo *easylayout*.

Os nós são coloridos de acordo com a distância em relação ao último ancestral comum (LCA) dos clados analisados e o humano (*Human-LCA*). A tonalidade mais escura indica clados mais antigos em relação ao humano, enquanto tons claros de azul representam clados mais novos, mais próximos do *Human-LCA*.

```
load("../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")
```

1.11.2 2.2.Visualização da Rede de Interação Proteína-Proteína em Humano

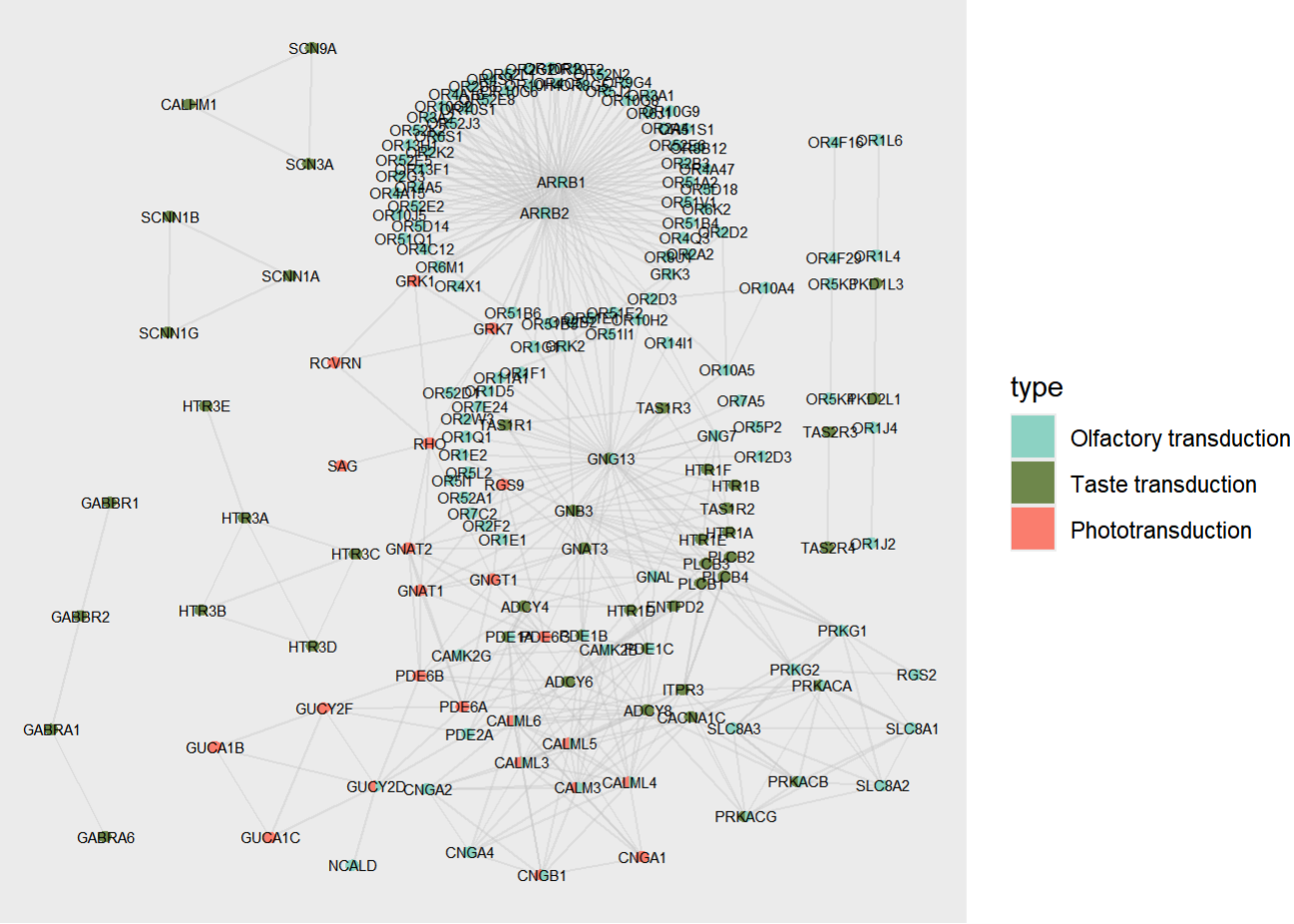
Para compreender melhor a relação entre proteínas humanas, plotamos a rede de interação onde os nós representam os genes humanos associados aos seus processos biológicos.

1.11.2.1 Descrição dos elementos do gráfico:

1. **Nós (Círculos):** As cores dos nós são divididas de acordo com os processos biológicos atribuídos a cada gene. O uso de diagramas de pizza permite a visualização de genes que participam de múltiplos processos.
2. **Arestas (Linhas):** Representam as interações proteicas baseadas em dados do STRINGdb.

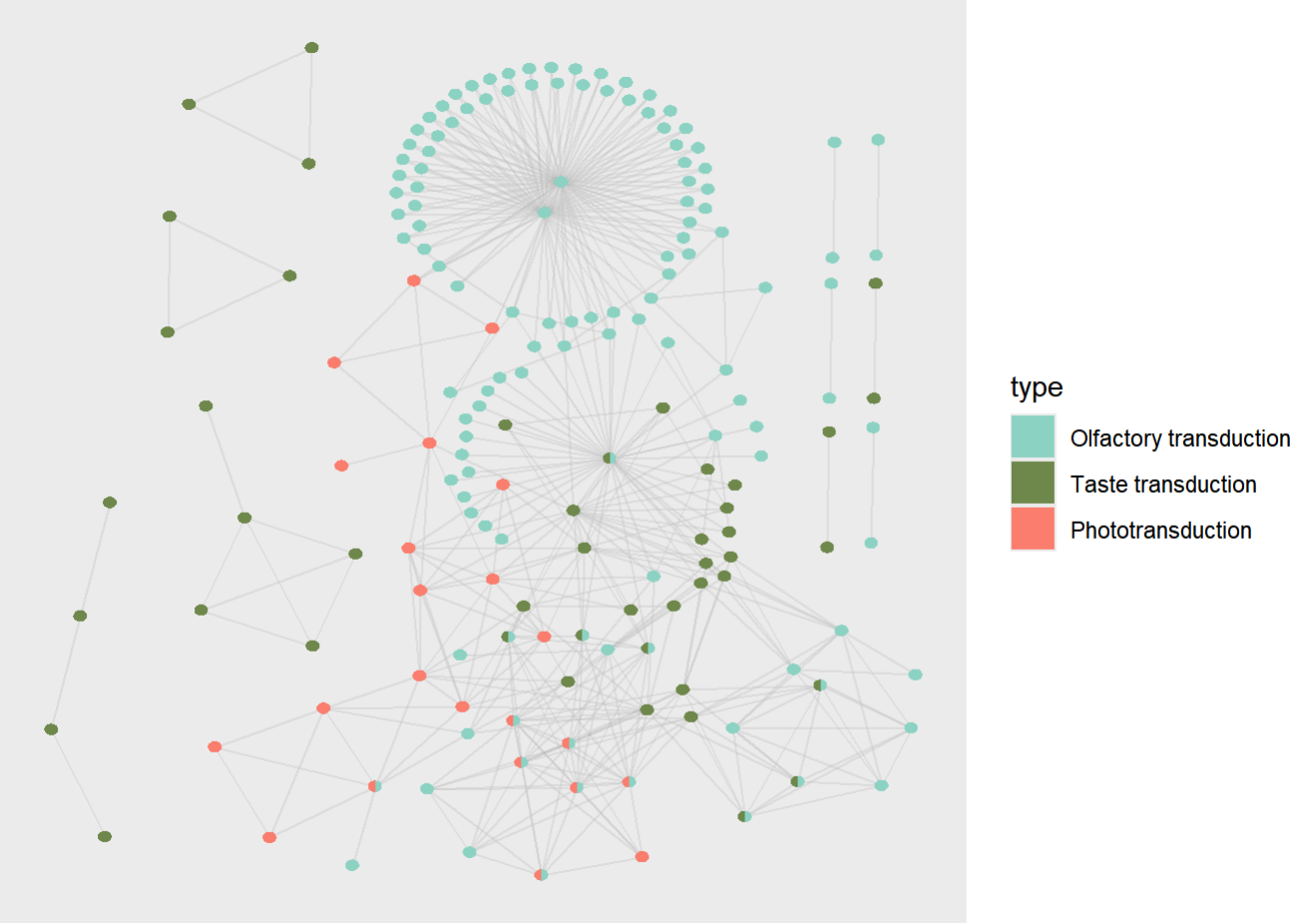
3. **Rótulos dos genes:** Cada nó está anotado com o símbolo do gene correspondente, posicionado estrategicamente para facilitar a leitura.

```
## 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
  ggplot2::scale_fill_manual(values = color_mappings, drop = FALSE)
```




```
#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)
```



1.11.3 2.3.Visualização da Rede de Interação Proteína-Proteína em Cada Clado

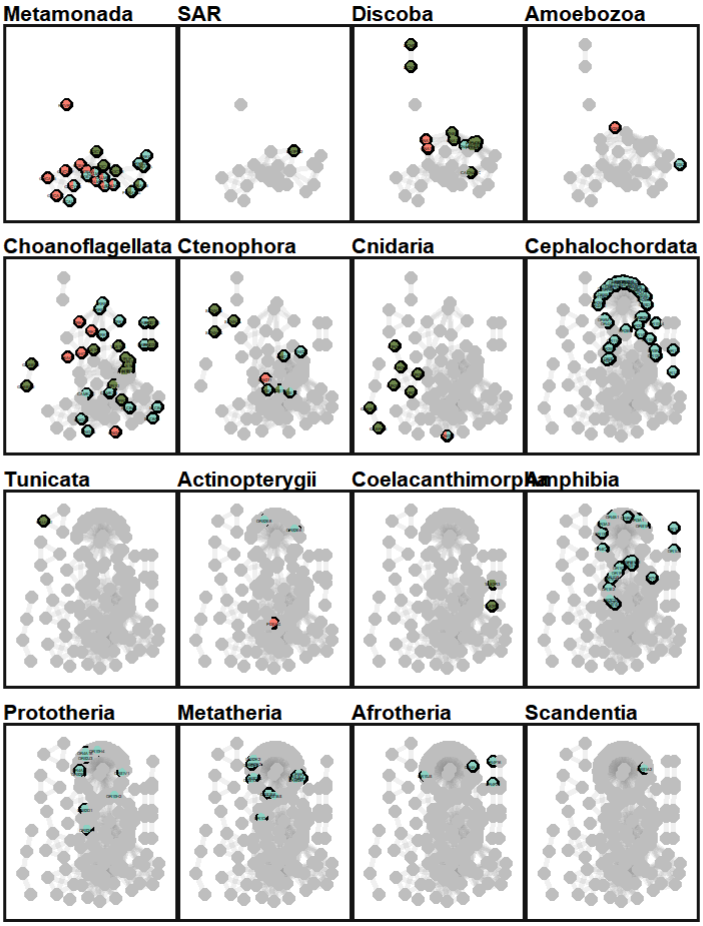
Nesta seção, visualizamos os genes que estão estatisticamente enraizados em cada clado. A disposição dos genes permite observar o incremento dos genes ortólogos em função da complexidade e antiguidade do sistema biológico.

1.11.3.1 Características da visualização:

1. **Evolução dos grafos:** Os grafos são organizados da esquerda para a direita e de cima para baixo, permitindo analisar a progressão evolutiva.
2. **Coloração dos nós:** A cor dos nós indica o nível de ancestralidade, como previamente destacado, onde tons mais escuros representam clados mais antigos e tons mais claros indicam proximidade evolutiva com humanos.
3. **Organismos de interesse:** Além de visualizar todos os clados, é possível gerar gráficos focados apenas em determinados grupos, como *Metamonada*, *Choanoflagellata*, *Cephalochordata* e *Amphibia*.

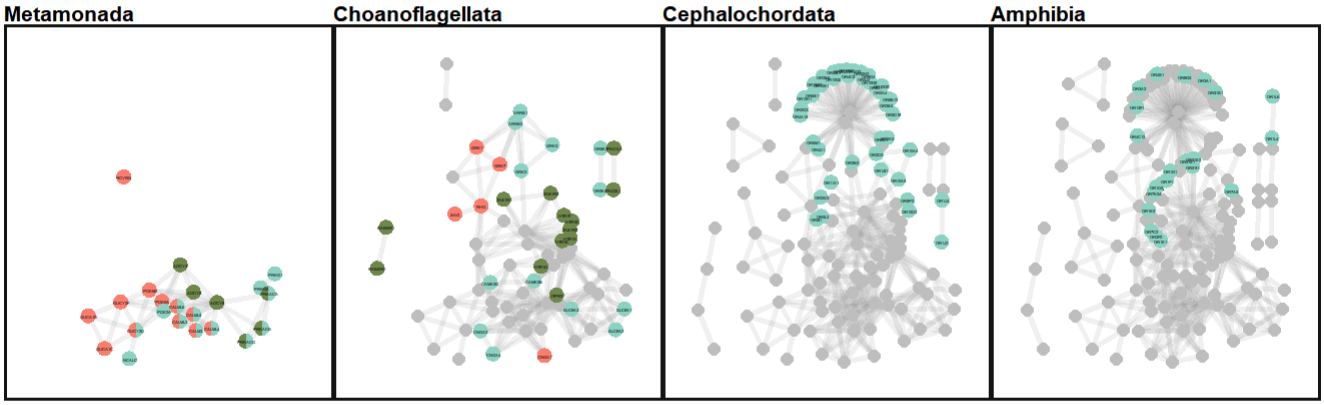
Com estas visualizações, é possível identificar padrões de evolução dos genes em diferentes clados e realizar comparações detalhadas com organismos de interesse específico.

```
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
)
```



1.12 Importando Pacotes

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

1.13 Definindo funções

Essas funções possuem a finalidade de organizar a anotação dos vértices de nossa nodelist e calcular o cumulativo de genes por clado e por processo biológico

```
calculate_cumulative_genes <- function(nodelist) {

    # Obter todas as categorias possíveis de clade_name
  all_clades <- node_annotation %>%
    arrange(desc(root)) %>%
    dplyr:: select(clade_name) %>%
```

```
    unique()

  # Definir as colunas de interesse
  process_columns <- c("queryItem", "root", "clade_name",
                       "Olfactory transduction",
                       "Taste transduction",
                       "Phototransduction")

  # Calcular o cumulativo agrupando por 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) {

  # Obter todas as categorias possíveis de clade_name
  all_clades <- node_annotation %>%
    arrange(desc(root)) %>%
    dplyr:: select(clade_name) %>%
    unique()

   # Definir as colunas de interesse
  process_columns <- c("queryItem", "root", "clade_name",
                       "Olfactory transduction",
                       "Taste transduction",
                       "Phototransduction")

  # Calcular a soma cumulativa para cada processo biológico
  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)
}
```

1.14 Definindo parâmetros estéticos para o ggplot

Atribuindo cores padrões para as vias metabólicas da análise e definindo os demais padrões estéticos para plotagens

```
# 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 = "#7
  ,strip.text.y       = element_text(size = 10, angle = 0, hjust = 0, vjust = 0.5, color = "#7
  ,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 = "#75
  ,strip.text.x       = element_text(size = 7, angle = 90, hjust = 0, vjust = 0.5, color = "#7
  ,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 = "#75
  ,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 }
}
```

1.15 Carregar tabelas necessárias

Para prosseguir com a análise, é necessário carregar uma tabela contendo informações do *taxid* de cada espécie. Essas informações serão posteriormente utilizadas para mapear os *taxid* das espécies com os *taxid* de cada COG.

1.15.1 Descrição da Tabela

- **Nome do Arquivo**: `string_eukaryotes.rda`
- **Conteúdo**: Informações taxonômicas de espécies eucarióticas.

A tabela carrega os seguintes campos principais:
- **TaxID** (*taxonomic identifier*): Um identificador único associado a cada espécie. - Nome da espécie

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

# Todo: salvar tabelas dos qmd anteriores
nodelist <- vroom::vroom(file = here("data/nodelist.csv"), delim = ",")
gene_cogs <-  vroom::vroom(file = here("data/gene_cogs.csv"), delim = ",")
sensorial_genes <- read.csv("../data/sensorial_genes.csv")
map_ids <- vroom::vroom("../data/map_ids.csv")
```

```
groot_df <- vroom::vroom("../data/groot_df.csv")

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

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

1.16 Visualização: Padrão de Enraizamento

Essa visualização permite analisar o padrão cumulativo de surgimento de genes ao longo do tempo evolutivo. Além disso, também mostra o crescimento dos processos biológicos separados por cada via metabólica.

1.16.1 Características dos Gráficos

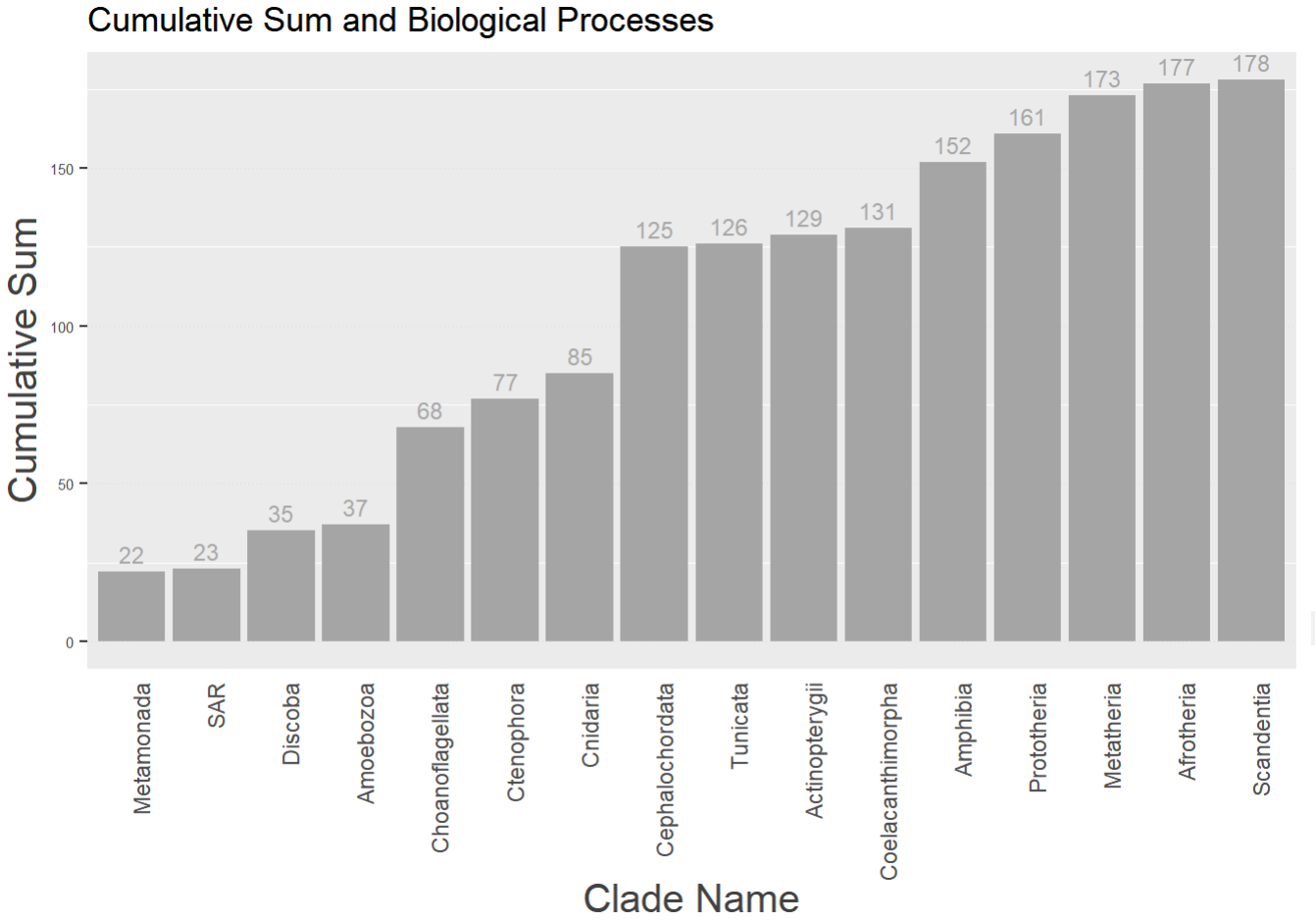
1. **Gráfico de Barras (Padrão Cumulativo)**:
   - Representa o somatório cumulativo de genes associados a cada clado.
   - Cada barra exibe o total acumulado de genes em determinado clado, permitindo identificar o padrão de diversificação.
2. **Gráfico Combinado (Barras e Linhas)**:
   - As barras indicam o somatório cumulativo de genes por clado.
   - As linhas representam o crescimento de diferentes processos biológicos (*Process*), separados por vias metabólicas, permitindo a comparação entre o surgimento cumulativo de genes e o desenvolvimento de funções biológicas.

1.16.2 Interpretação

- **Padrão Cumulativo**: O gráfico de barras mostra como os genes foram surgindo ao longo do tempo, cumulativamente. Essa visão ajuda a identificar momentos de maior diversificação genética.
- **Comparação por Processos Biológicos**: O gráfico combinado permite observar quais processos biológicos se destacam em diferentes momentos da evolução e como eles estão relacionados ao surgimento de novos genes.

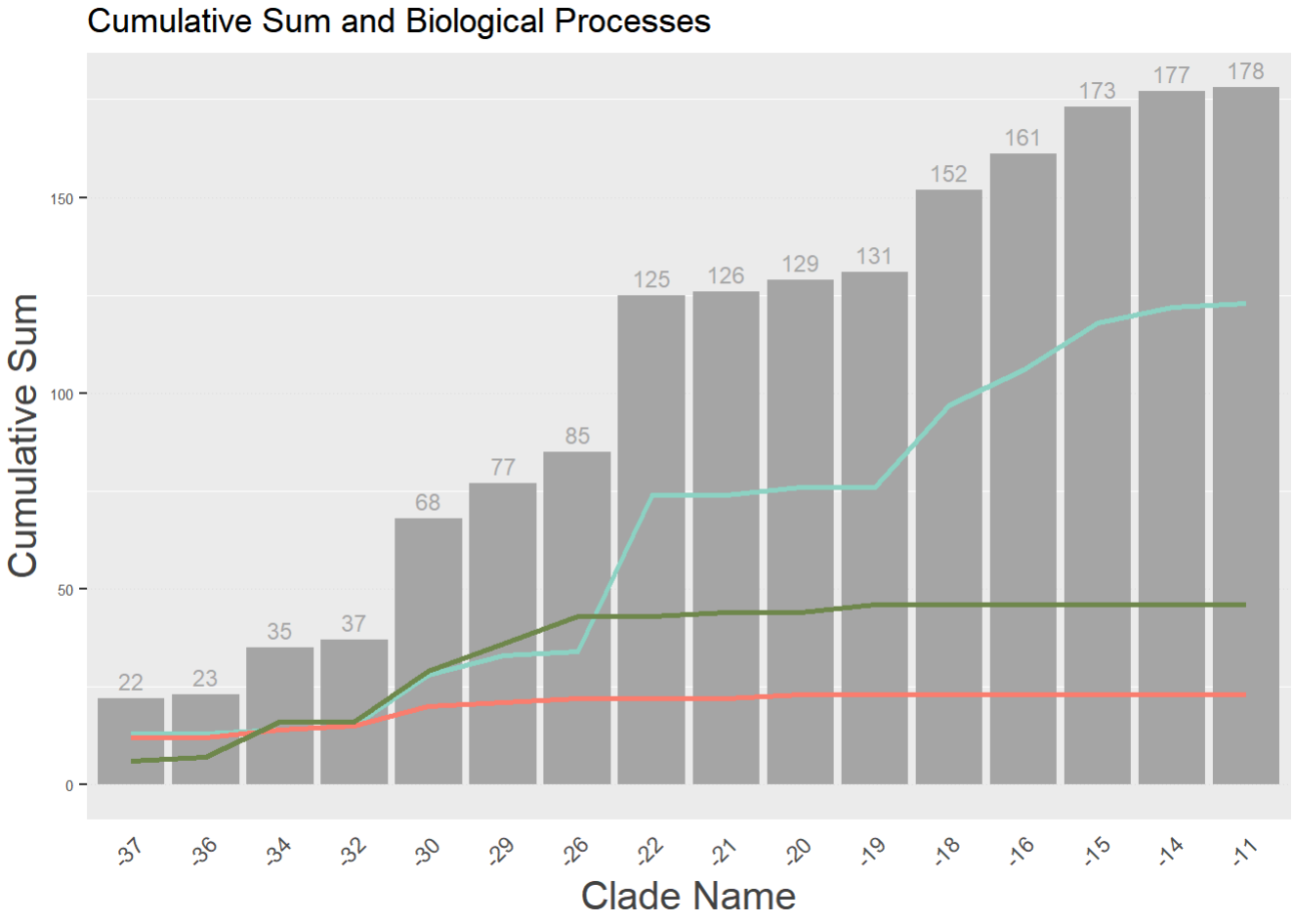
```
# Mapping roots and proteins info
node_annotation <- nodelist %>%
  inner_join(gene_cogs, by = c("node" = "protein_id", "cog_id")) %>%
  inner_join(sensorial_genes, by = c("queryItem" = "gene_symbol")) %>%
  distinct(queryItem, cog_id, pathway_name, root, clade_name)

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

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() +
  # Gráfico de barras para 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 = cumul
            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() +
  # Gráfico de barras para 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") +

  # Gráfico de linhas para os processos biológicos
  geom_line(data = long_data,
            aes(x = factor(-root), y = Value, color = Process, group = Process),
            size = 1) +

  # Usar a paleta de cores definida
  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)
```

1.17 Cálculo da Abundância de COGs por Função nas Espécies

Esta análise parte da premissa de que COGs similares compartilham a mesma função biológica. Assim, é possível calcular a abundância média das funções associadas aos COGs em cada espécie, utilizando a quantidade de COGs presentes na espécie identificada pelo *TaxID*.

1.17.1 Etapas do Cálculo

1. **Mapeamento das Espécies**
   Cada espécie foi mapeada para seu respectivo clado (nível hierárquico) com base nas informações do *TaxID*.
   - Foi utilizada a tabela `ogr@spbranches` para associar espécies (*ssp_id*) com seus *branches* (*lca*).
   - Os dados foram organizados para incluir a ordem dos *TaxIDs*.
2. **Anotação dos COGs**
   - A relação entre proteínas e genes foi enriquecida com informações funcionais, como identificadores de COGs e nomes de vias metabólicas (*pathway_name*).
   - Duplicatas foram removidas para garantir que apenas informações únicas fossem consideradas.
3. **Integração de Informações das Espécies**
   - Foi criado um mapeamento entre espécies (*TaxID*) e seus respectivos clados para adicionar as informações taxonômicas relevantes.
   - As espécies foram ordenadas com base em seus clados, para facilitar a análise hierárquica.
4. **Cálculo da Abundância Média por Função**
   - Para cada espécie e função metabólica (*pathway_name*), foi calculada a abundância média dos COGs.
   - Informações adicionais, como nomes das espécies (*ncbi_name*) e clados (*clade_name*), foram integradas ao resultado.
5. **Ajuste da Abundância (Capping)**
   - Para evitar valores extremos, a abundância média foi ajustada:
     - Valores acima de um limite (média + 3 desvios padrão) foram truncados para 100.
     - Os valores ajustados foram calculados separadamente por função metabólica.

```
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/neurotransmission

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()

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_

# 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)
```

1.17.2 Visualização

Nos gráficos a seguir, é possível observar a abundância média dos COGs em cada clado. Cada barra representa uma espécie dentro do respectivo clado, enquanto as cores indicam as diferentes funções metabólicas associadas aos grupos ortólogos.

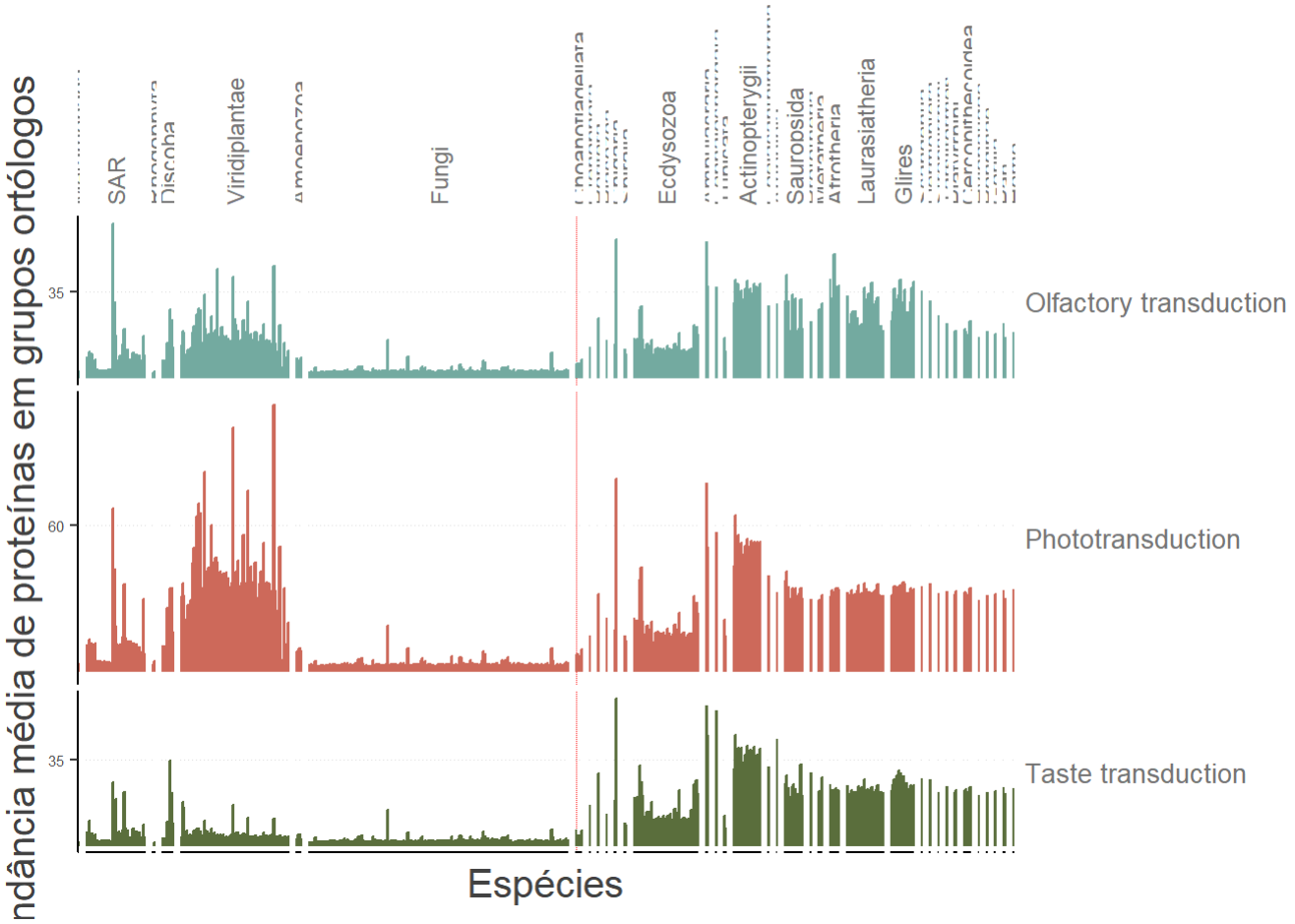
1.17.2.1 Gráfico de Abundância Média de COGs por Clado

- O eixo **X** representa as espécies.
- O eixo **Y** indica a abundância média de proteínas em grupos ortólogos.
- As barras são coloridas de acordo com a função metabólica (*pathway_name*).

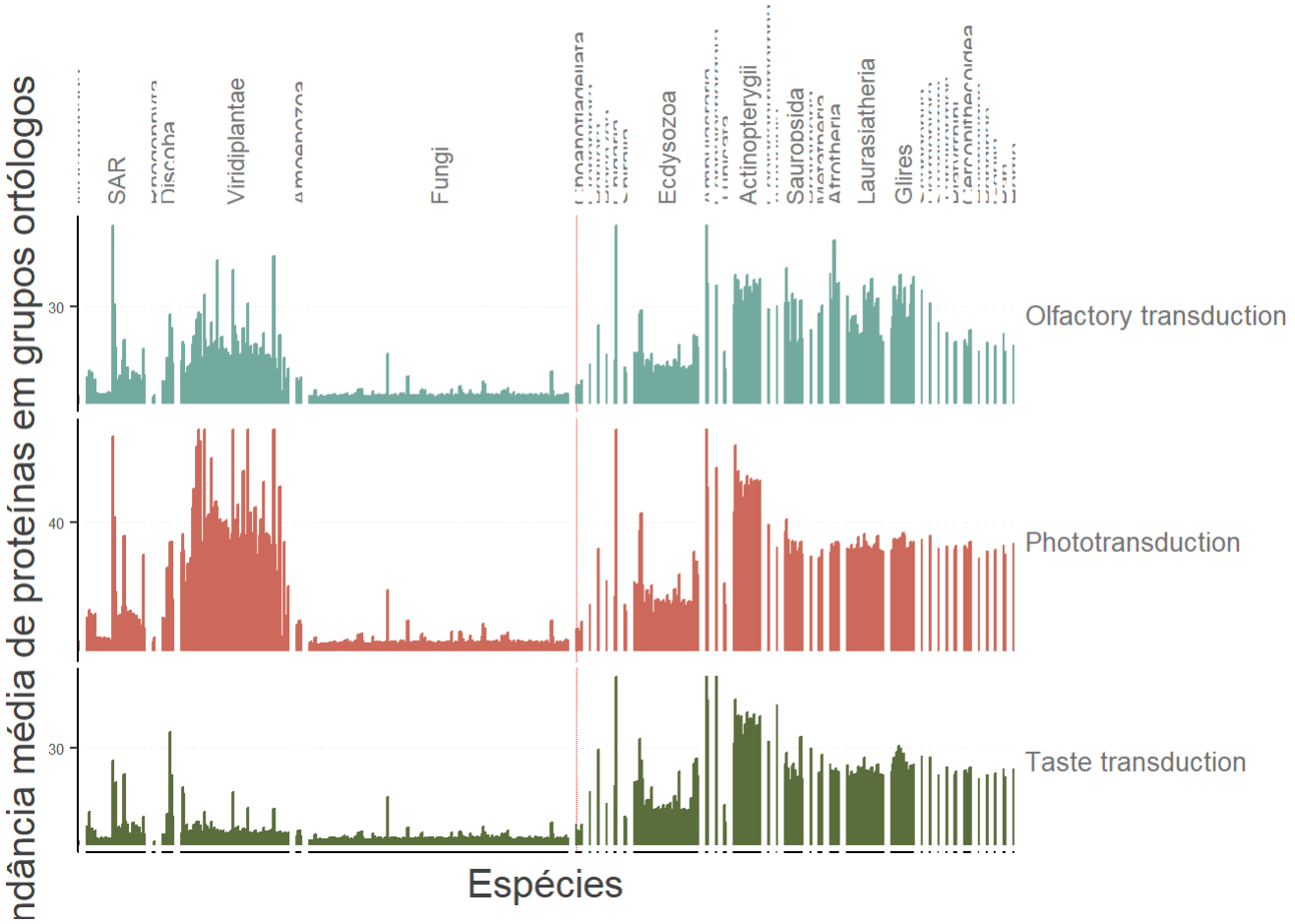
1.17.2.2 Gráfico com Abundância Ajustada (Capped)

- Para evitar distorções causadas por valores extremos, a abundância foi ajustada a um limite máximo (média + 3 desvios padrão).
- Este gráfico fornece uma visualização mais equilibrada, especialmente para funções metabólicas com valores muito elevados.

```
# 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
    ,stat = "identity"
  ) +
  # Labels  ---------------
 xlab("Espécies") +
  ylab("Abundância média de proteínas em grupos ortólogos") +
  #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.
    ,stat = "identity"
  ) +
  # Labels  ---------------
  xlab("Espécies") +
  ylab("Abundância média de proteínas em grupos ortólogos") +
  #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
```



1.17.2.3 Visualização Simplificada: Abundância por Clado

Para uma análise menos detalhada, é possível visualizar a abundância média de COGs apenas a nível de clado, ignorando as diferenças entre espécies individuais. Este gráfico fornece uma visão geral mais direta, ideal para identificar tendências globais.

- O eixo **X** representa os clados.
- O eixo **Y** mostra a abundância média de proteínas em grupos ortólogos por clado.
- As barras são agrupadas por clado e coloridas de acordo com as funções metabólicas (*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(fil
    ,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("Clados") +
  ylab("Abundância média por clado") +
  theme_classic() +
  theme_average
```