Supplementary Information

Ionotropic receptors as the driving force behind human synapse establishment

Lucas H. Viscardi Danilo O. Imparato Maria Cátira Bortolini
Rodrigo J. S. Dalmolin

Contents

Project structure **1**

Preprocessing **2**

Eukaryota species tree . . . 2

NCBI Taxonomy tree . . . 2

Hybrid tree . . . 4

Gene selection and annotation . . . 10

Neurotransmitter systems annotation . . . 10

Base ID lookup table . . . 11

Neuroexclusivity . . . 12

Expression neuroexclusivity . . . 12

Pathway neuroexclusivity . . . 14

Orthology data . . . 14

Network . . . 15

Retrieving network data . . . 15

Recomputing scores . . . 16

Analysis **17**

Root inference . . . 17

Geneplast . . . 17

Clade names . . . 18

Phyletic patterns . . . 19

Neuroexclusivity . . . 22

Expression neuroexclusivity . . . 22

Pathway neuroexclusivity . . . 22

Network . . . 23

Graph data . . . 23

Manuscript figure 1 . . . 25

Manuscript figure 2 27
Manuscript figure 3 27
Manuscript figure 4 30
Supplementary network figures 32
Manuscript set diagrams 35
Abundance 36

Project structure

This project is organized as an R package research compendium. Research compendia aim to provide a *"standard and easily recognizable way for organizing the digital materials of a research project to enable other researchers to inspect, reproduce, and extend the research"* (Marwick, Boettiger & Mullen, 2018). Project structure overview:

```
└──📦 neurotransmissionevolution/
    ├──🗀 analysis/
    │   ├──🗀 plots/
    │   ├──📓 geneplast.Rmd
    │   ├──🖹 geneplast_roots.tsv
    │   └── ...
    ├──🗀 data/
    │   ├──💾 cogs.rda
    │   ├──🖹 gene_annotation.tsv
    │   └── ...
    ├──🗀 data-raw/
    │   ├──📓 tree_ncbi.Rmd
    │   ├──🖹 tree_ncbi.nwk
    │   └── ...
    ├──🗀 inst/
    ├──🗀 man/
    ├──🗀 manuscripts/
    │   ├──📓 supplement.Rmd
    │   ├──📰 supplement.pdf
    │   └── ...
    ├──🗀 R/
    │   ├──🖹 utils.R
    │   └── ...
    ├──🖹 DESCRIPTION
    ├──🖹 NAMESPACE
    ├──Ⓡ neurotransmissionevolution.Rproj
    └──🖹 README.md
```

The main manuscript analysis is inside `analysis/` and uses preprocessed data from `data/`. Preprocessing steps are included in `data-raw/`. Everything is documented in RMarkdown. Some miscellaneous utility functions reside in `R/`. Additionally, both preprocessed data and utility functions are documented by roxygen2 and can be examined by usual help commands. To explore and tweak this project, users can download the repository's latest release from github (daniloimparato/neurotransmissionevolution) and run `devtools::install()`.

Preprocessing

This section mainly refers to data preprocessing steps done before the actual analysis with the intent of making it either simpler or faster.

Eukaryota species tree

We opted to use the TimeTree database to obtain an standardized Eukaryota species tree, although some target species were absent from it. Absent species were filled in based on NCBI Taxonomy data.

NCBI Taxonomy tree

The following topics describe how we retrieved a target species tree from NCBI Taxonomy data. **Resources**

Table 1: Lists all organisms in STRING v11.

string_species					
#	Col. name	Col. type	Used?	Example	Description
1	taxid	character	yes	9606	NCBI Taxonomy identifier
2	string_type	character	no	core	if the genome of this species is core or periphery
3	string_name	character	yes	Homo sapiens	STRING species name
4	ncbi_official_name	character	no	Homo sapiens	NCBI Taxonomy species name

Location: data-raw/download/species.v11.0.txt
Source: stringdb-static.org/download/species.v11.0.txt

Table 2: Links outdated taxon IDs to corresponding new ones.

ncbi_merged_ids					
#	Col. name	Col. type	Used?	Example	Description
1	taxid	character	yes	140100	id of node that has been merged
2	new_taxid	character	yes	666	id of node that is the result of merging

Location: data-raw/download/taxdump/merged.dmp
Source: ftp.ncbi.nlm.nih.gov/pub/taxonomy/taxdump.tar.gz

Table 3: Represents taxonomy nodes.

ncbi_edgelist					
#	Col. name	Col. type	Used?	Example	Description
1	taxid	character	yes	2	node id in NCBI taxonomy database
2	parent_taxid	character	yes	131567	parent node id in NCBI taxonomy database
3	rank	character	no	superkingdom	rank of this node
4	...	...	no	...	(too many unrelated fields)

Location: data-raw/download/taxdump/nodes.dmp
Source: ftp.ncbi.nlm.nih.gov/pub/taxonomy/taxdump.tar.gz

Table 4: Links taxon IDs to actual species names.

ncbi_taxon_names					
#	Col. name	Col. type	Used?	Example	Description
1	taxid	character	yes	2	the id of node associated with this name
2	name	character	yes	Monera	name itself
3	unique_name	character	no	Monera <bacteria>	the unique variant of this name if name not unique
4	name_class	character	yes	scientific name	type of name

Location: data-raw/download/taxdump/names.dmp
Source: ftp.ncbi.nlm.nih.gov/pub/taxonomy/taxdump.tar.gz

Duplicated genera

Some species from different kingdoms may have the same genus name. Duplicated genera must be noted down because we will use them to fill in missing species at a later time.

```r
# keeping genera nodes
genera_taxids <- ncbi_edgelist %>%
  filter(rank == "genus") %>%
  select(taxid = n1, rank)

duplicated_genera <- ncbi_taxon_names %>%
  # keeping scientific names
  filter(type == "scientific name") %>%
  select(taxid = name, ncbi_name) %>%
  inner_join(genera_taxids) %>%
  # extracting and saving duplicated values
  pull(ncbi_name) %>%
  extract(duplicated(.)) %>%
  write("duplicated_genera.txt")
```

Updating STRING taxon IDs

Some organisms taxon IDs are outdated in STRING. We must update them to work with the most recent NCBI Taxonomy data.

```r
string_species %<>%
  left_join(ncbi_merged_ids) %>%
  mutate(new_taxid = coalesce(new_taxid, taxid))
```

Creating tree graph

The first step is to create a directed graph representing the NCBI Taxonomy tree.

```r
# leaving only "scientific name" rows
ncbi_taxon_names %<>%
  filter(type == "scientific name") %>%
  select(name, ncbi_name)

# finding Eukaryota taxid
eukaryota_taxon_id <- subset(ncbi_taxon_names, ncbi_name == "Eukaryota", "name", drop = TRUE)

# creating graph
g <- graph_from_data_frame(ncbi_edgelist[,2:1], directed = TRUE, vertices = ncbi_taxon_names)

# easing memory
rm(ncbi_edgelist, ncbi_merged_ids)
```

Traversing the graph

The second step is to traverse the graph from the Eukaryota root node to STRING species nodes. This automatically drops all non-eukaryotes and results in a species tree representing only STRING eukaryotes (476).

```r
eukaryote_root <- V(g)[eukaryota_taxon_id]
eukaryote_leaves <- V(g)[string_species[["new_taxid"]]]

# not_found <- subset(string_species, !new_taxid %in% ncbi_taxon_names$name)

eukaryote_paths <- shortest_paths(g, from = eukaryote_root, to = eukaryote_leaves, mode = "out")$vpath

eukaryote_vertices <- eukaryote_paths %>% unlist %>% unique

eukaryote_tree <- induced_subgraph(g, eukaryote_vertices, impl = "create_from_scratch")
```

Saving

Saving `ncbi_tree` and `string_eukaryotes` for package use. These data files are documented by the package. We also create a plain text file `476_ncbi_eukaryotes.txt` containing the updated names of all 476 STRING eukaryotes. This file will be queried against the TimeTree website.

```r
ncbi_tree <- treeio::as.phylo(eukaryote_tree)

string_eukaryotes <- string_species %>%
  filter(new_taxid %in% ncbi_tree$tip.label) %>%
  inner_join(ncbi_taxon_names, by = c("new_taxid" = "name"))

write(string_eukaryotes[["ncbi_name"]],"476_ncbi_eukaryotes.txt")

write.tree(ncbi_tree, "tree_ncbi.nwk")
usethis::use_data(string_eukaryotes, overwrite = TRUE)
```

Hybrid tree

Once we have both the NCBI Eukaryota tree and the list of duplicated genera, we can start assembling the complete hybrid tree.

Resources

Besides downloading all TimeTree species data (`Eukaryota_species.nwk`) we also need to manually query the website for the 476 STRING eukaryotes (`476_ncbi_eukaryotes.txt`). The file is called `476_ncbi_eukaryotes.txt` because it contains updated NCBI Taxonomy names rather than STRING outdated names. This ensures better results.

```r
download_if_missing(
  paste0("http://timetree.org/ajax/direct_download",
         "?direct-download-format=newick",
         "&direct-download-id=23070",
         "&direct-download-rank=species"),
  "Eukaryota_species.nwk"
)
```

`timetree_newick` is the tree obtained by manually uploading `476_ncbi_eukaryotes.txt` to the TimeTree website. `tree_85k` is the complete Eukaryota tree we have just downloaded.

```r
# loading species names and taxon ids
load("../data/string_eukaryotes.rda")

# loading newick tree manually obtained from timetree
timetree_newick <- read.tree("download/timetree_335_eukaryotes.nwk")

# the following genera names are unreliable and should not be searched for
duplicated_genera <- scan("duplicated_genera.txt", what = "character")

# loading all TimeTree species data we have just download (85000 species)
tree_85k <- read.tree("download/Eukaryota_species.nwk")
```

Unfound species with matching genera

Some of the 476 STRING eukaryotes are not present in the TimeTree database. However, sometimes TimeTree does contain tree data for closely related species (e.g. *Monosiga brevicollis* is not present, but *Monosiga ovata* is). Therefore, we can use these closely related species as proxies for the actual species. This is done by searching for genera names in the complete database (`Eukaryota_species.nwk`). In the given *Monosiga brevicollis* example, we search for *Monosiga* in the complete database. We see that there is information for at least one other species of the *Monosiga* genus (in this case, *Monosiga ovata*), so we add *Monosiga brevicollis* as a sister branch to the found species.

When you search for a term in TimeTree, it uses a synonym list obtained from NCBI to try to resolve it. Sometimes TimeTree will resolve a searched term to a scientific name different from the one you searched for. The problem with this is that TimeTree does not make it obvious that it is returning a different term. The first step is to find out which species resolved to different names in the `timetree_335_eukaryotes.nwk` file:

```r
# plot(timetree_newick %>% ladderize, type = "cladogram", use.edge.length = F)

# replacing timetree species underscores with spaces
timetree_newick[["tip.label"]] %<>% str_replace_all("_", " ")

# which timetree species' names exactly match with ncbi's
taxid_indexes <- timetree_newick[["tip.label"]] %>% match(string_eukaryotes[["ncbi_name"]])

# find out which timetree species names didn't exactly match ncbi's
unmatched_names <- timetree_newick[["tip.label"]] %>% magrittr::extract(taxid_indexes %>% is.na)
print(unmatched_names)
```

```
## [1] "Cercospora fijiensis"     "Arthroderma benhamiae"
## [3] "Macropus eugenii"         "Ostreococcus lucimarinus"
## [5] "Oryza nivara"
```

```r
# manually creating lookup table to be joined
ncbi_to_timetree <- tribble(
  ~timetree_name,              ~ncbi_name,
  "Cercospora fijiensis",      "Pseudocercospora fijiensis",
  "Arthroderma benhamiae",     "Trichophyton benhamiae",
  "Macropus eugenii",          "Notamacropus eugenii",
  "Ostreococcus lucimarinus",  "Ostreococcus sp. 'lucimarinus'",
  "Oryza nivara",              "Oryza sativa f. spontanea"
)

# joining info
species_dictionary <- string_eukaryotes %>% left_join(ncbi_to_timetree)

# coalescing NAs to ncbi_name
species_dictionary %<>%
  mutate(timetree_name = coalesce(timetree_name, ncbi_name)) %>%
  mutate(timetree_name = ifelse(timetree_name %in% timetree_newick[["tip.label"]], timetree_name, NA))
```

Now we can start looking for unfound species genera in the complete tree data.

```r
# annotating genera
species_dictionary %<>%
  mutate(genus_search = coalesce(timetree_name, ncbi_name) %>%
  strsplit(" ") %>%
  sapply("[", 1))

# unique genera
selected_genera <- species_dictionary[["genus_search"]] %>% unique

# these are unreliable selected_genera:
unreliable_genera <- intersect(selected_genera, duplicated_genera)

# ensuring a cleaner newick file with only necessary data
# this is actually really important
tree_85k[["node.label"]] <- NULL
tree_85k[["edge.length"]] <- NULL

# replacing timetree's underscores with spaces
tree_85k[["tip.label"]] %<>% str_replace_all("_", " ")

# storing genus
tree_85k[["tip.genus"]] <- sapply(strsplit(tree_85k[["tip.label"]]," "), "[", 1)
tree_85k_genera <- tree_85k[["tip.genus"]] %>% unique
```

```
# subtracting unreliable genera
tree_85k_genera %<>% setdiff(unreliable_genera)

# keeping only selected genera, including unreliable ones
tree_genus <- tree_85k %$% keep.tip(., tip.label[tip.genus %in% selected_genera])
tree_genus[["tip.genus"]] <- sapply(strsplit(tree_genus[["tip.label"]]," "), "[", 1)

# unfound species which genera are present in the 85k tree
unfound_species <- species_dictionary %>%
  filter(is.na(timetree_name) & genus_search %in% tree_85k_genera)
```

Once we figured out which species have proxy genera in the complete data, we can start filling them in as sister branches.

```
# for each unfound species which genus is present in the 85k tree,
for(i in 1:nrow(unfound_species)){
  # we search for all species of this genus ("sister species") in the 85k tree
  # this part is tricky because bind.tip rebuilds the tree from scratch
  # so we need to keep removing underscores. there are better ways to do this.
  tip_genus <- tree_genus[["tip.label"]] %>% strsplit("[_ ]") %>% sapply("[", 1)
  sister_species <- tree_genus[["tip.label"]][tip_genus == unfound_species[[i, "genus_search"]]]
  # we obtain the sister_species' most recent common ancestor (MRCA)
  # c(.[1]) is a hack because the MRCA function only works with at least 2 nodes
  where <- getMRCA(tree_genus, sister_species %>% c(.[1]))
  # and then add a leaf node linked to this MRCA
  tree_genus %<>% bind.tip(tip.label = unfound_species[[i, "ncbi_name"]], where = where)
}

# for some reason bind.tip adds underscores to species names
tree_genus[["tip.label"]] %<>% str_replace_all("_", " ")

# keeping track of found species
found_species <- species_dictionary %>% filter(!is.na(timetree_name) | genus_search %in% tree_85k_genera)
# forced_name means it either was found in timetree or we forced it by looking at genera names
found_species %<>% mutate(forced_name = coalesce(timetree_name, ncbi_name))

# so we keep only found species in this tree we are building (timetree + forced by genera)
tree_genus %<>% keep.tip(found_species[["forced_name"]])

# which found_species rows correspond to each tip.label?
match_tiplabel_name <- match(tree_genus[["tip.label"]], found_species[["forced_name"]])

tree_genus %<>% list_modify(
# converting to ncbi taxids
  tip.label = found_species[["new_taxid"]][match_tiplabel_name]
)
```

Species of unfound genera

In this part, we try to fill in the remaining missing species (those which genera were not found in TimeTree) by searching for their closest relatives (according to NCBI Taxonomy) that are present in the current tree. Once we find its two closest relatives, we can add the missing species as a branch from their LCA. This is a conservative approach.

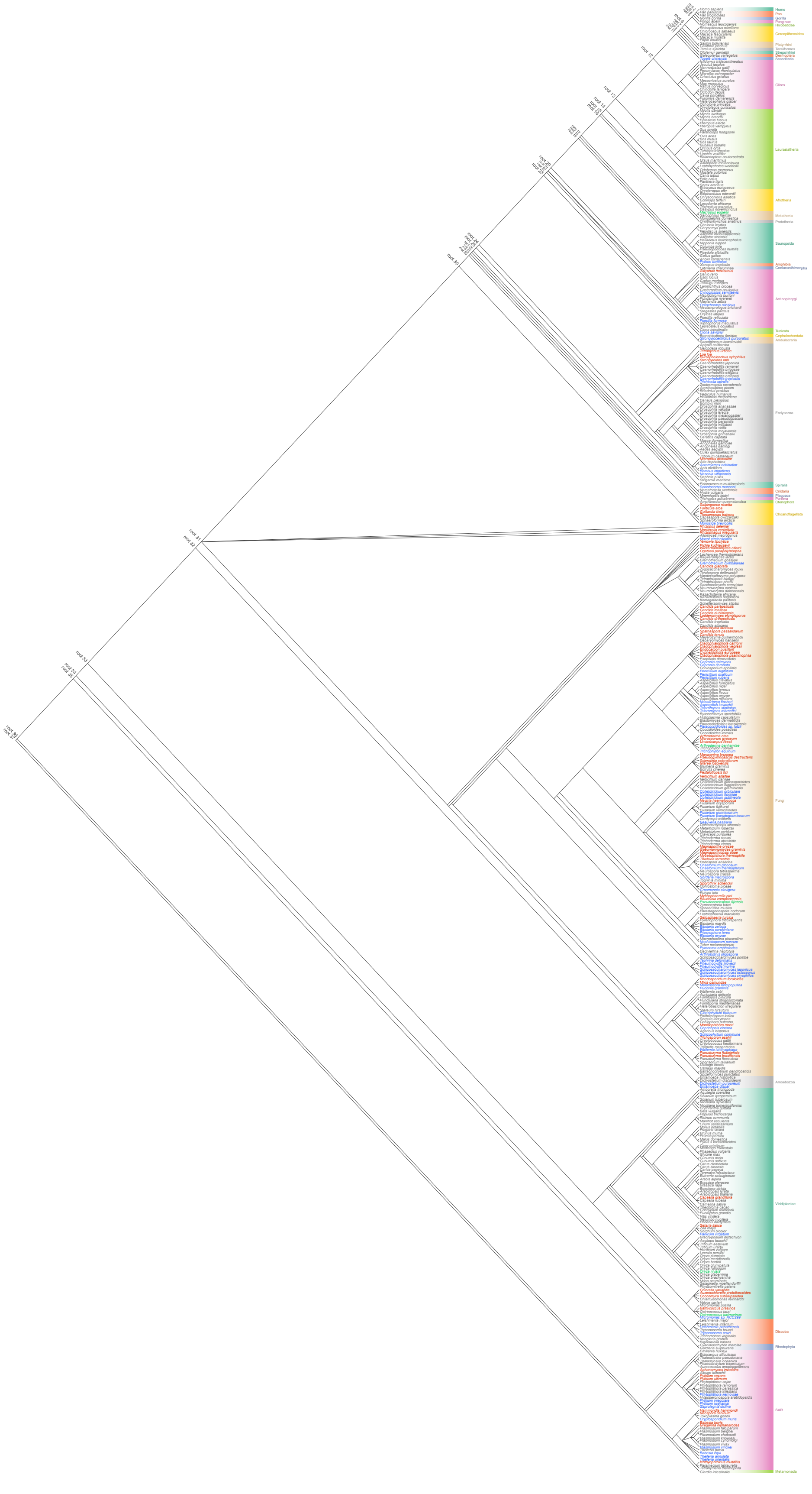
```
# converting ncbi phylo to igraph
graph_ncbi <- read.tree("tree_ncbi.nwk") %>% as.igraph.phylo(directed = TRUE)

# converting phylo to igraph
graph_genus <- as.igraph.phylo(tree_genus, directed = TRUE)

# for each species which genus is not in timetree
# we'll look for its two closest species (in the NCBI tree) which are present in the tree_genus we just built
unfound_genera <- species_dictionary %>% filter(is.na(timetree_name) & !genus_search %in% tree_85k_genera)

# this is the igraph equivalent of "phylo_tree$tip.label"
tip_nodes <- V(graph_ncbi)[degree(graph_ncbi, mode = "out") == 0]

# undirected distances between all species nodes
tip_distances <- graph_ncbi %>%
  distances(v = tip_nodes, to = tip_nodes, mode = "all") %>%
  as_tibble(rownames = "from") %>%
  pivot_longer(-from, names_to = "to", values_to = "distance")

# removing self references (zero distances)
tip_distances %<>% filter(distance > 0)
```

```r
# we only want to search for species of unfound genera
tip_distances %<>% inner_join(unfound_genera %>% select(from = new_taxid))

# we only want to find species already present in the genus_tree
tip_distances %<>% inner_join(found_species %>% select(to = new_taxid))

# we only want the two closest relatives
tip_distances %<>%
  group_by(from) %>%
  top_n(-2, distance) %>% # top 2 smallest distances
  top_n(2, to) # more than 2 species have the same smallest distance, so we get the first ones

# out distance matrix between all nodes in tree, needed to find MRCAs
out_distances <- graph_genus %>% distances(mode = "out")

# for each species of unfound genera,
# we find the MRCA for its two closest relatives
unfound_genera_mrca <- tip_distances %>% group_by(from) %>% summarise(mrca = {
  # which rows have no infinite distances? the last one represents the MRCA
  mrca_row_index <- max(which(rowSums(is.infinite(out_distances[, to])) == 0))
  rownames(out_distances)[mrca_row_index]
})

# adding unfound genera species nodes
graph_genus %<>% add_vertices(nrow(unfound_genera_mrca), color = "red", attr = list(name = unfound_genera_mrca[["from"]]))

# defining unfound genera species edges
# edges_to_add[1] -> edges_to_add[2], edges_to_add[2] -> edges_to_add[3]...
edges_to_add <- V(graph_genus)[unfound_genera_mrca %>% select(mrca, from) %>% t %>% as.vector]$name

# connecting species leafs to the supposed MRCA
graph_genus %<>% add_edges(V(graph_genus)[edges_to_add])

# plotting
# plot(as.undirected(graph_genus), layout = layout_as_tree(graph_genus), vertex.label = NA, vertex.size=2)

# finally converting to phylo format
phylo_graph_genus <- treeio::as.phylo(graph_genus)

# which species_dictionary rows correspond to each tip.label?
match_tiplabel_taxid <- match(phylo_graph_genus[["tip.label"]], species_dictionary[["new_taxid"]])

phylo_graph_genus %<>% list_modify(
  # adding tip.alias (this is not exported with write.tree)
  tip.alias = species_dictionary[["string_name"]][match_tiplabel_taxid],
  # converting back to string ids
  tip.label = species_dictionary[["taxid"]][match_tiplabel_taxid]
)

# ensuring a cleaner newick file with only necessary data
phylo_graph_genus[["node.label"]] <- NULL
phylo_graph_genus[["edge.length"]] <- NULL

# usethis::use_data(phylo_graph_genus, overwrite = TRUE)
# write.tree(phylo_graph_genus, "../data/hybrid_tree.nwk")
```

Ctenophora as sister to all animals

According to TimeTree, Ctenophora remains as a sister group to Cnidaria. We believe the most recent consensus in literature is to consider them a sister group to all animals. The following code block moves *Mnemiopsis leidyi*, the only ctenophore in our analysis, to the base of the metazoan lineage.

```r
# reordering tip.labels
from_to <- c(
  "400682" = "27923",  # amphimedon to mnemiopsis
  "10228"  = "400682", # trichoplax to amphimedon
  "27923"  = "10228"   # mnemiopsis to trichoplax
)

modified_phylo <- phylo_graph_genus

modified_phylo[["tip.label"]] %<>% recode(!!!from_to)

write.tree(modified_phylo, "../data/hybrid_tree_modified.nwk")
```



Supplementary Figure 1: Complete 476 eukaryotes tree. Green species have been filled in by a genus proxy in TimeTree. Red species have been filled in by looking at NCBI Taxonomy. Clade naming is described further in this document.

Gene selection and annotation

The anchoring point of our analysis is gene identifiers. This section describes the process of structuring an identifier dictionary as well as other useful annotation, like KEGG pathways. The goal is to have a table to which all kinds of additional data can be left joined.

Neurotransmitter systems annotation

We start by querying the KEGG api for the pathways of interest. Results are pivoted to a wider format.

Table 5: All links between genes and pathways in KEGG.

`link_pathway_entrez`

#	Col. name	Col. type	Used?	Example	Description
1	entrez_id	character	yes	hsa:10411	NCBI Taxonomy identifier
2	pathway_id	character	yes	path:hsa04726	KEGG pathway ID

Location: data-raw/download/link_pathway_entrez.tsv
Source: http://rest.kegg.jp/link/pathway/hsa

```r
pathways <- tribble(
  ~pathway_id,       ~pathway_name,
  "path:hsa04724",  "glutamatergic",
  "path:hsa04725",  "cholinergic",
  "path:hsa04726",  "serotonergic",
  "path:hsa04727",  "gabaergic",
  "path:hsa04728",  "dopaminergic"
)

# Removing hsa prefix
link_pathway_entrez[["entrez_id"]] %<>% str_split_n(":", 2)

# Filtering for pathways of interest and pivoting
gene_pathways <- inner_join(link_pathway_entrez, pathways) %>%
  mutate(n = 1) %>%
  pivot_wider(
    id_cols     = entrez_id,
    names_from  = pathway_name,
    values_from = n,
    values_fn   = list(n = length),
    values_fill = list(n = 0)
  ) %>%
  mutate(system_count = rowSums(select(., -entrez_id)))

# Exporting for package use
usethis::use_data(gene_pathways, overwrite = TRUE)
```

`tail(gene_pathways)`

entrez_id	glutamatergic	cholinergic	serotonergic	gabaergic	dopaminergic	system_count
805	0	0	0	0	1	1
808	0	0	0	0	1	1
810	0	0	0	0	1	1
84152	0	0	0	0	1	1
91860	0	0	0	0	1	1
9575	0	0	0	0	1	1

Base ID lookup table

Now we start building a base ID lookup table containing entrez gene IDs, STRING ensembl protein IDs, ensembl gene IDs, STRING protein names and entrez gene names. Every piece of data in subsequent analyses will be progressively joined to it.

Table 6: Conversion dictionary from entrez ID to STRING's ensembl protein ID.

`link_entrez_string`

#	Col. name	Col. type	Used?	Example	Description
1	taxid	numeric	no	9606	NCBI Taxonomy ID
2	entrez_id	numeric	yes	7157	entrez gene ID
3	string_id	character	yes	9606.ENSP00000269305	STRING ID

Location: data-raw/download/human.entrez_2_string.2018.tsv.gz
Source: https://string-db.org/mapping_files/entrez/human.entrez_2_string.2018.tsv.gz

Table 7: Conversion dictionary from STRING ID to protein name.

`string_names`

#	Col. name	Col. type	Used?	Example	Description
1	taxid	numeric	no	9606	NCBI Taxonomy ID
2	string_name	character	yes	TP53	protein name
3	string_id	character	yes	9606.ENSP00000269305	STRING ID

Location: data-raw/download/human.name_2_string.tsv.gz
Source: https://string-db.org/mapping_files/STRING_display_names/human.name_2_string.tsv.gz

Table 8: Conversion dictionary from entrez ID to gene name.

`entrez_names`

#	Col. name	Col. type	Used?	Example	Description
1	taxid	numeric	no	9606	taxon ID
2	entrez_id	character	yes	7157	entrez gene ID
3	entrez_name	character	yes	TP53	gene name
4	...	...	no	...	(too many unrelated fields)

Location: data-raw/download/Homo_sapiens.gene_info.gz
Source: https://ftp.ncbi.nlm.nih.gov/gene/DATA/GENE_INFO/Mammalia/Homo_sapiens.gene_info.gz

Table 9: Conversion dictionary from entrez ID to ensembl gene (ENSG) ID.

`link_ensembl_entrez`

#	Col. name	Col. type	Used?	Example	Description
1	entrez_id	character	yes	hsa:7157	entrez gene ID
2	ensembl_id	character	yes	ensembl:ENSG00000141510	ensembl gene ID

Location: data-raw/download/link_ensembl_entrez.tsv
Source: http://rest.genome.jp/link/ensembl/hsa

```r
# Removing all kegg prefixes (e.g. "hsa:")
link_ensembl_entrez %<>% mutate_all(str_split_n, ":", 2)

# Joining all data
gene_ids ← gene_pathways %>%
  select(entrez_id) %>%
  left_join(link_ensembl_entrez) %>%
  left_join(link_entrez_string) %>%
  left_join(string_names) %>%
  left_join(entrez_names)
```

Some STRING proteins couldn't be automatically resolved, so we resolve them manually

```r
gene_ids[!complete.cases(gene_ids),]
```

entrez_id	ensembl_id	string_id	string_name	entrez_name
100137049	ENSG00000243708	NA	NA	PLA2G4B
85358	ENSG00000251322	NA	NA	SHANK3
8681	ENSG00000168970	NA	NA	JMJD7-PLA2G4B
1139	ENSG00000175344	NA	NA	CHRNA7
107987478	NA	NA	NA	LOC107987478
107987479	NA	NA	NA	LOC107987479
1564	ENSG00000205702	NA	NA	CYP2D7
801	ENSG00000198668	NA	NA	CALM1
805	ENSG00000143933	NA	NA	CALM2
808	ENSG00000160014	NA	NA	CALM3

```r
complete_info ← tribble(
##########################################################################################
 ~entrez_id,       ~ensembl_id,             ~string_id,    ~string_name,    ~entrez_name,#
"100137049", "ENSG00000243708", "9606.ENSP00000396045",       "PLA2G4B",       "PLA2G4B",#
    "85358", "ENSG00000251322",                      NA,              NA,        "SHANK3",#
     "8681", "ENSG00000168970", "9606.ENSP00000371886", "JMJD7-PLA2G4B", "JMJD7-PLA2G4B",#
     "1139", "ENSG00000175344", "9606.ENSP00000407546",        "CHRNA7",        "CHRNA7",#
"107987478",                NA,                      NA,              NA,  "LOC107987478",#
"107987479",                NA,                      NA,              NA,  "LOC107987479",#
     "1564", "ENSG00000205702",                      NA,              NA,        "CYP2D7",#
      "801", "ENSG00000198668", "9606.ENSP00000349467",         "CALM1",         "CALM1",#
      "805", "ENSG00000143933", "9606.ENSP00000272298",         "CALM2",         "CALM2",#
      "808", "ENSG00000160014", "9606.ENSP00000291295",         "CALM3",         "CALM3" #
)#########################################################################################

# Removing incomplete cases and adding updated ones
gene_ids %<>% na.omit %>% bind_rows(complete_info)

# Removing taxid prefix from STRING IDs
gene_ids[["string_id"]] %<>% str_split_n("\\.", 2)

# Exporting for package use
usethis::use_data(gene_ids, overwrite = TRUE)
```

Neuroexclusivity

Neuroexclusivity data consists of gene expression collected from Gene Expression Atlas and the KEGG pathways themselves. This data is cleaned and subsetted prior to analysis.

Expression neuroexclusivity

Multiple wide .tsv files are preprocessed into a single long data frame. We also create a template file for manually classifying tissues into nervous or non-nervous categories.

Resources

We start by searching Gene Expression Atlas for experiments that have human baseline expression data at the tissue level. For each experiment, TPM expression data is downloaded to the `data-raw/download/gxa/` directory. The following 8 experiments could be found (hyperlinked):

- E-MTAB-513
- E-MTAB-2836
- E-MTAB-3358
- E-MTAB-3708
- E-MTAB-3716
- E-MTAB-4344
- E-MTAB-4840
- E-MTAB-5214

Reshaping

Loading and pivoting all data to a long format.

```
# Loading
gene_expression ← sapply(
  list.files("download/gxa/", full.names = T),
  read_tsv,
  comment = "#",
  simplify = FALSE,
  USE.NAMES = TRUE
)

# Pivoting
gene_expression %<>%
  map_dfr(pivot_longer, cols = -(1:2), names_to = "tissue", values_to = "tpm") %>%
  na.omit %>%
  select(ensembl_id = `Gene ID`, tissue, tpm)
```

Cleaning

A lot of tissue annotation can be collapsed into single levels (e.g. “brain” and “brain fragment” can be considered the same tissue). The cleaning is performed and expression data is exported for analysis.

```
# E-MTAB-4840 has comma separated developmental stage info (removing everything before ", ")
gene_expression %<>% mutate(tissue = str_remove(tissue, "^.+, "))

tissue_names_fix ← c(
  "brain fragment"                     = "brain",
  "forebrain fragment"                 = "forebrain",
  "forebrain and midbrain"             = "forebrain",
  "hindbrain fragment"                 = "hindbrain",
  "hindbrain without cerebellum"       = "hindbrain",
  "hippocampus proper"                 = "hippocampus",
  "hippocampal formation"              = "hippocampus",
  "diencephalon and midbrain"          = "diencephalon",
  "visceral (omentum) adipose tissue"  = "adipose tissue",
  "subcutaneous adipose tissue"        = "adipose tissue",
  "spinal cord (cervical c-1)"         = "spinal cord",
  "C1 segment of cervical spinal cord" = "spinal cord"
)

gene_expression %<>% mutate(tissue = recode(tissue, !!!tissue_names_fix))

# Subseting for genes of interest
gene_expression %<>% filter(ensembl_id %in% gene_ids[["ensembl_id"]])

# Exporting for package use
usethis::use_data(gene_expression, overwrite = TRUE)
```

Tissue classification

For subsequent analyses, we need to distinguish if a tissue is part of the nervous system or not. This is done by hand. The first step is to write a temp file to `data-raw/temp/temp_tissue_classification.tsv` with all tissue names. This serves as a base for the completed `data/neuroexclusivity_classification_tissue` file.

```
gene_expression %>%
  select(tissue) %>%
  unique %>%
  arrange %>%
  mutate(is_nervous = NA) %>%
  write_tsv("temp/temp_tissue_classification.tsv")
```

Pathway neuroexclusivity

In this section we create a template file for classifying pathways into nervous or non-nervous.

Resources For `link_pathway_entrez` see Table 5.

Table 10: KEGG pathway names.

pathway_names					
#	Col. name	Col. type	Used?	Example	Description
1	pathway_id	character	yes	path:hsa04726	KEGG pathway ID
2	pathway_name	character	yes	Serotonergic synapse - Homo sapiens (human)	pathway name

Location: data-raw/download/pathway_names.tsv
Source: http://rest.kegg.jp/list/pathway/hsa

Pathway classification

Just like tissues, we need to distinguish if a pathway is related to the nervous system or not. This is done by hand. The first step is to write a temp file to `data-raw/temp/temp_pathway_classification.tsv` with all pathway names. This serves as a base for the completed `data/neuroexclusivity_classification_pathway.tsv` file.

```
# Removing species prefix ("hsa:")
link_pathway_entrez[["entrez_id"]] %<>% str_split_n("\\:", 2)

selected_genes_pathways ← link_pathway_entrez %>% filter(entrez_id %in% gene_ids[["entrez_id"]])

unique_pathway_ids ← selected_genes_pathways %>% pull(pathway_id) %>% unique

pathway_names %<>% filter(pathway_id %in% unique_pathway_ids) %>%
  mutate(is_nervous = NA) %>%
  write_tsv("temp/temp_pathway_classification.tsv")
```

Orthology data

This section refers to orthology data exported for geneplast use. Essentially, we subset the global STRING mapping between proteins and orthologous groups into a smaller dataset containing only information about orthogroups related to eukaryotic species.

Table 11: Orthologous groups (COGs, NOGs, KOGs) and their proteins.

cogs					
#	Col. name	Col. type	Used?	Example	Description
1	taxid.string_id	character	yes	9606.ENSP00000269305	STRING protein ID
2	start_position	numeric	no	1	residue where orthogroup mapping starts
3	end_position	numeric	no	393	residue where orthogroup mapping ends
4	cog_id	character	yes	NOG08732	orthologous group ID
5	protein_annotation	character	no	Cellular tumor antigen p53; [...]	protein description

Location: data-raw/download/COG.mappings.v11.0.txt.gz
Source: https://stringdb-static.org/download/COG.mappings.v11.0.txt.gz

```r
# Spliting first column into taxid and string_id
cogs %<>% separate(taxid.string_id, into = c("taxid","string_id"), sep = "\\.", extra = "merge")

# Subsetting cogs of interest
gene_cogs <- cogs %>% filter(string_id %in% gene_ids[["string_id"]]) %>% select(-taxid)

cogs %<>%
  # leave only eukaryotes
  filter(taxid %in% string_eukaryotes[["taxid"]])

# Exporting for package use
usethis::use_data(cogs, overwrite = TRUE)
usethis::use_data(gene_cogs, overwrite = TRUE)
```

Network

We use our gene dictionary to retrieve the corresponding human protein-protein interaction network. This is done through the STRING API. In the end, we recompute combined interaction scores.

Retrieving network data

Querying the API endpoint for the STRING IDs we previously resolved:

```r
identifiers <- gene_ids %>% pull(string_id) %>% na.omit %>% paste0(collapse="%0d")

if (!file.exists("download/string_ids.tsv")) {
    postForm(
      "http://string-db.org/api/tsv/get_string_ids"
      ,identifiers = identifiers
      ,echo_query  = "1"
      ,species     = "9606"
    ) %>%
    write("download/string_ids.tsv")
}
```

Table 12: STRING interaction network with channel specific scores.

		string_ids			
#	Col. name	Col. type	Used?	Example	Description
1	queryItem	character	yes	ENSP00000258400	queried term
2	queryIndex	numeric	yes	266	index of queried term
3	stringId	character	yes	9606.ENSP00000258400	STRING ID
4	ncbiTaxonId	numeric	yes	9606	NCBI Taxonomy ID
5	taxonName	character	yes	Homo sapiens	species name
6	preferredName	character	yes	HTR2B	common protein name
7	annotation	character	yes	5-hydroxytryptamine receptor 2B; [...]	protein annotation

Location: data-raw/download/string_ids.tsv
Source: http://string-db.org/api/tsv/get_string_ids

Making sure that the API succesfully resolves queried protein IDs.

```r
api_ids <- read_tsv("download/string_ids.tsv", comment = "", quote = "")

# Removing taxid prefix
api_ids %<>% mutate(stringId = str_split_n(stringId, "\\.", 2))

# Removing inexact matches (queried id is different from resolved id)
api_ids %<>% group_by(queryItem) %>% filter(queryItem == stringId)
```

```r
# Setequal must return true if ids matched exatcly
setequal(
  gene_ids %>% pull(string_id) %>% na.omit,
  api_ids  %>% pull(stringId)
) %>% stopifnot
```

Once IDs are correctly resolved, we can query the network API endpoint to obtain the protein-protein interaction edgelist.

```r
# It is important to query this endpoint with the species prefix ("9606.")
identifiers <- api_ids %>% pull(stringId) %>% na.omit %>% { paste0("9606.", ., collapse="%0d") }

if (!file.exists("download/string_edgelist.tsv")) {
    postForm(
      "http://string-db.org/api/tsv/network"
      ,identifiers = identifiers
      ,species     = "9606"
    ) %>%
    write("download/string_edgelist.tsv")
}
```

Table 13: STRING interaction network with channel specific scores.

`string_edgelist`

#	Col. name	Col. type	Used?	Example	Description
1	stringId_A	character	yes	ENSP00000215659	STRING ID (protein A)
2	stringId_B	character	yes	ENSP00000211287	STRING ID (protein B)
3	preferredName_A	character	yes	MAPK12	common protein name (protein A)
4	preferredName_B	character	yes	MAPK13	common protein name (protein B)
5	ncbiTaxonId	numeric	yes	9606	NCBI Taxonomy ID
6	score	numeric	yes	0.948	combined score
7	nscore	numeric	yes	0	gene neighborhood score
8	fscore	numeric	yes	0	gene fusion score
9	pscore	numeric	yes	0.014223	phylogenetic profile score
10	ascore	numeric	yes	0	coexpression score
11	escore	numeric	yes	0.485	experimental score
12	dscore	numeric	yes	0.9	database score
13	tscore	numeric	yes	0.02772	textmining score

Location: data-raw/download/string_edgelist.tsv
Source: http://string-db.org/api/tsv/network

Recomputing scores

From string-db.org:

> "In STRING, each protein-protein interaction is annotated with one or more 'scores'. Importantly, these scores do not indicate the strength or the specificity of the interaction. Instead, they are indicators of confidence, i.e. how likely STRING judges an interaction to be true, given the available evidence. All scores rank from 0 to 1, with 1 being the highest possible confidence."

For the sake of this project, we will only use experimental and database scores with a combined value >= 0.7, a high confidence threshold according to the STRING database. The combined score is given by the following expression, as stated in von Mering C et al, 2005:

$$S = 1 - \prod_i (1 - S_i)$$

```r
string_edgelist <- read_tsv("download/string_edgelist.tsv")

string_edgelist %<>%
  mutate(cs = combine_scores(., c("e","d"))) %>%
  filter(cs >= 0.7) %>%
  select(stringId_A, stringId_B)

# How many edgelist proteins are absent in gene_ids? (should return 0)
setdiff(
  string_edgelist %$% c(stringId_A, stringId_B),
  gene_ids %>% pull(string_id)
) %>% { stopifnot(. == 0) }

# Exporting for package use
usethis::use_data(string_edgelist, overwrite = TRUE)
```

Analysis

This section focuses on analyses and results reported in the main manuscript.

Root inference

To estimate the evolutionary root of a given gene (i.e. the ancestor from which its genetic archetype was vertically inherited), we use orthologous group annotation from the STRING database. The presence and absence of an orthologous group in a species tree is used to determine its most likely evolutionary root. Loading initial resources:

```r
library(tidyverse)
library(magrittr)
library(geneplast)
library(ape)
library(XML)
library(rentrez)
library(neurotransmissionevolution)

data(
  cogs,
  gene_cogs,
  string_eukaryotes,
  package = "neurotransmissionevolution"
)

phyloTree <- read.tree("../data/hybrid_tree_modified.nwk") %>% rotatePhyloTree("9606")
```

We perform some minor data formatting before feeding it to geneplast

```r
# Formating cogdata column names for geneplast
cogs %<>% rename(protein_id = string_id, ssp_id = taxid) %>% select(protein_id, ssp_id, cog_id)

# Adding species names to taxid tree
phyloTree %<>% list_modify(
  tip.alias = string_eukaryotes %$% string_name[match(phyloTree[["tip.label"]], taxid)]
)
```

Geneplast

Geneplast's `groot.preprocess` function structures an `ogr` object on which `groot` will perform the rooting. We then retrieve the numeric root (`groot.get("results")`) for the `cogs_of_interest`, that is, orthologous groups pertaining to neurotransmission genes.

```r
cogs_of_interest <- gene_cogs %>% pull(cog_id) %>% unique

ogr <- groot.preprocess(
  cogdata   = cogs,
  phyloTree = phyloTree,
  spid      = "9606",
  cogids    = cogs_of_interest
)

roots <- groot(ogr, nPermutations = 1) %>%
  groot.get("results") %>%
  rownames_to_column("cog_id") %>%
  select(cog_id, root = Root) %>%
  write_tsv("geneplast_roots.tsv")
```

Clade names

Each root branches to a clade that diverged from humans some time in the past. It is nice to have these clades taxonomically named to ease our interpretation. Unlike NCBI Taxonomy, TimeTree's internal nodes are not named. Therefore, we query the NCBI Taxonomy API to try to find most clade names automatically. It is important to note that we are using a hybrid tree primarily built from TimeTree data. This means NCBI Taxonomy naming will not perfectly match clades in our tree. For instance, root #36 branches to a clade containing 38 species from the SAR supergroup, but also 1 species from the Haptista rank, namely *Emiliania huxleyi*. The Haptista group is a sister clade to SAR, so it might be the case that *Emiliania huxleyi* is actually correctly placed together with SAR species by TimeTree, given their evolutionary proximity. Resolving these naming conflicts is not trivial and falls out of our scope.

```r
lineages <- entrez_fetch(
  db      = "taxonomy",
  id      = string_eukaryotes[["new_taxid"]],
  rettype = "xml",
  retmode = "xml",
  parsed  = TRUE
)

string_eukaryotes %<>% mutate(
  root        = ogr@tree$tip.group[taxid],
  lineage_txt = xpathSApply(lineages, "//Lineage", XML::xmlValue)
)

write(string_eukaryotes[["lineage_txt"]], "temp/species_lineages.txt")

root_names <- string_eukaryotes %>%
  # Long format lineages
  mutate(
     lineage_split = strsplit(lineage_txt, "; ")
  ) %>%
  unnest_longer(
     col        = lineage_split
    ,values_to  = "clade_name"
    ,indices_to = "clade_depth"
  ) %>%
  # Counting and dropping the last group
  group_by(root, clade_depth, clade_name) %>%
  tally(sort = TRUE) %>%
  # Collapsing lineages by clade depths
  summarise(
     diverging_rank = n_distinct(clade_name) > 1
    ,clade_name     = ifelse(diverging_rank, paste0(clade_name, " (", n,")", collapse = "; "), clade_name)
  ) %>%
  # Removing diverging ranks after the first one
  filter(cumsum(diverging_rank) <= 1) %>%
  # Removing irrelevant basal ranks (eg Eukaryota)
  group_by(clade_depth) %>%
  arrange(root) %>%
  filter(!(duplicated(clade_name) | duplicated(clade_name, fromLast = TRUE)) | diverging_rank) %>%
  # Choosing name
  group_by(root) %>%
  summarise(clade_name = first(clade_name, order_by = clade_depth)) %>%
  write_tsv("temp/temp_geneplast_clade_names.tsv")
```

Some automatically named clades are resolved by hand. The following table shows clade names before and after manual checking:

```
# Loading manually resolved names, based on temp/temp_geneplast_clade_names.tsv
lca_names <- read_tsv("geneplast_clade_names.tsv")

root_names %>%
  rename("automatic_name" = clade_name) %>%
  inner_join(lca_names) %>%
  rename("corrected_name" = clade_name) %>%
  knitr::kable(caption = "Clade names before and after manual checking.", booktabs = TRUE, linesep = "") %>%
  kableExtra::kable_styling(position = "left", latex_options = c("striped", "HOLD_position"))
```

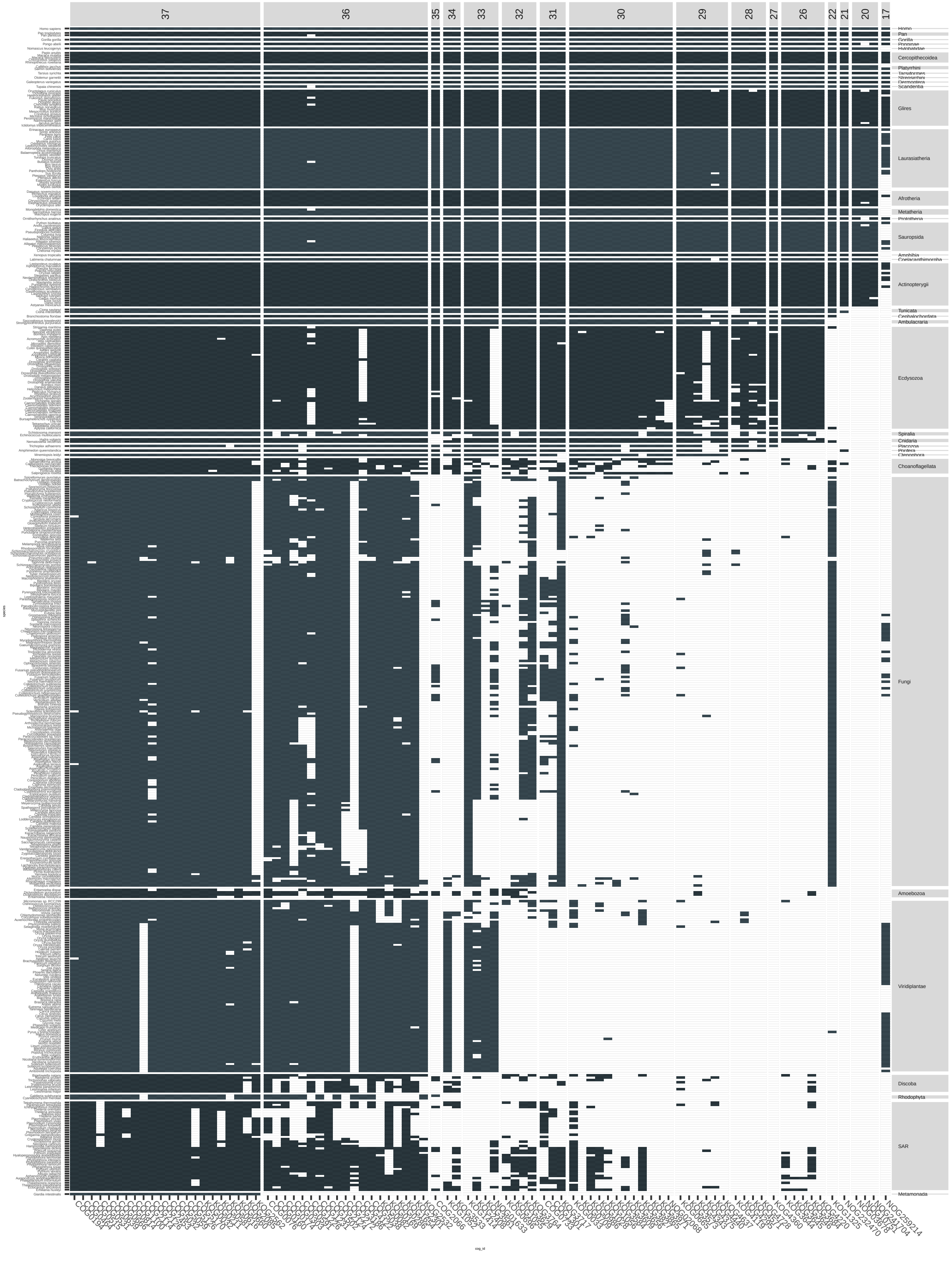
Table 14: Clade names before and after manual checking.

root	automatic_name	corrected_name
1	Homo	Homo
2	Pan	Pan
3	Gorilla	Gorilla
4	Ponginae	Ponginae
5	Hylobatidae	Hylobatidae
6	Cercopithecoidea	Cercopithecoidea
7	Platyrrhini	Platyrrhini
8	Tarsiiformes	Tarsiiformes
9	Strepsirrhini	Strepsirrhini
10	Dermoptera	Dermoptera
11	Scandentia	Scandentia
12	Glires	Glires
13	Laurasiatheria	Laurasiatheria
14	Afrotheria (6); Xenarthra (1)	Afrotheria
15	Metatheria	Metatheria
16	Prototheria	Prototheria
17	Sauropsida	Sauropsida
18	Amphibia	Amphibia
19	Coelacanthimorpha	Coelacanthimorpha
20	Actinopterygii	Actinopterygii
21	Tunicata	Tunicata
22	Cephalochordata	Cephalochordata
23	Echinodermata (1); Hemichordata (1)	Ambulacraria
24	Ecdysozoa (43); Spiralia (2)	Ecdysozoa
25	Spiralia	Spiralia
26	Cnidaria	Cnidaria
27	Placozoa	Placozoa
28	Porifera	Porifera
29	Ctenophora	Ctenophora
30	Opisthokonta (5); Apusozoa (1); Cryptophyceae (1)	Choanoflagellata
31	Fungi	Fungi
32	Amoebozoa	Amoebozoa
33	Viridiplantae	Viridiplantae
34	Discoba (6); Metamonada (1); Sar (1)	Discoba
35	Rhodophyta	Rhodophyta
36	Sar (38); Haptista (1)	SAR
37	Metamonada	Metamonada

Phyletic patterns

Visualizing the presence/absence matrix according to inferred roots and species' clades

```r
lca_names %<>% rename("lca" = root)

lca_spp <- ogr@spbranches %>%
  rename("taxid" = ssp_id, "species" = ssp_name, "lca" = `9606`) %>%
  mutate(taxid_order = row_number())

# Saving for use in abundance computation
lca_spp %>% select(lca, taxid, taxid_order) %>% write_tsv("geneplast_clade_taxids.tsv")

cog_pam <- ogr@orthoct[,-1]

long_pam <- cog_pam %>%
  rownames_to_column("taxid") %>%
  pivot_longer(-taxid, names_to = "cog_id") %>%
  left_join(lca_spp) %>%
  left_join(lca_names) %>%
  left_join(roots) %>%
  mutate(
    cog_id       = fct_reorder(cog_id, root),
    species      = fct_reorder(species, desc(taxid_order)),
    clade_name   = fct_reorder(clade_name, lca),
    root         = as_factor(root),
    clade_stripe = as.numeric(as_factor(lca)) %% 2 == 0
  ) %>%
  # Stripe every other species
  group_by(cog_id) %>%
  mutate(spp_stripe = as.numeric(species) %% 2 == 0) %>%
  # Removing empty tiles
  filter(value == 1) %>%
  # Stripe every other cog
  group_by(taxid) %>%
  mutate(cog_stripe = as.numeric(cog_id) %% 2 == 0)

ggplot(long_pam, aes(x = cog_id, y = species)) +
  geom_tile(aes(fill = clade_stripe + 0.3 * xor(spp_stripe, cog_stripe))) +
  scale_fill_gradient(low = "#37474F", high = "#263238") +
  facet_grid(clade_name ~ fct_rev(root), scales = "free", space='free') +
  theme(
    text               = element_text(size = 2),
    panel.spacing      = unit(1, "pt"),
    panel.grid.major.x = element_blank(),
    strip.background   = element_rect(colour = "#FFFFFF"),
    strip.text.x       = element_text(size = 6, angle = 90),
    strip.text.y       = element_text(size = 3, angle = 0, hjust = 0, lineheight = 3),
    axis.text.x        = element_text(size = 5, angle = -45, vjust = 0, hjust = 0),
    legend.position    = "none"
  )
```



Supplementary Figure 2: Presence of orthologous groups in species. The horizontal axis is grouped by COGs rooted at some specific LCA. The vertical axis is grouped by species' clades. A checkerboard pattern is superimposed to aid visual examination.

Neuroexclusivity

We characterize the relevance of a gene to the nervous system by inspecting what proportion of its activity is related to nervous processes. To do this, we rely on tissue RNA-Seq data, as well as KEGG's pathways themselves. Loading resources:

```r
library(tidyverse)
library(magrittr)

data(
   gene_ids
  ,gene_pathways
  ,gene_expression
  ,package = "neurotransmissionevolution"
)
```

Expression neuroexclusivity

We start by averaging all `gene_expression` collected from the Expression Atlas by tissue (`tpm_avg`). The averaged expression is filtered for values greather than 0.5 TPM. This ensures further computations only account for tissues in which genes are actually expressed. Then, we add the manual tissue classification indicating which tissues are nervous or not (described in Preprocessing). The neuroexclusivity index for a gene is the sum of its `tpm_avg` values in nervous tissues divided by the sum its values in all tissues.

```r
tissue_classification ← read_tsv(
  file      = "../data/neuroexclusivity_classification_tissue.tsv"
 ,col_types = "ci"
)

# Averaging TPM expression by tissue
avg_by_tissue ← gene_expression %>%
  group_by(ensembl_id, tissue) %>%
  summarise(tpm_avg = mean(tpm)) %>%
  filter(tpm_avg ⩾ 0.5) %>%
  left_join(tissue_classification)

# Measuring expression neuroexclusivity
expression_neuroexclusivity ← avg_by_tissue %>%
  group_by(ensembl_id) %>%
  summarise(expression_neuroexclusivity = sum(tpm_avg[is_nervous == 1])/sum(tpm_avg)) %>%
  write_tsv("neuroexclusivity_expression.tsv")
```

Pathway neuroexclusivity

To find the pathway neuroexclusivity of a gene, we simply divide the count of nervous pathways by the count of all pathways it participates in.

```r
pathway_classification ← read_tsv(
   file      = "../data/neuroexclusivity_classification_pathway.tsv"
  ,col_types = "cci"
)

link_pathway_entrez ← read_tsv(
   file      = "../data-raw/download/link_pathway_entrez.tsv"
  ,col_names = c("entrez_id", "pathway_id")
  ,col_types = "cc"
)

# Removing "hsa:" prefix
link_pathway_entrez[["entrez_id"]] %<>% str_split_n("\\:", 2)

# Pathway data related to our genes of interest
selected_genes_pathways ← link_pathway_entrez %>%
  filter(entrez_id %in% gene_ids[["entrez_id"]]) %>%
  left_join(pathway_classification) %>%
  drop_na # Dropping general pathways
```

```r
# Measuring pathway neuroexclusivity
pathway_neuroexclusivity ← selected_genes_pathways %>%
  group_by(entrez_id) %>%
  summarise(pathway_neuroexclusivity = sum(is_nervous)/length(is_nervous)) %>%
  write_tsv("neuroexclusivity_pathway.tsv")
```

Network

This section describes the steps for reproducing most of the manuscript figures. A lot of it is concerned with plotting particularities and visual styling.

Graph data

Loading resources:

```r
# Data manipulation
library(tidyverse)
library(igraph)
library(magrittr)

# Plotting dependencies
library(scatterpie)
library(UpSetR)
library(gridExtra)
library(patchwork)

# Utils
library(neurotransmissionevolution)

# Packaged data
data(
   gene_ids
  ,gene_cogs
  ,gene_pathways
  ,string_edgelist
  ,package = "neurotransmissionevolution"
)

# Fresh analysis data
cog_roots                   ← read_tsv("geneplast_roots.tsv",             col_types = "ci")
clade_names                 ← read_tsv("geneplast_clade_names.tsv",       col_types = "ic")
pathway_neuroexclusivity    ← read_tsv("neuroexclusivity_pathway.tsv",    col_types = "cn")
expression_neuroexclusivity ← read_tsv("neuroexclusivity_expression.tsv", col_types = "cn")

# Collapsing similar functions
gene_annotation ← read_tsv("../data/gene_annotation.tsv", col_types = "cc") %>%
  mutate(annotation = case_when(
     grepl("clearance",   annotation) ~ "depletion"
    ,grepl("degradation", annotation) ~ "depletion"
    ,grepl("transport",   annotation) ~ "synthesis"
    ,TRUE ~ annotation
  ))
```

We start by joining all gene data and creating the graph object.

```r
# If a gene has more than 1 COG, select the oldest one.
# This is unusual, but can happen in cases of gene fusion, for instance.
gene_cogs %<>%
  inner_join(cog_roots) %>%
  group_by(string_id) %>%
  filter(root == max(root)) %>%
  inner_join(clade_names)

# Gathering all gene info available
vertices ← gene_ids %>%
  na.omit %>%
  inner_join(gene_cogs) %>%
  inner_join(gene_pathways) %>%
  inner_join(gene_annotation) %>%
  inner_join(pathway_neuroexclusivity) %>%
  inner_join(expression_neuroexclusivity) %>%
  mutate(ne = pathway_neuroexclusivity >= 0.9) %>%
  select(string_id, everything())

# Quick color hack to aid visualization
vertices %<>%
  unite(color, glutamatergic:dopaminergic, remove = F) %>%
  mutate(color = rainbow(color %>% n_distinct)[color %>% as.factor])

g <- graph_from_data_frame(string_edgelist, directed = F, vertices = vertices)

# Setting node sizes
V(g)$size <- V(g)$system_count %>% sqrt %>% multiply_by(5)
```

The following block calls an utility function that handles the force directed layout with the aid of a shiny web server and the VivaGraphJS javascript library. A computed layout is already available in this folder.

```r
if(file.exists("network_layout.tsv")) {
  layout <- read_tsv("network_layout.tsv", col_types = "dd") %>% as.matrix
} else {
  layout <- vivagraph(g, precompute_multiplier = 200, precompute_niter = 1000)
}

# inserting layout coordinates into graph object
V(g)$x <-  layout[, 1]
# layout matrix comes vertically flipped
V(g)$y <- -layout[, 2]
```

We use base ggplot2 to draw the network. Edges are represented by a common `geom_path` layer. The following block retrieves tidy edge coordinates for the `geom_path` calls.

```r
# Recreating the vertices data.frame, now with layout coordinates (lazy way)
vertices <- igraph::as_data_frame(g, what = "vertices") %>% rename(string_id = name)

# The edges data.frame will be used to draw lines with geom_path
edges <- string_edgelist %>%
    map(match, vertices[["string_id"]]) %>%
    map_dfr(~ vertices[.x,]) %>%
    select(x:y) %>%
    cbind(group = 1:nrow(string_edgelist))
```

Setting up reusable aesthetic parameters to avoid code duplication.

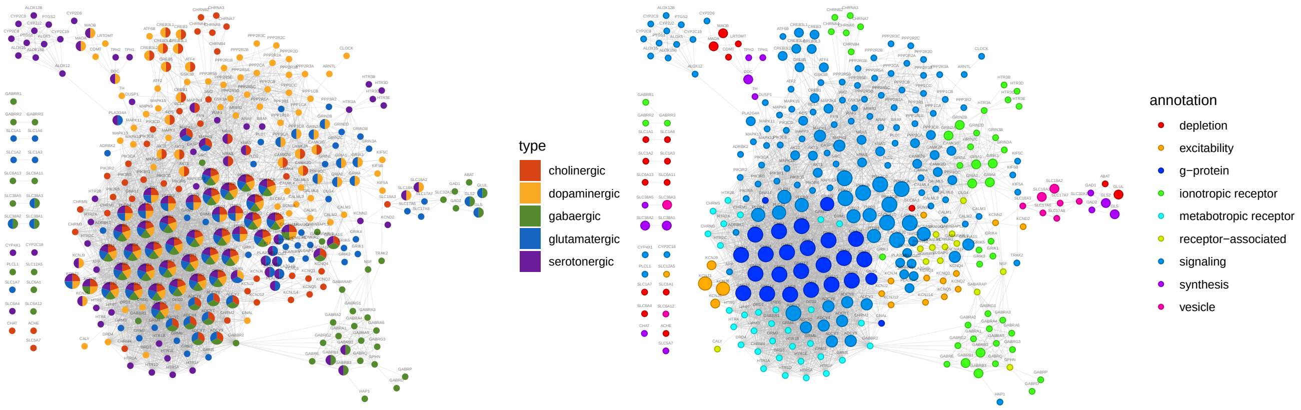
```r
pie_colors <- c(
   "cholinergic"   = "#D84315"
  ,"dopaminergic"  = "#F9A825"
  ,"gabaergic"     = "#558B2F"
  ,"glutamatergic" = "#1565C0"
  ,"serotonergic"  = "#6A1B9A"
)
plot_pie_fill <- scale_fill_manual(values = pie_colors)

element_colors <- c(
   "depletion"             = "#F40000"
  ,"excitability"          = "#FFAB00"
  ,"receptor-associated"   = "#D6EE00"
  ,"ionotropic receptor"   = "#43FF1C"
  ,"metabotropic receptor" = "#18FFFF"
  ,"signaling"             = "#0091EA"
  ,"g-protein"             = "#0033ff"
  ,"synthesis"             = "#AA00FF"
  ,"vesicle"               = "#FF00AA"
  #------- is_neuroexclusive --------
  ,"TRUE"                  = "#00BFC4"
  ,"FALSE"                 = "#F8766D"
)
# Color and size scales for neurotransmission functions
plot_scales <- list(
   scale_fill_manual(values = element_colors)
  ,scale_color_manual(values = element_colors %>% darken(0.25))
  ,scale_radius(range = c(1.75, 5.00), guide = FALSE)
)
```

```
systems <- names(pie_colors)

edge_color <- rgb(0.7, 0.7, 0.7, alpha = 0.3)

past_fill  <- "#FFFFFF" # past nodes' fill color
past_color <- "#888888" # past nodes' border color

# Baking some aesthetic properties into the vertices data.frame
vertices %<>% mutate(
  shape      = ifelse(ne, "square filled", "circle filled"),
  color_node = ifelse(ne, "#000000", element_colors[annotation] %>% darken(0.2)),
  color_pie  = ifelse(ne, "#000000", NA),
)

# Some recurrent ggplot aesthetics
edge_aes <- aes(x = x, y = y, group = group)
text_aes <- aes(x = x, y = y, label = string_name)
pie_aes  <- aes(x = x, y = y, group = string_id, r = size^(0.94) - 1.5)

# Fixing xy limits across all plots
xy_lim <- list(
  scale_x_continuous(limits = range(vertices[["x"]]) + c(-50, 50)),
  scale_y_continuous(limits = range(vertices[["y"]]) + c(-50, 50))
)

# Emptying theme defaults
plot_theme <- list(coord_equal(), theme_void())

# Allowing more space for multiple network plots
diff_theme <- list(
  coord_equal(),
  theme_void(),
  theme(
     plot.title         = element_text(size = 8, hjust = 0.5)
    ,legend.text        = element_text(size = 6)
    ,legend.title       = element_text(size = 8)
    ,legend.key.size    = unit( 1, "mm")
    ,legend.box.spacing = unit(-2, "mm")
    ,legend.box.margin  = unit(c(0, 2, 0, 0), "mm")
    ,plot.margin        = unit(c(0, 0, 0, 0), "mm")
  )
)

# Numeric vector named with clade names
roots <- vertices %>%
  arrange(-root) %>%
  distinct(root, clade_name) %$%
  set_names(root, clade_name)

upset_texts <- c(
   3   #ytitle
  ,2   #ytick
  ,1   #setsizetitle
  ,1.5 #setsizetick
  ,2   #setnames
  ,2.5 #barnums
)
```

Manuscript figure 1

```
####################
## Common elements
####################
plot_edges <- geom_path(
   data    = edges
  ,mapping = edge_aes
  ,color   = edge_color
  ,size    = 0.1
)

plot_text <- geom_text(
   data    = vertices
  ,mapping = text_aes
  ,size    = 1
  ,vjust   = 0
  ,nudge_y = 6
  ,alpha   = 0.5
```

```r
)

##############
## Figure 1A
##############
plot_pies <- geom_scatterpie(
   data    = vertices
  ,mapping = pie_aes
  ,cols    = systems
  ,color   = NA
)

fig1a <- ggplot() +
  plot_theme +
  plot_edges +
  plot_pies +
  plot_pie_fill +
  plot_text

##############
## Figure 1B
##############
plot_nodes <- geom_point(
   data    = vertices
  ,mapping = aes(x, y, fill = annotation, color = annotation, size = size)
  ,shape   = 21
  ,stroke  = 0.5
)

fig1b <- ggplot() +
  plot_theme +
  plot_edges +
  plot_nodes +
  plot_scales +
  plot_text

# Plotting and saving
fig1a + fig1b
```



Supplementary Figure 3: Unedited manuscript figure 1. The human neurotransmission network with nodes colored by neurotransmitter systems (left) and neurotransmission functions (right).

```r
ggsave("plots/fig1_raw.pdf", width = 14, height = 7, onefile = F, useDingbats = F)
```

Manuscript figure 2

This figure is produced externally by a program called ViaComplex. ViaComplex superimposes a heatmap over the network layout based on a node property. This property, in our case, is a node's neuroexclusivity. The following block handles data formatting related to ViaComplex.

```r
# Retrieving the largest connected component
subgraphs <- decompose.graph(g)
lcc_index <- which.max(sapply(subgraphs, vcount))
lcc       <- subgraphs[[lcc_index]]

# Writing network data to viacomplex's custom format (similar to pajek)
# xy_hack adds some extra margin to the plot
xy_hack <- data.frame(
   name                          = c("top", "bot")
  ,x                             = range(V(lcc)$x) + c(-75, 75)
  ,y                             = range(V(lcc)$y) + c(-75, 75)
  ,pathway_neuroexclusivity      = 0
  ,expression_neuroexclusivity   = 0
  ,stringsAsFactors              = F
)

pajek_nodes <- lcc %>%
  igraph::as_data_frame("vertices") %>%
  bind_rows(xy_hack) %>%
  mutate(id = row_number(), y = -y)

pajek_edges <- igraph::as_data_frame(lcc, "edges")

# Creating the network_viacomplex.net file and sequentially populating it
write("*edges", "network_viacomplex.net")
write_tsv(
   x            = pajek_edges
  ,path         = "network_viacomplex.net"
  ,append       = T
  ,col_names    = F
  ,quote_escape = F
)
write("*nodes", "network_viacomplex.net", append = T)
write_tsv(
   x            = pajek_nodes %>% select(name, x, y)
  ,path         = "network_viacomplex.net"
  ,append       = T
  ,col_names    = F
  ,quote_escape = F
)

write_tsv(
   x    = pajek_nodes %>% select(id, name, pathway_neuroexclusivity)
  ,path = "network_viacomplex_pathway.dat"
)
write_tsv(
   x    = pajek_nodes %>% select(id, name, expression_neuroexclusivity)
  ,path = "network_viacomplex_expression.dat"
)
```

Manuscript figure 3

The process for generating Figures 3 and 4 is roughly the same. It consists of finding what nodes have numeric roots in a given range. In our analysis, the largest root is numbered 37 and represents the oldest common ancestor to humans in the cladogram (the Human-Metamonada LCA, as seen in previous sections). Root number 1 is represented by *Homo sapiens* itself.

The nodes we need to draw are either `current_nodes` (roots in a specified numeric range), or `past_nodes` (roots > such specified range). The edges we need to draw are all edges between both sets of nodes.

Manuscript Figure 3A

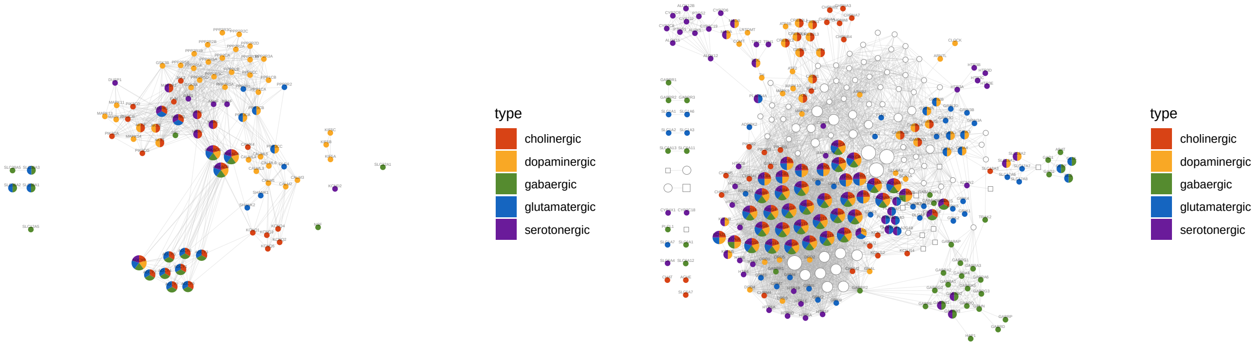
```r
# Finding which genes should be drawn
current_genes <- vertices %>% filter(root == 37)

# Finding which edges should be drawn
partial_ids   <- current_genes %>% pull(string_id)
```

```r
which_edges   <- apply(string_edgelist, 1, function(r) all(r %in% partial_ids))
partial_edges <- edges[which_edges,]

plot_edges <- geom_path(
    data     = partial_edges
  ,mapping = edge_aes
  ,color   = edge_color
  ,size    = 0.1
)

plot_text <- geom_text(
    data     = current_genes
  ,mapping = text_aes
  ,size    = 1
  ,vjust   = 0
  ,nudge_y = 1.75
  ,alpha   = 0.5
)

plot_current_pies <- geom_scatterpie(
    data     = current_genes
  ,mapping = pie_aes
  ,cols    = systems
  ,color   = NA
)

# Assembling
fig3a <- ggplot() +
  plot_edges +
  plot_scales +
  xy_lim +
  plot_current_pies +
  plot_pie_fill +
  plot_text +
  plot_theme
```

Manuscript Figure 3B

For Figure 3B, we want to see what nodes have numeric roots < 37 (Human-Metamonada LCA) and >= 26 (Human-Cnidaria LCA).

```r
# Finding which genes should be drawn
current_genes <- vertices %>% filter(root < 37 & root >= 26)
past_genes    <- vertices %>% filter(root == 37)

# Finding which edges should be drawn
partial_ids   <- c(current_genes[["string_id"]], past_genes[["string_id"]])
which_edges   <- apply(string_edgelist, 1, function(r) all(r %in% partial_ids))
partial_edges <- edges[which_edges,]

plot_edges <- geom_path(
    data     = partial_edges
  ,mapping = edge_aes
  ,color   = edge_color
  ,size    = 0.1
)

plot_past <- geom_point(
    data     = past_genes
  ,mapping = aes(x, y, size = size)
  ,fill    = past_fill
  ,color   = past_color
  ,shape   = past_genes$shape
  ,stroke  = 0.25
)

plot_text <- geom_text(
    data     = current_genes
  ,mapping = text_aes
  ,size    = 1
  ,vjust   = 0
  ,nudge_y = 1.75
  ,alpha   = 0.5
)

plot_current_pies <- geom_scatterpie(
    data     = current_genes
  ,mapping = pie_aes
  ,cols    = systems
  ,color    = NA
)

# Assembling
fig3b <- ggplot() +
  plot_edges +
  plot_past +
  plot_scales +
  xy_lim +
  plot_current_pies +
  plot_pie_fill +
  plot_text +
  plot_theme

# Plotting and saving
fig3a + fig3b
```



Supplementary Figure 4: Unedited manuscript figure 3. The human neurotransmission network with nodes colored by neurotransmitter systems and neurotransmission functions.

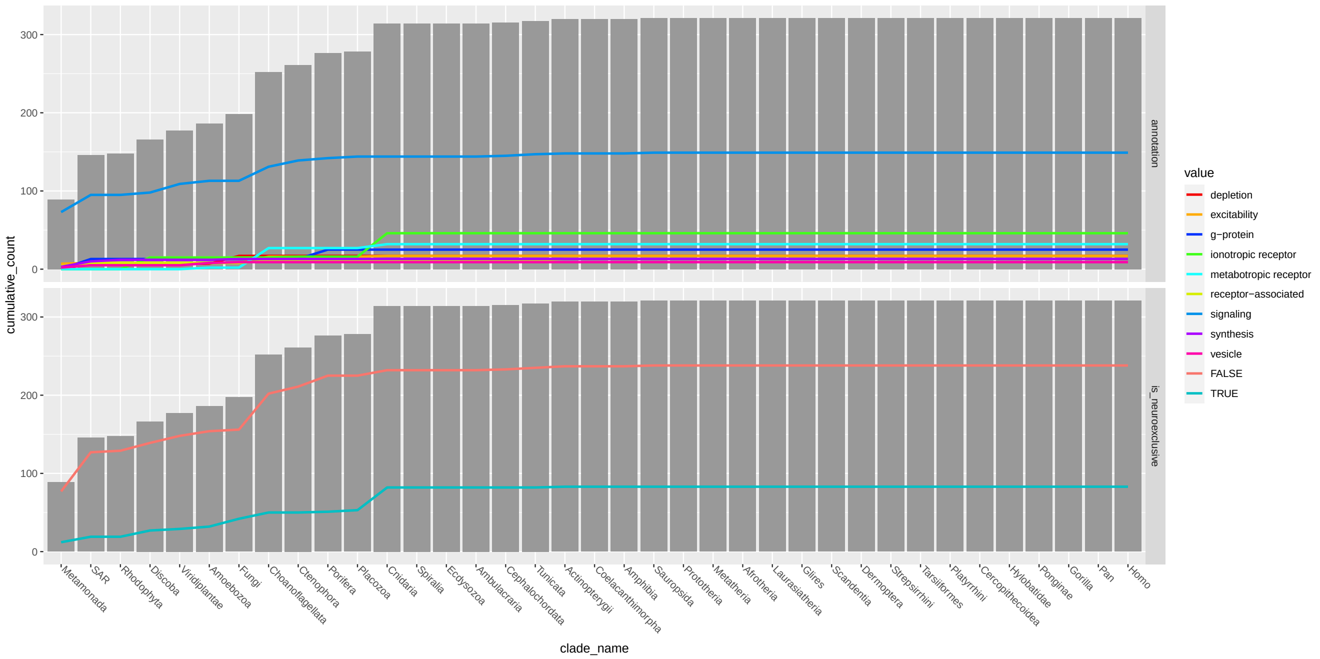
```
ggsave("plots/fig3_raw.pdf", width = 14, height = 7, onefile = F, useDingbats = F)
```

Additionally, we cumulatively count nodes by their categories (function and neuroexclusivity) and inferred root:

```
cumulative_emergence <- vertices %>%
  select(root, annotation, is_neuroexclusive = ne) %>%
  # Adding clade info
  right_join(clade_names) %>%
  # Pivoting from wide to long
  pivot_longer(annotation:is_neuroexclusive, values_ptypes = list(value = "character")) %>%
  # Counting nodes by category (name) for each root
  count(root, clade_name, name, value) %>%
  # Making absent counts explicit
  group_by(name) %>%
  complete(nesting(root, clade_name), name, value, fill = list(n = 0)) %>%
  # No reason to include NA observations in cumulative sum
  na.omit %>%
  # Cumulative sum node count at each root
  group_by(name, value) %>%
  mutate(cumulative_count = order_by(-root, cumsum(n)))
```

Plotting such cumulative counts:

```
cumulative_emergence %<>% ungroup %>%
  # Creating ordered factors for plotting
  mutate(
     clade_name = fct_reorder(clade_name, -root)
    ,value      = fct_reorder(value, name)
  )

ggplot(cumulative_emergence) +
  #----- Barplot ------
  geom_bar(
     mapping     = aes(clade_name, cumulative_count, group = value)
    ,stat        = "sum"
    ,fill        = "#999999"
    ,show.legend = F
  ) +
  #----- Lines ------
  geom_line(
     mapping = aes(clade_name, cumulative_count, group = value, color = value)
    ,size    = 1
  ) +
  #----- Styling ------
  scale_color_manual(values = element_colors) +
  facet_grid(name ~ .) +
  theme(axis.text.x = element_text(angle = -45, vjust = 0, hjust = 0))
```

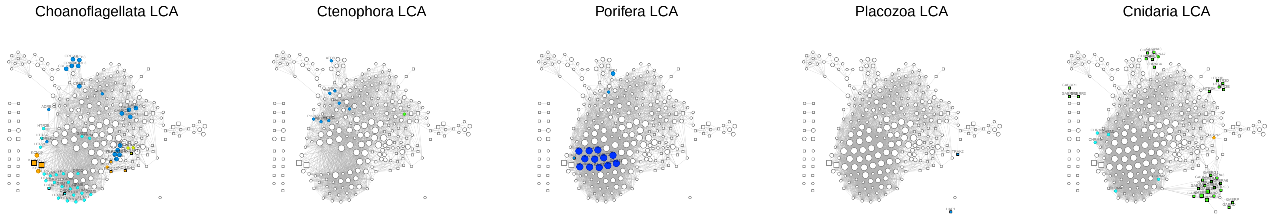


Supplementary Figure 5: Cumulative node counts by categories at each root.

Manuscript figure 4

Visualizing nodes with roots <= 30 (Human-Porifera LCA) and >= 26 (Human-Cnidaria LCA) at every distinct root.

```
plot_size ← scale_radius(range = c(0.5, 1.5), guide = FALSE)

fig4 ← roots[roots ⩾ 26 & roots ⩽ 30] %>%
  imap(~ {
  # Finding which genes should be drawn
  current_genes ← vertices %>% filter(root == .x)
  past_genes    ← vertices %>% filter(root  > .x)

  # Finding which edges should be drawn
  partial_ids   ← c(current_genes[["string_id"]], past_genes[["string_id"]])
  which_edges   ← apply(string_edgelist, 1, function(r) all(r %in% partial_ids))
  partial_edges ← edges[which_edges,]

  plot_edges ← geom_path(
     data     = partial_edges
    ,mapping = edge_aes
    ,color   = edge_color
    ,size    = 0.1
  )

  plot_past <- geom_point(
     data     = past_genes
    ,mapping = aes(x, y, size = size)
    ,fill    = past_fill
    ,color   = past_color
    ,shape   = past_genes$shape
    ,stroke  = 0.25
  )

  plot_text <- geom_text(
     data     = current_genes
    ,mapping = text_aes
    ,size    = 0.8
    ,vjust   = -0.5
    ,nudge_y = 1
    ,alpha   = 0.5
  )

  plot_current_nodes <- geom_point(
     data     = current_genes
    ,mapping = aes(x, y, fill = annotation, size = size)
    ,color   = current_genes$color_node
    ,shape   = current_genes$shape
    ,stroke  = 0.25
  )

  remove_legend <- guides(fill = "none", colour = "none")

  # Assembling
  ggplot() +
    ggtitle(paste(.y, "LCA")) +
    diff_theme +
    xy_lim +
    plot_edges +
    plot_past +
    plot_current_nodes +
    plot_scales +
    plot_size +
    plot_text +
    remove_legend
})

fig4 <- invoke(grid.arrange, fig4, ncol = 5)
```



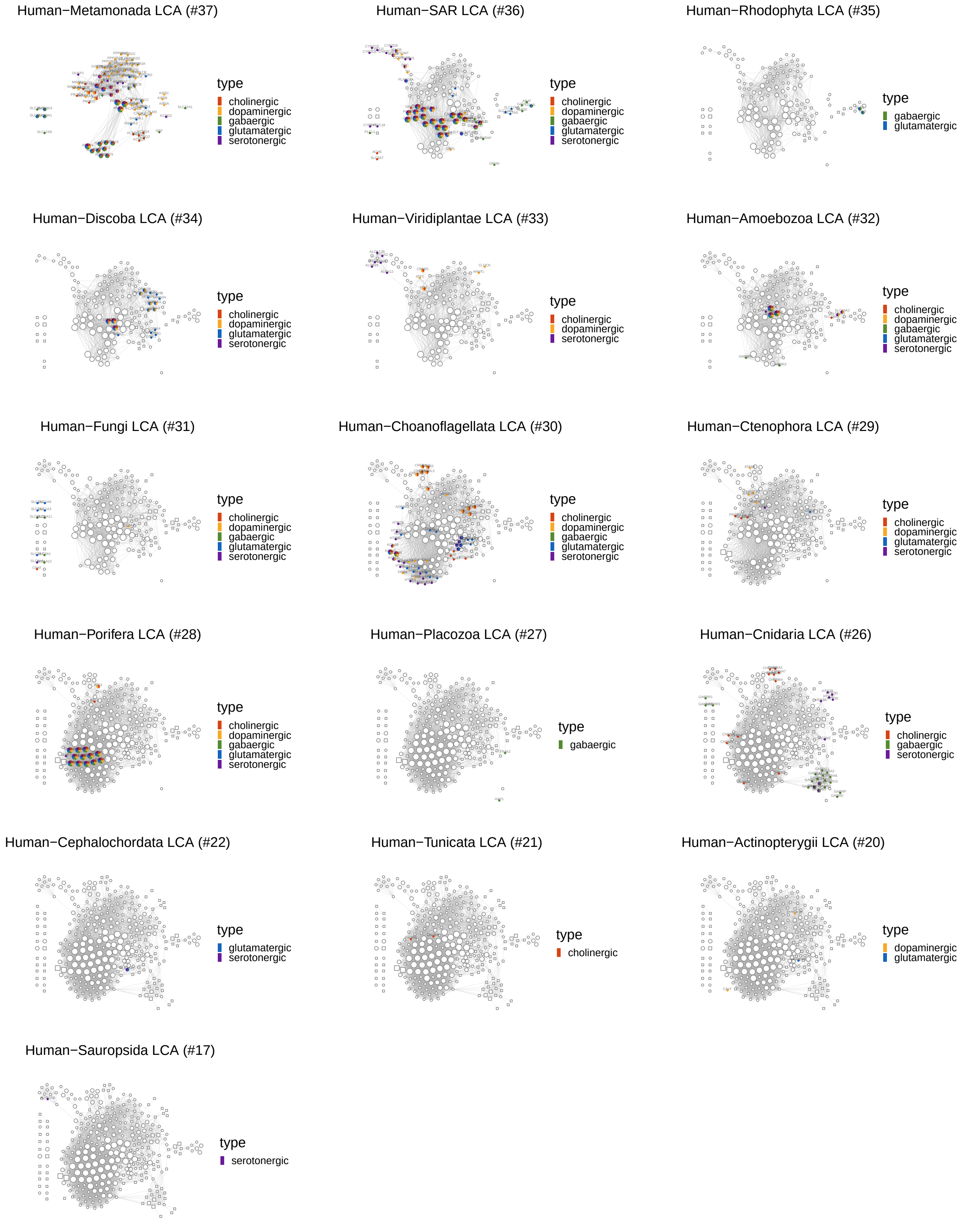
Supplementary Figure 6: Unedited manuscript figure 4. The human neurotransmission network with nodes rooted between roots 30 (Human-Choanoflagellata LCA) and 26 (Human-Cnidaria LCA).

```
ggsave(
  "plots/fig4_raw.pdf"
  ,plot        = fig4
  ,width       = 9*0.9
  ,height      = 5*0.9
  ,onefile     = F
  ,useDingbats = F
)
```

Supplementary network figures

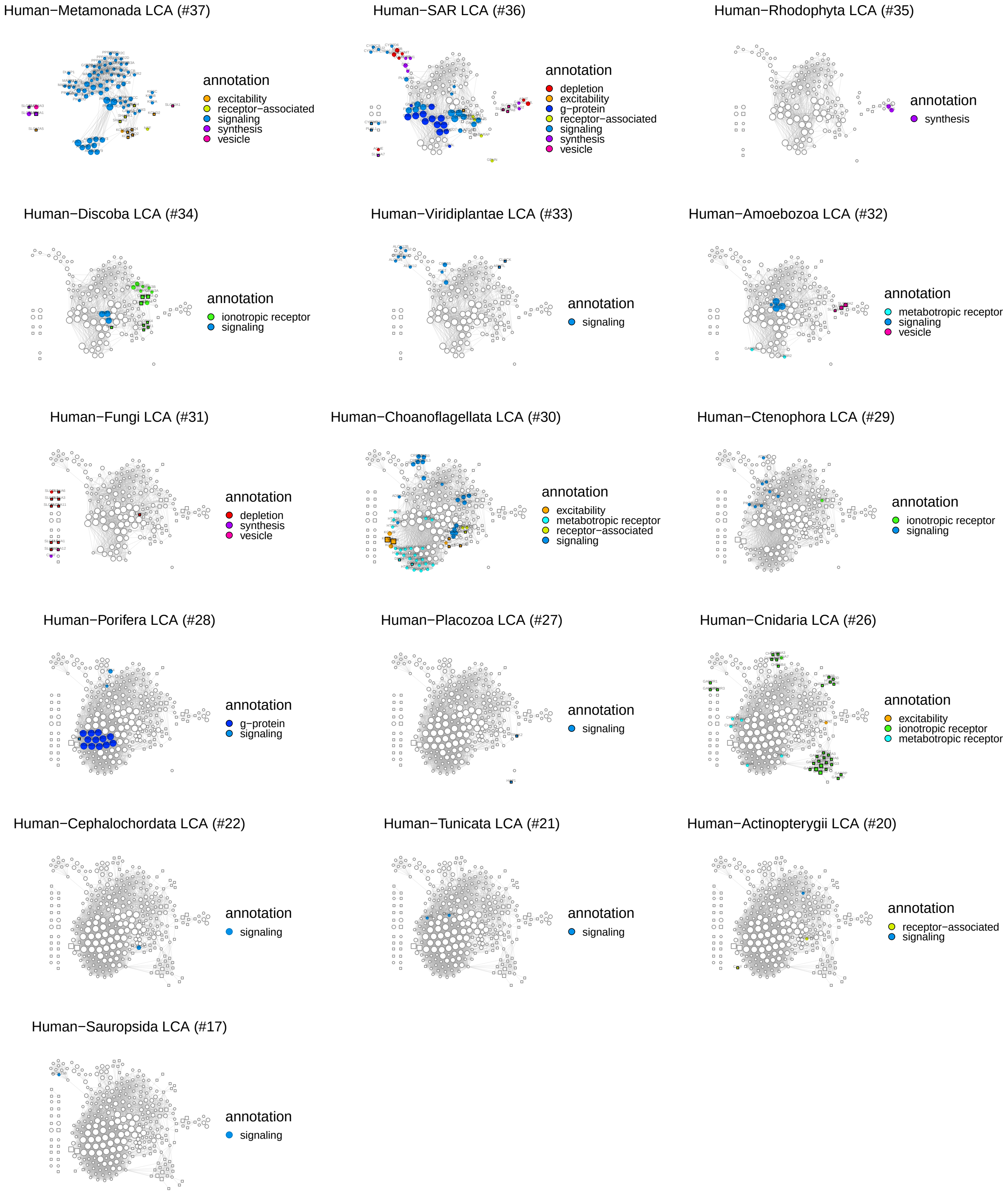
The following supplementary figures help us see what nodes have been rooted at each LCA. Nodes rooted at previous LCAs are colored white.

```r
system_plots   <- list()
function_plots <- list()

iwalk(roots, ~ {
  # Finding which genes should be drawn
  current_genes <- vertices %>% filter(root == .x)
  past_genes    <- vertices %>% filter(root  > .x)

  # Finding which edges should be drawn
  partial_ids   <- c(current_genes[["string_id"]], past_genes[["string_id"]])
  which_edges   <- apply(string_edgelist, 1, function(r) all(r %in% partial_ids))
  partial_edges <- edges[which_edges,]

  # Common elements -----------
  plot_edges <- geom_path(
     data    = partial_edges
    ,mapping = edge_aes
    ,color   = edge_color
    ,size    = 0.1
  )

  plot_past <- geom_point(
     data    = past_genes
    ,mapping = aes(x, y, size = size)
    ,fill    = past_fill
    ,color   = past_color
    ,shape   = past_genes$shape
    ,stroke  = 0.25
  )

  plot_text <- geom_text(data = current_genes, text_aes, size = 0.75, nudge_y = 4, alpha = 0.5)

  base <- ggplot() +
    ggtitle(paste0("Human-", .y, " LCA (#", .x, ")")) +
    diff_theme +
    xy_lim +
    plot_edges +
    plot_past +
    plot_size

  # Nodes colored by system -----------
  plot_current_pies <- geom_scatterpie(data = current_genes, pie_aes, cols = systems, color = NA)

  system_plots[[as.character(.x)]] <<- base +
    plot_current_pies +
    plot_pie_fill +
    plot_text

  # Nodes colored by function --------
  plot_current_nodes <- geom_point(
     data    = current_genes
    ,mapping = aes(x, y, fill = annotation, size = size)
    ,color   = current_genes$color_node
    ,shape   = current_genes$shape
    ,stroke  = 0.25
  )

  function_plots[[as.character(.x)]] <<- base +
    plot_current_nodes +
    plot_scales +
    plot_size +
    plot_text +
    guides(fill = guide_legend(override.aes = list(shape = 21))) # legend hack
})

# Printing and saving
arrangeGrob(grobs = system_plots, ncol = 3) %>% grid::grid.draw()
```



Supplementary Figure 7: The human neurotransmission network with nodes rooted at each human LCA.

```r
arrangeGrob(grobs = function_plots, ncol = 3) %>% grid::grid.draw()
```

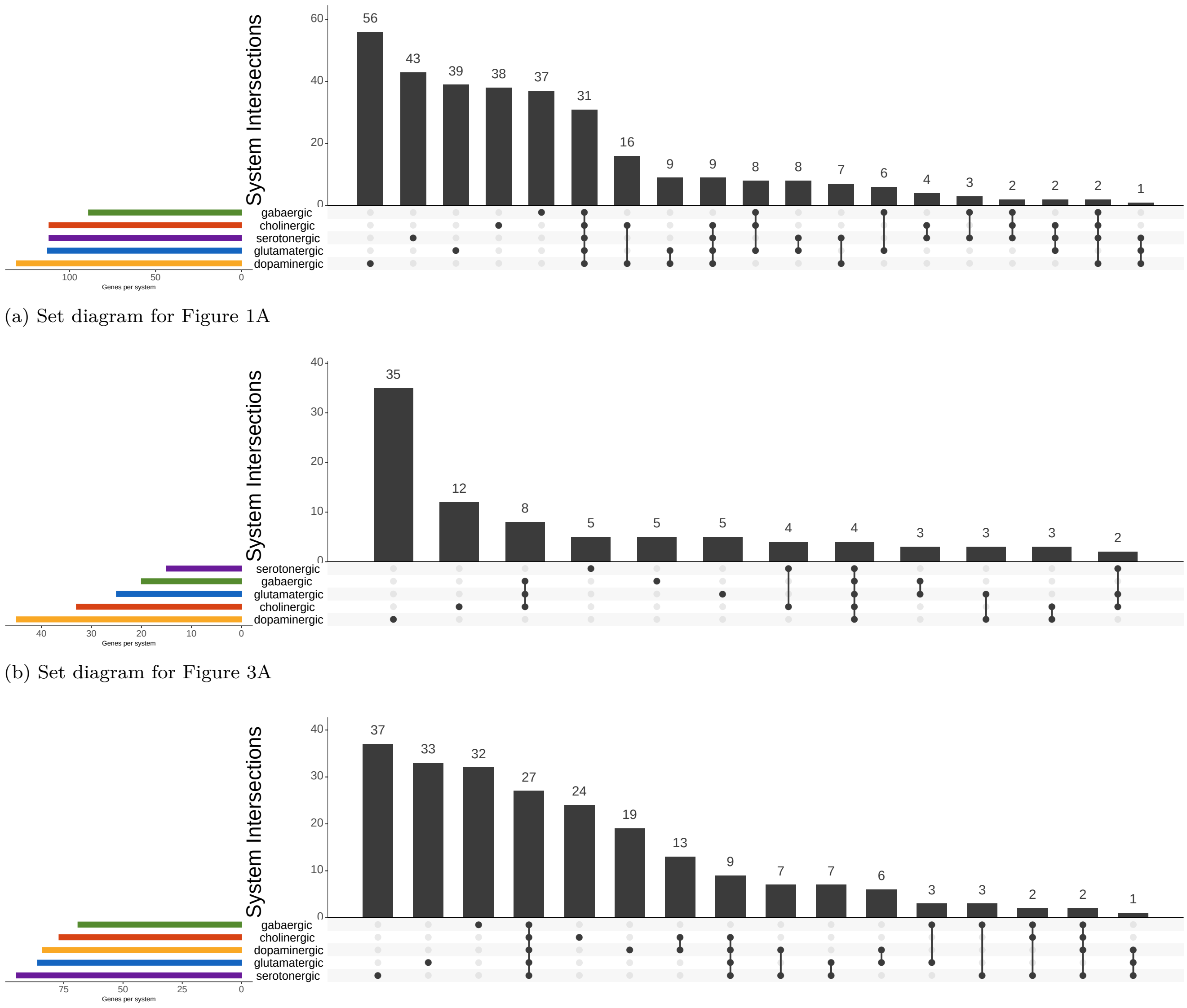


Supplementary Figure 8: The human neurotransmission network with nodes rooted at each human LCA.

Manuscript set diagrams

Given the dificulties of joining ggplot and base plots, the set diagrams have to be plotted by themselves:

```r
# We have to manually find the correct order of colors
# Because UpSetR does not understand named vectors
get_colors <- function(df) {
  ordered_systems <- df %>%
    select(systems) %>%
    colSums %>%
    extract(. > 0) %>%
    extract(order(., names(.), decreasing = T))

  pie_colors[names(ordered_systems)]
}

# Figure 1A set diagram
upset(
   select(vertices, systems)
  ,mb.ratio        = c(0.7, 0.3)
  ,order.by        = "freq"
  ,mainbar.y.label = "System Intersections"
  ,sets.x.label    = "Genes per system"
  ,text.scale      = upset_texts
  ,point.size      = 3.5
  ,line.size       = 1
  ,sets.bar.color  = get_colors(vertices)
)
dev.print(pdf, "plots/fig1a_set_raw.pdf", width = 18, height = 10, onefile = F, useDingbats = F)

# Figure 3A set diagram
fig3a_set <- vertices %>% filter(root == 37) %>% select(systems)
upset(
   fig3a_set
  ,mb.ratio        = c(0.7, 0.3)
  ,order.by        = "freq"
  ,mainbar.y.label = "System Intersections"
  ,sets.x.label    = "Genes per system"
  ,text.scale      = upset_texts
  ,point.size      = 3.5
  ,line.size       = 1
  ,sets.bar.color  = get_colors(fig3a_set)
)
dev.print(pdf, "plots/fig3a_set_raw.pdf", width = 16, height = 8, onefile = F, useDingbats = F)

# Figure 3B set diagram
fig3b_set <- vertices %>% filter(root < 37 & root >= 26) %>% select(systems)
upset(
   fig3b_set
  ,mb.ratio        = c(0.7, 0.3)
  ,order.by        = "freq"
  ,mainbar.y.label = "System Intersections"
  ,sets.x.label    = "Genes per system"
  ,text.scale      = upset_texts
  ,point.size      = 3.5
  ,line.size       = 1
  ,sets.bar.color  = get_colors(fig3b_set)
)
dev.print(pdf, "plots/fig3b_set_raw.pdf", width = 16, height = 8, onefile = F, useDingbats = F)
```



(a) Set diagram for Figure 1A

(b) Set diagram for Figure 3A

(c) Set diagram for Figure 3B

Supplementary Figure 9: Set diagrams

Abundance

Abundances are the number of proteins in orthogroups of a species. These abundances are aggregated and averaged according to the function of their corresponding orthogroups. The orthogroup function is considered to be the function of its human proteins. Loading initial resources:

```r
# Data manipulation
library(tidyverse)
library(magrittr)

# Utils
library(neurotransmissionevolution)

# Packaged data
data(
   cogs
  ,gene_ids
  ,gene_cogs
  ,string_eukaryotes
  ,package = "neurotransmissionevolution"
)

# Fresh analysis data
cog_roots                   ← read_tsv("geneplast_roots.tsv",             col_types = "ci")
clade_names                 ← read_tsv("geneplast_clade_names.tsv",       col_types = "ic")
clade_taxids                ← read_tsv("geneplast_clade_taxids.tsv",      col_types = "ici")

# Collapsing similar functions
gene_annotation ← read_tsv("../data/gene_annotation.tsv", col_types = "cc") %>%
  mutate(annotation = case_when(
     grepl("clearance",   annotation) ~ "depletion"
    ,grepl("degradation", annotation) ~ "depletion"
    ,grepl("transport",   annotation) ~ "synthesis"
    ,TRUE ~ annotation
  ))
```

We start by setting up reusable data frames with useful metrics.

```
# If a gene has more than 1 COG, select the oldest one.
# This is unusual, but can happen in cases of gene fusion, for instance.
gene_cogs %<>%
  inner_join(cog_roots) %>%
  group_by(string_id) %>%
  filter(root == max(root))

# The function of a COG is the function of its proteins
cog_annotation ← gene_ids %>%
  inner_join(gene_cogs) %>%
  inner_join(gene_annotation) %>%
  distinct(cog_id, annotation)

# Number of proteins in a COG in every species
cog_abundance_by_taxid ← cogs %>%
  filter(cog_id %in% gene_cogs[["cog_id"]]) %>%
  count(taxid, cog_id,  name = "abundance") %>%
  left_join(cog_annotation)

# Mapping species to clade info
ordered_species ← string_eukaryotes %>%
  select(taxid, ncbi_name) %>%
  left_join(clade_taxids) %>%
  left_join(clade_names, by = c("lca" = "root")) %>%
  mutate(
     ncbi_name  = fct_reorder(ncbi_name, -taxid_order)
    ,clade_name = fct_reorder(clade_name, -taxid_order)
  )

# Plotting colors
annotation_colors ← c(
   "depletion"             = "#F40000"
  ,"excitability"          = "#FFAB00"
  ,"receptor-associated"   = "#D6EE00"
  ,"ionotropic receptor"   = "#43FF1C"
  ,"metabotropic receptor" = "#18FFFF"
  ,"signaling"             = "#0091EA"
  ,"g-protein"             = "#0033ff"
  ,"synthesis"             = "#AA00FF"
  ,"vesicle"               = "#FF00AA"
)
```

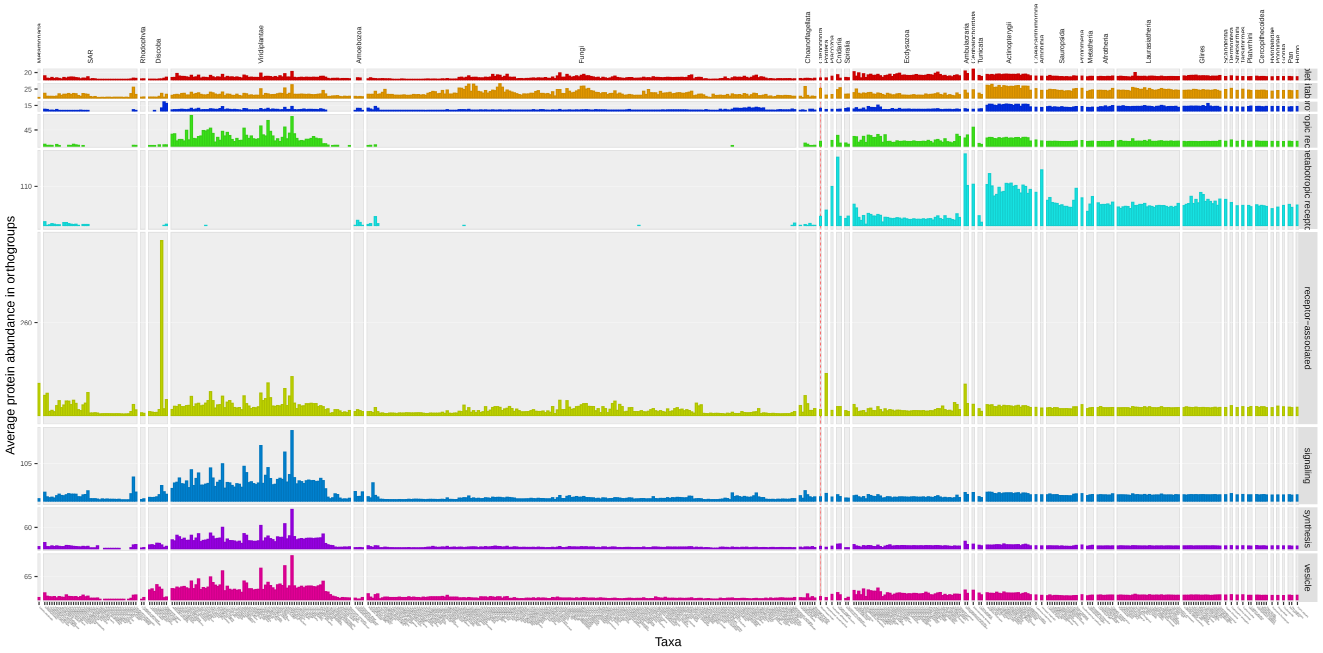
The average orthogroup abundances are finally computed for each species according to the function of orthogroups.

```
avg_abundance_by_function ← cog_abundance_by_taxid %>%
  group_by(taxid, annotation) %>%
  summarise(avg_abundance = mean(abundance)) %>%
  # Adding species and clade info
  left_join(ordered_species)
```

Plotting:

```
# This vertical line indicates the first metazoan (Mnemiopsis leidyi / Ctenophora)
metazoa_line ← geom_vline(
   xintercept = "Mnemiopsis leidyi"
  ,color      = "#FF0000"
```

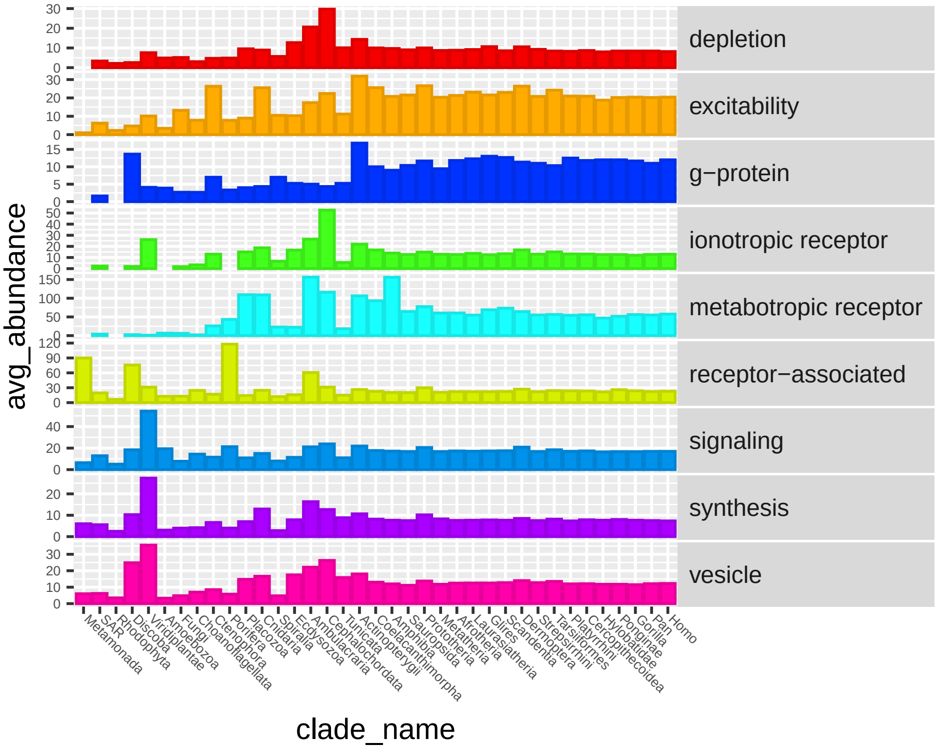
```r
  ,linetype   = "11"
  ,alpha      = 1
  ,size       = 0.25
)
tick_function <- function(x) {
  seq(x[2], 0, length.out = 3) %>% head(-1) %>% tail(-1) %>% { ceiling(./5)*5 }
}

# Capping abundance values based on metazoan mean
capped_abundance_by_function <- avg_abundance_by_function %>%
  # mutate(capped_abundance = ifelse(abundance >= 100, 100, abundance)) %>%
  group_by(annotation) %>%
  mutate(
     max_abundance = avg_abundance[lca <= 29] %>% { mean(.) + 3*sd(.) }
    ,avg_abundance = ifelse(avg_abundance >= max_abundance, pmin(max_abundance, 100), pmin(avg_abundance, 100))
  )

# Plotting
abundance_plot <- ggplot(capped_abundance_by_function) +
  # Geoms  ---------------
  metazoa_line +
  geom_bar(
     aes(x = ncbi_name, y = avg_abundance, fill = annotation, color = after_scale(darken(fill, 0.1)))
    ,stat = "identity"
  ) +
  # Labels  --------------
  xlab("Taxa") +
  ylab("Average protein abundance in orthogroups") +
  # Scales ---------------
  scale_y_continuous(breaks = tick_function, minor_breaks = NULL) +
  scale_fill_manual(values = annotation_colors %>% darken(0.1)) +
  # Styling --------------
  facet_grid(annotation ~ clade_name, scales = "free", space = "free") +
  theme(
     panel.spacing      = unit(2.5, "pt")
    ,strip.background.x = element_blank()
    ,strip.background.y = element_rect(fill="#E0E0E0")
    ,panel.grid.major.x = element_blank()
    ,panel.grid.major.y = element_line(colour = "#F5F5F5", size = 0.25)
    ,panel.background   = element_rect(fill = '#EEEEEE', colour = '#E0E0E0')
    ,strip.text.x       = element_text(size = 6, angle = 90, hjust = 0, vjust = 0.5)
    ,strip.text.y       = element_text(size = 8, vjust = 0.5)
    ,axis.text.x        = element_text(size = 2, angle = -45, vjust = 0, hjust = 0)
    ,axis.text.y        = element_text(size = 6)
    ,legend.position    = "none"
  )
ggsave("plots/fig5_raw.pdf", abundance_plot, width = 16, height = 6)

# Uncapped abundances for supplementary text
abundance_plot %+% avg_abundance_by_function
```



Supplementary Figure 10: Abundance values by species. Species are ordered like in Supplementary Figure 1.

Species-specific average abundances, now averaged by clades:

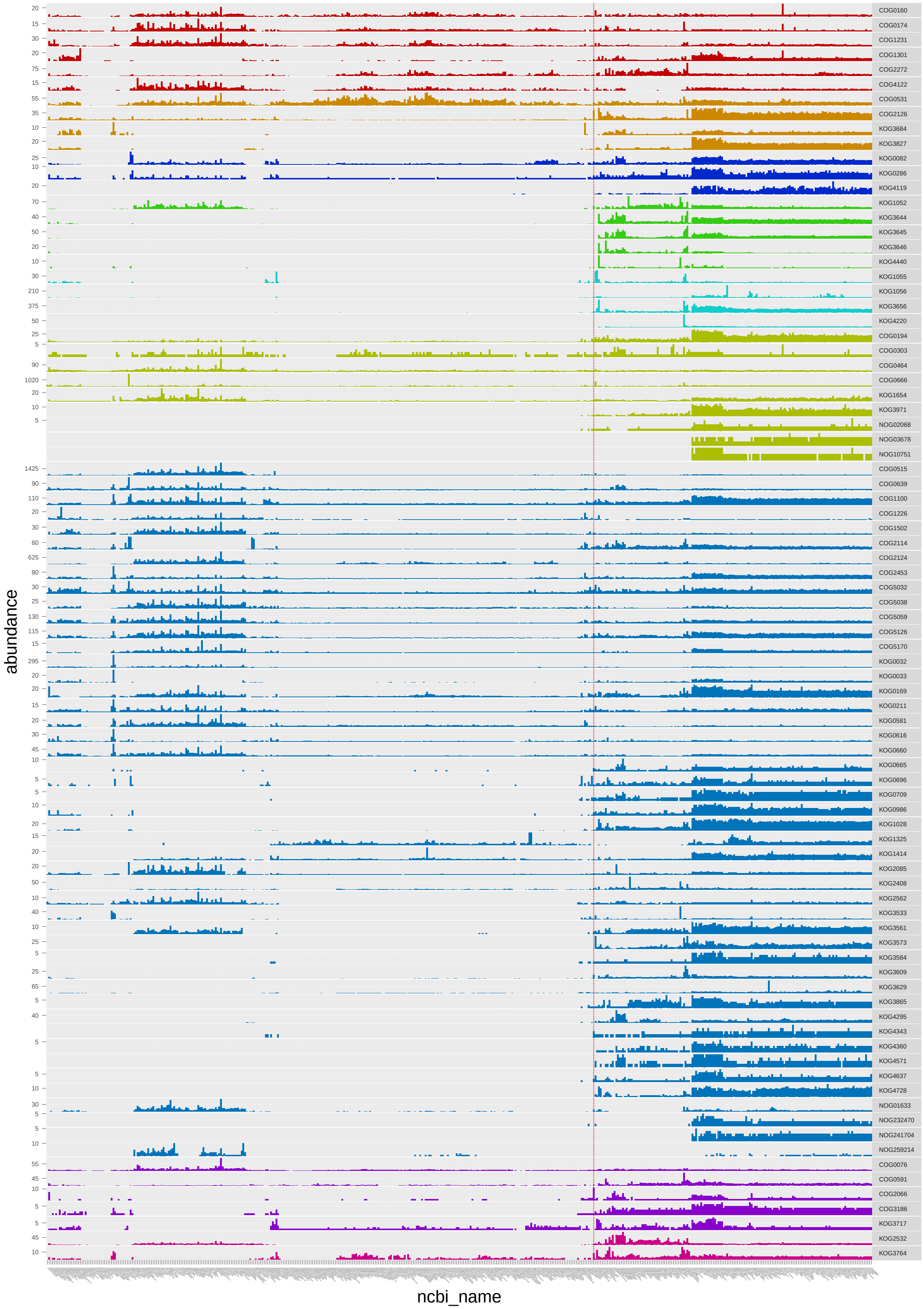
```r
ggplot(avg_abundance_by_function) +
  geom_bar(
    aes(x = clade_name, y = avg_abundance, fill = annotation, color = after_scale(darken(fill, 0.1)))
    ,stat = "summary"
    ,fun  = "mean"
  ) +
  scale_fill_manual(values = annotation_colors, guide = "none") +
  facet_grid(annotation ~ ., scales = "free") +
  theme(
     panel.spacing       = unit(1, "pt")
    ,strip.text.y        = element_text(angle = 0, hjust = 0)
    ,axis.text.x         = element_text(size = 5, angle = -45, vjust = 0, hjust = 0)
    ,axis.text.y         = element_text(size = 5)
  )
```



Supplementary Figure 11: Abundances averaged by clades.

Plain protein abundance in single orthogroups

```r
# Merging mixed COG functions
cog_annotation_collapsed <- cog_annotation %>%
  group_by(cog_id) %>%
  summarise(annotation = paste(annotation, collapse = "/"))

# Adding colors for mixed COGs
annotation_colors %<>% c(
   "vesicle/synthesis"      = "#808080"
  ,"depletion/vesicle"      = "#808080"
  ,"signaling/excitability" = "#808080"
)

# Simply joining cog annotation and sizes
protein_count_by_cog <- cog_abundance_by_taxid %>%
  inner_join(cog_annotation_collapsed) %>%
  left_join(ordered_species) %>%
  arrange(annotation) %>%
  mutate(cog_id = fct_inorder(cog_id))

ggplot(protein_count_by_cog) +
  metazoa_line +
  geom_bar(aes(x = ncbi_name, y = abundance, fill = annotation), stat = "identity") +
  scale_fill_manual(values = annotation_colors %>% darken(0.2), guide = "none") +
  scale_y_continuous(breaks = tick_function, minor_breaks = NULL) +
  facet_grid(cog_id ~ ., scales = "free_y") +
  theme(
     panel.spacing      = unit(0.5, "pt")
    ,panel.grid.major.x = element_blank()
    ,panel.grid.major.y = element_line(size = 0.1, linetype = "dashed")
    ,strip.text.y       = element_text(size = 4, angle = 0, hjust = 0)
    ,axis.text.x        = element_text(size = 1.25, angle = -45, vjust = 0, hjust = 0)
    ,axis.text.y        = element_text(size = 4)
    ,axis.ticks         = element_line(size = 0.1)
  )
```



Supplementary Figure 12: Number of proteins in each neurotransmission COG, for every species.

```
#
```