Ionotropic receptors as the driving force behind human synapse establishment

Supplementary Material

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

Abstract

Model uncertainty and limited data are fundamental challenges to robust management of human intervention in a natural system. These challenges are acutely highlighted by concerns that many ecological systems may contain tipping points, such as Allee population sizes. Before a collapse, we do not know where the tipping points lie, if they exist at all. Hence, we know neither a complete model of the system dynamics nor do we have access to data in some large region of state-space where such a tipping point might exist.

Contents

Project structure

This is the title page

Preprocessing

This topic refers mainly to data wrangling done before the actual analysis with the intent of making it simpler.

Eukaryota species tree

We opted to use the TimeTree database in order to obtain an standardized Eukaryota species tree. However, some species were not present in it, so we devised a way to fill them in based on NCBI Taxonomy data.

NCBI Taxonomy tree

First we preprocess NCBI Taxonomy data to leave only STRING eukaryotes, thus making the task easier.

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Table 2: Links outdated taxon IDs to corresponding new ones.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

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.

```
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.

```
# 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).

```
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.

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

# plot(ncbi_tree %>% ape::ladderize(), type="cladogram")

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")

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

```
## <U+2714> Setting active project to 'C:/R/neuro'
## <U+2714> Saving 'string_eukaryotes' to 'data/string_eukaryotes.rda'
```

Duplicated Genera

Some species from different kingdoms may share the same genus name. These genera must be noted down because one of the ways we fill in missing species is by looking at genera names.

Loading data

See Table 3 and Table 4.

```
taxid_rank <- read_delim(
  "download/taxdump/nodes.dmp",
  skip = 1,
  delim = "|",
  trim_ws = TRUE,
  col_names = c("taxid","rank"),
  col_types = "c-c"
)

ncbi_taxon_names <- read_delim(
  "download/taxdump/names.dmp",
  delim = "|",
  trim_ws = TRUE,
  col_names = c("taxid","ncbi_name","type"),
  col_types = "cc-c"
)
```

Finding duplicated genera

```r
# keeping genera nodes
taxid_rank %<>% filter(rank == "genus")

# keeping scientific names
ncbi_taxon_names %<>%
  filter(type == "scientific name") %>%
  select(taxid, ncbi_name) %>%
  inner_join(taxid_rank)

# extracting and saving duplicated values
duplicated_genera <- ncbi_taxon_names %>%
  pull(ncbi_name) %>%
  extract(duplicated(.)) %>%
  write("duplicated_genera.txt")
```

Hybrid tree

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

Downloading data

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"
)
```

Loading data

```r
# loading species names and taxon ids
data(string_eukaryotes, package = "neurotransmissionevolution")

# 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.

```r
# 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.

```r
# converting ncbi phylo to igraph
graph_ncbi <- read.tree("ncbi_tree.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)

# 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")
```

```r
# 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
# moving ctenophora before porifera
mnemiopsis_taxid <- species_dictionary %>% filter(ncbi_name == "Mnemiopsis leidyi") %>% pull(taxid)
amphimedon_taxid <- species_dictionary %>% filter(ncbi_name == "Amphimedon queenslandica") %>% pull(taxid)

# 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")
```

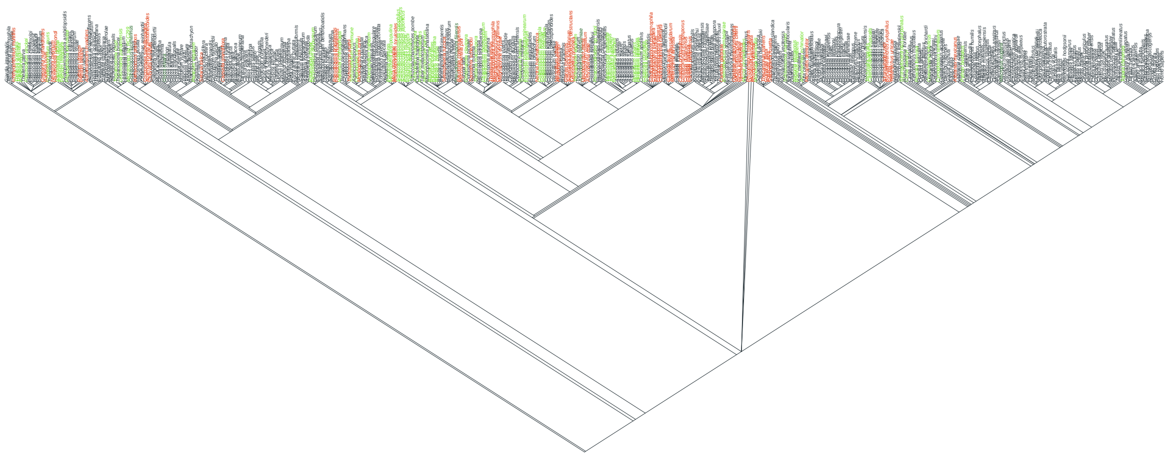

Figure 1: Complete 476 eukaryotes tree. Green species have been filled in by a genus proxy in TimeTree. Red species have been filled in by looking at NCBI Taxonomy.

Gene selection and annotation

The anchoring point for this study is basic annotation about genes and the pathways in which they participate. This section describes the process of structuring such data. In the end we will have a table to which all kinds of additional data will be left joined into.

Neurotransmitter systems annotation

We start by querying the KEGG api for the pathways of interest. Resulting data is then 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

```
pathways <- tribble(
  ~pathway_id,      ~pathway_name,
  "path:hsa04724",  "glutamatergic",
  "path:hsa04725",  "cholinergic",
  "path:hsa04726",  "serotonergic",
  "path:hsa04727",  "gabaergic",
  "path:hsa04728",  "dopaminergic",
  "path:hsa04721",  "vesicle"
)

# 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) %>%
```

```r
  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)
  ) %>%
  # filling 1's in all systems for synaptic vesicle genes
  # mutate_at(pathways[["pathway_name"]], ~ ifelse(vesicle == 1, 1L, .)) %>%
  mutate(system_count = rowSums(select(., -entrez_id, -vesicle)))

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

```
## <U+2714> Setting active project to 'C:/R/neuro'
## <U+2714> Saving 'gene_pathways' to 'data/gene_pathways.rda'
```

tail(gene_pathways)

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

```
# 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 perform it manually

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

entrez_id	ensembl_id	string_id	string_name	entrez_name
9296	ENSG00000128524	NA	NA	ATP6V1F
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

```
complete_info <- tribble(
  ~entrez_id,    ~ensembl_id,        ~string_id,              ~string_name,     ~entrez_name,
  "9296",       "ENSG00000128524",  "9606.ENSP00000417378",  "ATP6V1F",        "ATP6V1F",
  "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)
```

```
## <U+2714> Saving 'gene_ids' to 'data/gene_ids.rda'
```

Neuroexclusivity

Neuroexclusivity data consists of gene expression collected from Gexe Expression Atlas and the KEGG pathways themselves.

Expression neuroexclusivity

In this section we preprocess multiple wide .tsv files into a single long data.frame. We also create a template file for classifying tissues into nervous or non-nervous.

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. Found experiments:

- 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 data

We load and pivot all files to a long format.

```
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 information can be collapsed into a single level (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)
```

```
## <U+2714> Setting active project to 'C:/R/neuro'
## <U+2714> Saving 'gene_expression' to 'data/gene_expression.rda'
```

head(gene_expression)

ensembl_id	tissue	tpm
ENSG00000006125	adipose tissue	73
ENSG00000006125	adrenal gland	93
ENSG00000006125	bone marrow	59
ENSG00000006125	cerebral cortex	333
ENSG00000006125	colon	131
ENSG00000006125	duodenum	61

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.

```r
# 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. Essentialy, we subset the global STRING mapping between proteins and orthologous groups into a smaller dataset containing only information about the orthogroups related to our selected genes.

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
cogs_of_interest <- cogs %>% filter(string_id %in% gene_ids[["string_id"]]) %>% select(-taxid)

cogs %<>%
  # leave only eukaryotes
  filter(taxid %in% string_eukaryotes[["taxid"]]) %>%
  # leave only proteins which are part of cogs of interest
  filter(cog_id %in% cogs_of_interest[["cog_id"]])
```

```r
# exporting for package use
usethis::use_data(cogs, overwrite = TRUE)
```

```
## <U+2714> Setting active project to 'C:/R/neuro'
## <U+2714> Saving 'cogs' to 'data/cogs.rda'
```

```r
usethis::use_data(cogs_of_interest, overwrite = TRUE)
```

```
## <U+2714> Saving 'cogs_of_interest' to 'data/cogs_of_interest.rda'
```

Network

In this section we search the STRING API for our proteins of interest and recompute the combined interaction score.

Retrieving network data

Querying the API endpoint for the STRING IDs we collected.

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

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

Table 12: STRING interaction network with channel specific scores.

api_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/api_ids.tsv
Source: http://string-db.org/api/tsv/get_string_ids

Now we need to make sure that the API succesfully resolves the protein IDs we searched for.

```r
api_ids <- read_tsv("download/api_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)

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

```
## [1] TRUE
```

Once IDs are correct, we can query the network API endpoint to obtain the 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.

#	Col. name	Col. type	Used?	Example	Description
	string_edgelist				
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)
)

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

Analysis

Analysis

```
#leave this chunk
```