Anexo: *Scripts* desarrollados. Trabajo de Fin de Máster: "Modelización metabólica de comunidades microbianas estables crecidas con fuentes de carbono y energía únicas y simples"

Silvia Talavera Marcos

Scripts desarrollados

Este Anexo recoge los *scripts* desarrollados para automatizar los análisis correspondientes al Trabajo de Fin de Máster "Modelización metabólica de comunidades microbianas estables crecidas con fuentes de carbono y energía únicas y simples" (Silvia Talavera Marcos, Máster en Bioinformática y Biología Computacional, Universidad Autónoma de Madrid, 2020-2021).

A continuación se incluye una breve descripción de todos ellos:

- **modelado.R** incluye el alineamiento con Nucmer, la creación de modelos con CarveMe y (opcionalmente) el análisis con Smetana para una pareja de nodos dada.
- **annotate.R** se encarga de la anotación funcional con eggNOG-mapper y, opcionalmente, la creación de un modelo consenso. También es capaz de llamar a Nucmer para iniciar el proceso desde el principio.

Las funciones definidas para estos scripts se encuentran en **utils.R**.

- **RefrFBA.py** es un *script* que se encarga de ejecutar simulaciones de FBA para todos los modelos de un PCG dado. Devuelve las tasas de crecimiento en forma de archivos *.csv* y por la salida estándar.
- **parser.R** se encarga de generar informes en formato de texto plano que resumen los resultados de Smetana.
- Por último, se han desarrollado dos scripts en Python para crear los consensos: **consenso.py** crea un modelo SBML a partir de otros modelos SBML y **consenso_EGG.py** crea una tabla de anotaciones consenso a partir de múltiples archivos de anotaciones.

Todos los *scripts* aquí incluidos se pueden encontrar también en el repositorio https://github.com/urihs/TFM. En todos los casos se incluye el código del *script* seguido de una explicación del funcionamiento del mismo.

modelado.R

```
#!/usr/bin/env Rscript

start.time <- Sys.time()

# -------------------------
#        FUNCIONES
# -------------------------

library("parallel")
library("optparse")
home <- strsplit(paste0("./",getopt::get_Rscript_filename()),split="/")[[1]]
home <- paste(home[-length(home)],collapse="/")
source(paste0(home,"/utils.R")) # cargo las funciones del paquete



# -------------------------------
# 1 --> Definiciones preliminares
# -------------------------------
```

```
option_list <- list(
  make_option(c("-n", "--nodes"), type="character", default=NULL,
              help="Node input information text file name (e.g. 'my_node_data.txt'). Th
e file needs to have the following format:\n\t\tNode35562    k__Bacteria;p__Proteobacte
ria;c__Gammaproteobacteria;o__Enterobacteriales;f__Enterobacteriaceae;\n\t\tNode27828 k_
_Bacteria;p__Proteobacteria\nThe taxonomy must include at least two fields, with the sec
ond one being the phylum.", metavar="character"),
  make_option(c("-m", "--medium"), type="character", default="M9",
              help="medium (e.g. 'M9', 'M9[glc]'", metavar="character"),
  make_option(c("--mediadb"), type="character", default=paste0(home,"/my_media.tsv"),
              help="media database file name", metavar="character"),
  make_option(c("-c", "--checking"), type="logical", default=FALSE,
              help="TRUE or FALSE (default). If TRUE, generated and analysed models are
limited to those pairs of species that are co-ocurring in pairs in one or more samples o
f a specified experiment.", metavar="logical"),
  make_option(c("-e", "--experiment"), type="character", default=NULL,
              help="Experiment input information text file name (e.g. 'table.from_biom.
tsv'). The tab-separated file needs to have the following format:\n\t\t#OTU ID\tS1\tS2\t
S3\t(...)\n\t\tO1\t000000\t000001\t000002\t(...) ", metavar="character"),
  make_option(c("--coupling"), type="logical", default=TRUE,
              help="If TRUE, smetana computes an aditional analysis without the --no-co
upling option (see smetana help). Default is TRUE.", metavar="logical"),
  make_option(c("--nucmer"), type="character", default=paste0(home,"/MUMmer3.23/nucmer")
,
              help="Path to the nucmer executable (e.g. './MUMmer3.23/nucmer', '~/my_ap
ps/nucmer'.", metavar="character"),
  make_option(c("--showcoords"), type="character", default=paste0(home,"/MUMmer3.23/show
-coords"),
              help="Path to the show-coords executable (e.g. './MUMmer3.23/show-coords'
, '~/my_apps/show_coords'.", metavar="character"),
  make_option(c("--tree"), type="character", default=paste0(home,"/99_otus_nodes.tree"),
              help="16S phylogenetic tree file name. The node names of the trees should
be modified, and a genuine name should be given for all. This could be done for example
using R with the function 'makeNodeLabel' from 'ape' package. Default is './99_otus_node
s.tree' (Greengenes gg_13_5).", metavar="character"),
  make_option(c("--fasta"), type="character", default=paste0(home,"/99_otus.fasta"),
              help="16S sequences to be analyzed in multifasta format. Default is './99
_otus_nodes.tree' (Greengenes gg_13_5).", metavar="character"),
  make_option(c("--db16s"), type="character", default=paste0(home,"/bac120_ssu_reps_r95.
fna"),
              help="16S sequences database. Nucmer will align the tree leaves' 16S sequ
ences to this database. Default is './bac120_ssu_reps_r95.fna' (from GTDB https://data.a
ce.uq.edu.au/public/gtdb/data/releases/release95/95.0/genomic_files_reps/).", metavar="c
haracter"),
  make_option(c("--dbproteins"), type="character", default=paste0(home,"/protein_faa_rep
s/bacteria/"),
              help="Aminoacid sequences database. CarveMe will take files from here tha
t correspond to Nucmer hits and create SBML models from those files. Default is 'protein
_faa_reps/bacteria/' (from GTDB https://data.ace.uq.edu.au/public/gtdb/data/releases/rel
ease95/95.0/genomic_files_reps/).", metavar="character"),
  make_option(c("--run_smetana"), type="logical", default=TRUE,
              help="If TRUE, runs a Smetana analysis for inter-node pairs. If FALSE, sk
ips the Smetana analysis. Default: TRUE.", metavar="logical"),
  make_option(c("--cores"), type="numeric", default=4,
              help="Number of cores to use in parallelization processes (mclapply). Def
ault: 4.", metavar="numeric"))


parser <- OptionParser(option_list=option_list)
```

```
opt <- parse_args(parser)

t <- read.table(opt$nodes,sep = "\t")
nodos      <- levels(t[[1]])
taxonom    <- levels(t[[2]])
medium     <- opt$medium
mediadb    <- opt$mediadb
checking   <- opt$checking
run_smetana<- opt$run_smetana
coupling   <- opt$coupling

if (checking==TRUE & is.null(opt$experiment)) {
  stop("When --checking TRUE, a experiment file must be specified")
  } else if (checking == TRUE) {
  exp = opt$experiment
  }

nucmer_path     <- opt$nucmer
showcoords      <- opt$showcoords

tree_file         <- opt$tree
otus_fasta_file   <- opt$fasta
db_16S            <- opt$db16s
db_protein_folder <- opt$dbproteins

grampospath <- paste0(home,"/grampos.csv")
gramnegpath <- paste0(home,"/gramneg.csv")

cores <- opt$cores

# ------------------------
# 2 --> carga de archivos
# ------------------------
if (!require("ape", quietly=TRUE)) BiocManager::install("ape")
tn = ape::read.tree(tree_file)
nodos_hojas <- mclapply(nodos, function(nodo) {ape::extract.clade(tn, nodo)}, mc.cores=
cores)

# divido el fasta en dos archivos para facilitar su parsing después:
system(paste0("cat ",otus_fasta_file," | grep '>' > 99_otus_col1"))
system(paste0('cat ',otus_fasta_file,' | grep ">" -v  > 99_otus_col2'))

df1 <- read.csv("99_otus_col1",header = F)
fasta <- read.csv("99_otus_col2",header = F)
rownames(fasta) <- sub(">", "", df1[,1]) # elimino ">"
fasta <- as.data.frame(t(fasta)) # Columna 1: > ID. Columna 2: secuencia

system("rm 99_otus_col*") # elimino archivos temporales

# ------------------------------------------------
# 3 --> alinear cada hoja, filtrar y hacer modelos
# ------------------------------------------------
# Asigno secuencias 16S a cada hoja
if (checking == TRUE) {
  # Pares de hojas de los dos nodos. INTER-NODO.
  filtered_pairs <- check(nodos=nodos_hojas, exp=exp,cores=cores)
  # De cada hoja que pase el checking, tomo la secuencia de 16S del fasta original.
  # Son todas las hojas de cada nodo que pasan cualquiera de los dos checkings.
  checked_tipl <- list(levels(filtered_pairs[,1]),levels(filtered_pairs[,2]))
```

```
  nodos_16S    <- mclapply(checked_tipl, function(n) {fasta[n]},mc.cores=cores)
  } else {
    # De cada hoja (sin checking), cojo la secuencia de 16S del fasta original
    nodos_16S    <- mclapply(nodos_hojas, function(nodo) {fasta[nodo$tip.label]},mc.cor
es=cores)
  }


system("mkdir models")
# Para cada nodo creo una carpeta de resultados
for (i in c(1:length(nodos))) {
  filepath   <- paste("models/",nodos[i],"/",sep="")
  system(paste("mkdir", filepath)) # la carpeta tendrá el mismo nombre que el nodo

  # Alineo cada hoja de cada nodo con Nucmer y obtengo un genoma adecuado para cada una,
  # seleccionando solo las hojas cuyos hits pasan un filtro de calidad
  nucmer_res_final <- find_alignment_hits(filepath, nodos_16S[[i]], nucmer_path, db_16S,
showcoords, cores)

  # Modelado con CarveMe de todas las hojas de cada nodo que pasan el filtro de Nucmer
  print(paste0("Creating models for ",nodos[i],"..."))
  dump <- simplify2array(mclapply(nucmer_res_final,
                                  FUN = function(line) {carve(line, taxonom[i], filepa
th, db_protein_folder)},mc.cores=cores))
  print(paste0("Finished modelling for ",nodos[i],"."))

  }

# -------------------------------------
# 4 --> análisis metabólico con Smetana
# -------------------------------------

if (run_smetana == TRUE) {
  # Definimos la lista de parejas a analizar
  if (checking == TRUE) {
    pairs <- filtered_pairs
    } else {
      pairs <- expand.grid(nodos_hojas[[1]]$tip.label, nodos_hojas[[2]]$tip.label, KEEP
.OUT.ATTRS = F)
    }

  # Ejecutamos Smetana para cada pareja inter-nodo de hojas para las que se ha creado
  # un modelo. Esta lista se parejas se guardará en smetana_results/generated_pairs.txt
.
  # También guardamos la lista de parejas que no han sido analizadas por Smetana, en el
  # archivo smetana_results/filtered_out_pairs.txt. Incluimos el porcentaje de parejas
  # que pasan y no pasan el filtro.
  #~~~~~~~~~~~~~~~~~~
  output = "smetana_results/"
  system(paste0("mkdir ",output))
  system(paste0("mkdir ",output,"global"))
  system(paste0("mkdir ",output,"detailed"))
  if (coupling==TRUE) {
    output_coupling = paste0(output,"coupling/")
    system(paste("mkdir",output_coupling))
    system(paste0("mkdir ",output_coupling,"global"))
    system(paste0("mkdir ",output_coupling,"detailed"))
  } else {
    output_coupling = NULL}
```

```
  #~~~~~~~~~~~~~~~~~
  generated_pairs_filename = "generated_pairs.txt"
  dump <- file.create(paste0(output,generated_pairs_filename)) # vaciamos el archivo, de
existir, o lo creamos si no existe

  pairs <- as.data.frame(t(pairs)) # preparamos la matriz para el siguiente paso
  failed_pairs <- mcmapply(pairs, FUN=function(z) {
    smetana(z, modelfilepath = "models/", output=output,
                        coupling=coupling, output_coupling=output_coupling,
                        generated_pairs_filename=generated_pairs_filename)
    }, mc.cores=cores)

  failed_pairs[simplify2array(mclapply(failed_pairs, is.null, mc.cores=cores))] <- NULL

  # Anotamos las parejas que no han sido analizadas por Smetana (no pasaron el filtro de
Nucmer)
  write(x=paste0("filtered out: ", 100*length(failed_pairs)/length(pairs[,1])/2,"%"), f
ile=paste0(output,"filtered_out_pairs.txt"))
  dump <- mclapply(colnames(failed_pairs), FUN=function(col){
    cat(failed_pairs[,col],"\n", file=paste0(output,"filtered_out_pairs.txt"), append=T)
    }, mc.cores=cores)

  print(paste0("Finished SMETANA analysis."))
}

end.time <- Sys.time()
time.taken <- end.time - start.time
print(paste0("Execution time: ",format(time.taken,format = "%H %M %S")))```
```

Uso: `modelado.R [options]`

Argumentos:

Este *script*, como todos los demás, se ejecuta llamándolo desde el terminal. Los argumentos se recogen con la función `OptionParser` de la librería `optparse`. La opción `--help` devuelve una explicación en inglés de todos ellos.

- **`-n, --nodes`**. Nombre del archivo de texto con información sobre los dos nodos de interés (e. g. 'my_node_data.txt'). El formato del archivo debe ser el que se muestra a continuación. La taxonomía debe incluir al menos dos filos, correspondiendo el segundo al filo:

  ```
    Node35562   k__Bacteria;p__Proteobacteria;c__Gammaproteobacteria;o__Enterobacteri
  ales;f__Enterobacteriaceae;

    Node27828   k__Bacteria;p__Proteobacteria
  ```

- **`-m, --medium`**. Medio (e.g. 'M9', 'M9[glc]').
- **`--mediadb`**. Archivo con la base de datos de medios, por defecto "my_media.tsv" incluido en la carpeta original del *script*.
- **`-c, --checking`**. Si `TRUE`, los modelos generados y analizados se limitan a aquellos pares de especies que coinciden en una o más muestra de un experimento especificado (corresponde a la Estrategia 1). `FALSE` (Estrategia 2) por defecto.
- **`-e, --experiment`**. Nombre del archivo de texto con información del experimento de interés (e.g. 'table.from_biom.tsv'). Es un archivo separado por tabulaciones con el siguiente formato:

```
#OTU ID S1  S2  S3  (...)

O1  000000  000001  000002  (...)
```

- **`--coupling`**. Si TRUE, Smetana hace un análisis adicional sin su opción –no-coupling.
- **`--nucmer`**. Ruta del ejecutable de Nucmer (e.g. ‘./MUMmer3.23/nucmer’, ‘~/my_apps/nucmer’).
- **`--showcoords`**. Ruta del ejecutable del show-coords de MUMmer (e.g. ‘./MUMmer3.23/show-coords’, ‘~/my_apps/show_coords’).
- **`--tree`**. Nombre del archivo del árbol filogenético 16S. Los nombres de los nodos deben modificarse para que coincidan con los del archivo de información de nodos (`-n, --nodes`). Esta modificación se puede hacer fácilmente por ejemplo usando la función `makeNodeLabel` del paquete `ape` de R. El árbol por defecto es ‘99_otus_nodes.tree’ (Greengenes gg_13_5), en la carpeta original del *script*.
- **`--fasta`**. Archivo multifasta de secuencias 16S de las hojas del árbol utilizado. El archivo por defecto es’99_otus_nodes.tree’ (Greengenes gg_13_5) en la carpeta original del *script*.
- **`--db16s`**. Base de datos de secuencias 16S contra la cual Nucmer va a alinear las secuencias de las hojas del árbol. Por defecto se usa ‘bac120_ssu_reps_r95.fna’ (GTDB https://data.ace.uq.edu.au/public/gtdb/data/releases/release95/95.0/genomic_files_reps/).
- **`--dbproteins`**. Base de datos de secuencias de aminoácidos. CarveMe tomará de aquí los archivos que correspondan a los *hits* de Nucmer y creará modelos metabólicos a partir de ellos. Por defecto es ‘protein_faa_reps/bacteria/’ (GTDB https://data.ace.uq.edu.au/public/gtdb/data/releases/release95/95.0/genomic_files_reps/).
- **`--run_smetana`**. Si `TRUE`, ejecuta un análisis (global y detallado) de Smetana para parejas internodo. Si `FALSE`, se salta este paso.
- **`--cores`**. Número de núcleos usados en procesos de paralelización (mclapply y mcmapply). Por defecto son 4.

Descripción: `modelado.R` se encarga de generar con CarveMe modelos de los nodos/PCGs especificados en la entrada (`--nodos`) y, opcionalmente, los análisis de Smetana correspondientes.

En primer lugar, si `--checking` es `TRUE`, se seleccionan las hojas de cada nodo que sí coincidían con las hojas del otro nodo en al menos una muestra del experimento especificado (guardado en la variable `exp`). Se hace con la función `check` definida en `utils.R`.

En segundo lugar, se hace un alineamiento con Nucmer de las secuencias 16S (GreenGenes) de las hojas deseadas para cada nodo y una base de datos de genomas completos, GTDB. Solo se seleccionan aquellas asignaciones que superan un filtro fijo (identidad > 97% y cobertura (de la referencia/*hit*) > 90%). A las hojas cuyo alineamiento pase el filtro se les asigna un archivo con una lista completa de todas las proteínas de su genoma. Ese archivo es a partir del cual CarveMe generará modelos SBML. La función principal de este proceso es `find_alignment_hits`.

En tercer lugar, se generan los modelos con CarveMe. Los modelos se guardan con el nombre de la hoja correspondiente. Es uno de los pasos más largos, pero está paralelizado con `mcapply` para ahorrar tiempo de ejecución. Además, si ya existe un modelo con el nombre de esa hoja no se genera de nuevo, permitiendo ahorrar tiempo si la ejecución de `modelado.R` se ha reiniciado. Cada nodo tiene su propia carpeta con el nombre del nodo (el que se indique en el archivo dado con `--nodos`) que está dentro, a su vez, de la carpeta “models”. Esta carpeta se guarda automáticamente en el directorio de trabajo desde el que se ha ejecutado el *script* (que no tiene por qué coincidir con el directorio donde se encuentra el mismo).

Por último, se ejecuta el paso de Smetana, `run_smetana==TRUE`. Primero se crea una lista con todas las parejas inter-nodo posibles de modelos o, si `checking==TRUE`, solo aquellas parejas inter-nodo que coincidieran en la misma muestra del experimento. A continuación hay, para cada pareja, dos o tres

ejecuciones según si `coupling` es `TRUE` o no. En cada ejecución, Smetana va comprobando a) que los modelos SBML existen y b) que el análisis no existiera ya. Asimismo, estas ejecuciones están paralelizadas con `mcapply`. Los análisis se guardan en la carpeta "smetana_results", que también se crea en el directorio de trabajo, con subcarpetas para cada tipo de análisis.

Durante la ejecución se imprimen en la salida estándar mensajes que indican la actividad del *script*. Al final se imprime el tiempo de ejecución.

Salida:

- Modelos SBML para OTUs/hojas de los nodos/PCGs especificados.
- Si `--run_SMETANA` es `TRUE`, análisis globales y detallados de Smetana.
- Si `--coupling` es `TRUE`, se generan también informes detallados sobre los metabolitos acoplados al crecimiento.
- También se generan archivos con el output del alineamiento de Nucmer, incluyendo un archivo `queries_v_hits` para cada PCG.

Requiere: R (3.5.0), CarveMe (1.4.0+), Smetana (1.2.0), NUCmer (cualquier versión; usamos la 3.1; MUMmer 3.23), ape (se instala automáticamente), parallel, optparse

---

annotate.R

```
#!/usr/bin/env Rscript

start.time <- Sys.time() # para devolver al final el tiempo de ejecución
# INPUT: annotate.R -g genomes -o outputname --outputdir outputdir
# INFO: THIS SCRIPT TAKES A LIST OF GENOMES, ANNOTATES ALL OF THEM WITH EGGNOG MAPPER AN
D RETURNS THE ANNOTATED FILES PLUS A CONSENSUS ONE

# =======================
#        FUNCIONES
# =======================
library("optparse")
library("parallel")

home <- strsplit(paste0("./",getopt::get_Rscript_filename()),split="/")[[1]]
home <- paste(home[-length(home)],collapse="/")
source(paste0(home,"/utils.R")) # cargo las funciones del paquete

# ==============
#  Definiciones
# ==============

option_list <- list(
  make_option(c("-g", "--genomes"), type="character", default=NULL,
              help="File containing the list of genomes to annotate. The format may be
the following:\n\tRS_GCF_000281895.1\n\tRS_GCF_900100495.1\n\tRS_GCF_900104015.1\n\t(...
)", metavar="character"),
  make_option(c("-n", "--nodes"), type="character", default=NULL,
              help="Node input information text file name (e.g. 'my_node_data.txt'). Th
e file needs to have the following format:\n\t\tNode35562    k__Bacteria;p__Proteobacte
ria;c__Gammaproteobacteria;o__Enterobacteriales;f__Enterobacteriaceae;\n\t\tNode27828 k_
_Bacteria;p__Proteobacteria\nThe taxonomy must include at least two fields, with the sec
ond one being the phylum.", metavar="character"),
  make_option(c("--skip_consensus"), type="logical", default=FALSE,
              help="If TRUE, eggNOG-mapper creates the annotation files but a consensus
file is not made. Default: FALSE.", metavar="logical"),
  make_option(c("-m", "--medium"), type="character", default="M9",
```

```
              help="medium (e.g. 'M9', 'M9[glc]'", metavar="character"),
  make_option(c("--outputdir"), type="character", default="./annotate_results",
              help="output directory for annotation files and the consensus model(s)",
metavar="character"),
  make_option(c("-o","--outputname"), type="character", default="node_consensus",
              help="output name for the consensus model(s)", metavar="character"),
  make_option(c("--mediadb"), type="character", default=paste0(home,"/my_media.tsv"),
              help="media database file name", metavar="character"),
  make_option(c("-c", "--checking"), type="logical", default=FALSE,
              help="TRUE or FALSE (default). If TRUE, annotated genomes are limited to
those pairs of species that are co-ocurring in pairs in one or more samples of a specifi
ed experiment. Only applicable when input is --nodes.", metavar="logical"),
  make_option(c("-e", "--experiment"), type="character", default=NULL,
              help="Experiment input information text file name (e.g. 'table.from_biom.
tsv'). Only needed when input is --nodes and --checking is TRUE. The tab-separated file
needs to have the following format:\n\t\t#OTU ID\tS1\tS2\tS3\t(...)\n\t\tO1\t000000\t000
001\t000002\t(...) ", metavar="character"),
  make_option(c("--nucmer"), type="character", default=paste0(home,"/MUMmer3.23/nucmer")
,
              help="Path to the Nucmer executable (e.g. './MUMmer3.23/nucmer', '~/my_ap
ps/nucmer'). Only needed when input is --nodes.", metavar="character"),
  make_option(c("--showcoords"), type="character", default=paste0(home,"/MUMmer3.23/show
-coords"),
              help="Path to the show-coords executable (e.g. './MUMmer3.23/show-coords'
, '~/my_apps/show_coords'). Only needed when input is --nodes.", metavar="character"),
  make_option(c("--emapper_path"), type="character", default=paste0(home,"/eggnog-mapper
-master/emapper.py"),
              help="Path to the emapper.py executable file (e.g. './eggnog-mapper-my_ve
rsion/emapper.py').",metavar="character"),
  make_option(c("--tree"), type="character", default=paste0(home,"/99_otus_nodes.tree"),
              help="16S phylogenetic tree file name. Only needed when input is --nodes.
The node names of the trees should be modified, and a genuine name should be given for a
ll. This could be done for example using R with the function 'makeNodeLabel' from 'ape'
package. Default is './99_otus_nodes.tree' (Greengenes gg_13_5).", metavar="character"),
  make_option(c("--fasta"), type="character", default=paste0(home,"/99_otus.fasta"),
              help="16S sequences of all the leaves of the tree, in multifasta format.
Only needed when input is --nodes. Default is './99_otus_nodes.tree' (Greengenes gg_13_5
).", metavar="character"),
  make_option(c("--db16s"), type="character", default=paste0(home,"/bac120_ssu_reps_r95.
fna"),
              help="16S sequences database. Only needed when input is --nodes. Nucmer w
ill align the tree leaves' 16S sequences to this database. Default is './bac120_ssu_reps
_r95.fna' (from GTDB https://data.ace.uq.edu.au/public/gtdb/data/releases/release95/95.0
/genomic_files_reps/).", metavar="character"),
  make_option(c("--dbproteins"), type="character", default=paste0(home,"/protein_faa_rep
s/bacteria/"),
              help="Aminoacid sequences database. EggNOG-mapper will annotate files fro
m here. Default is 'protein_faa_reps/bacteria/' (from GTDB https://data.ace.uq.edu.au/pu
blic/gtdb/data/releases/release95/95.0/genomic_files_reps/).", metavar="character"),
  make_option(c("--dmnd_db"), type="character", default=NULL,
              help="Path to a custom Diamond database. Specially useful when using a Di
amond version different from the eggNOG-mapper one, such as the one created with create_
compatible_database.R.", metavar="character"),
  make_option(c("--cores"), type="numeric", default=4,
              help="Number of cores to use in parallelization processes (mclapply). Def
ault: 4.", metavar="numeric"))

parser <- OptionParser(option_list=option_list, usage = "Desktop/TFM/annotate.R [options
]\n\nThe main input data is either --genomes or --nodes")
```

```
opt <- parse_args(parser)

cores <- opt$cores
skip_consensus <- opt$skip_consensus

if (!is.null(opt$nodes) && is.null(opt$genomes)) {
  skip_alignment <- FALSE
  t <- read.table(opt$nodes,sep = "\t")
  nodos      <- levels(t[[1]])
  taxonom    <- levels(t[[2]])
  nucmer_path<- opt$nucmer
  showcoords <- opt$showcoords
  tree_file         <- opt$tree
  otus_fasta_file   <- opt$fasta
  db_16S            <- opt$db16s

  checking   <- opt$checking
  if (checking==TRUE & is.null(opt$experiment)) {
    stop("When --checking TRUE, a experiment file must be specified")
  } else if (checking == TRUE) {
    exp = opt$experiment
  }

} else if (is.null(opt$nodes) && !is.null(opt$genomes)) {
  skip_alignment=TRUE

} else if (is.null(opt$nodes) && is.null(opt$genomes)) {
  stop("Please provide an input file. It may be either a list of genomes to annotate or
a node data file.")

} else { # both --genomes and --nodes were provided
  stop("Only one kind of input (--genomes or --nodes) can be provided")}

# Annotation variables
outputdir         <- opt$outputdir
outputname        <- opt$outputname
db_protein_folder <- opt$dbproteins
emapper_path      <- opt$emapper_path


if (is.null(opt$dmnd_db)) {
  emapper_folder  <- system(paste0("echo $(dirname ",emapper_path,")"),intern=TRUE)
  dmnd_db         <- paste0(emapper_folder,"/data/eggnog_proteins.dmnd")
} else
  dmnd_db         <- opt$dmnd_db

# ========================
#   Anotar los genomas
# ========================

if (skip_alignment == TRUE) {
  genomes <- scan(genomes,character(),sep="\n")

  #   Anotar el genoma asignado a cada hoja con eggNOG-mapper
  # ------------------------------------------------------------
  annotate(genomes, outputdir, db_protein_folder, emapper_path, cores)
  system(paste0("rm ",outputdir,"/*seed_orthologs"))
```

```
  #   Crear genoma consenso
  # -------------------------
  if (!skip_consensus) {
    system(paste("python3 consenso_EGG.py $(realpath",outputdir,") $(realpath",outputdir
,")",outputname))

  #  Fabricar modelo con CarveMe
  # ----------------------------
    system(paste0("carve --egg ",outputname,".tsv -o ",outputdir,"/",outputname,".xml")
)

  }


} else {

  #   Obtener secuencias 16S para todas las hojas de ambos nodos
  # ------------------------------------------------------------
  if (checking == TRUE) {
    filtered_pairs <- check(nodos=nodos_hojas, exp=exp)
    checked_tipl <- list(levels(filtered_pairs[,1]),levels(filtered_pairs[,2]))
    nodos_16S    <- mclapply(checked_tipl, function(n) {fasta[n]},mc.cores=cores)
  } else {
    nodos_16S    <- mclapply(nodos_hojas, function(nodo) {fasta[nodo$tip.label]},mc.cor
es=cores)
  }

  system("mkdir models")
  # Para cada nodo haremos alineamientos, anotaciones y un consenso

  genomes=list()
  for (i in c(1:length(nodos))) {
    filepath   <- paste("models/",nodos[i],"/",sep="")
    system(paste("mkdir", filepath))

    #    Alineamiento con Nucmer
    # ----------------------------
    nucmer_res_final <- find_alignment_hits(filepath, nodos_16S[[i]], nucmer_path, db_1
6S, showcoords, cores)
    genomes[[i]] <- scan(file=paste0(filepath,"genomes"),what=character(),sep="\n")

    #   Anotar el genoma asignado a cada hoja con eggNOG-mapper
    # ---------------------------------------------------------
    node_outputdir  <- paste0(outputdir,"/",nodos[i])
    node_outputname <- paste0(outputname,"_",nodos[i])
    # En esta(s) carpeta(s) se guardará la lista de los nombres de los genomas obtenido
s por Nucmer
    system(paste0("mkdir ",node_outputdir))

    annotate(genomes[[i]], node_outputdir, db_protein_folder, emapper_path, cores)
    system(paste0("rm ",outputdir,"/*seed_orthologs"))

    if (!skip_consensus) {

    #   Crear genoma consenso
    # -------------------------
      system(paste("python3 consenso_EGG.py $(realpath",node_outputdir,") $(realpath",n
ode_outputdir,")",node_outputname))
```

```
    #  Fabricar modelo con CarveMe
    # ----------------------------
      system(paste0("carve --egg ",node_outputname,".tsv ",gram(taxonom[i])," -o ",node
_outputdir,"/",node_outputname,".xml"))
    }
  }
}


end.time <- Sys.time()
time.taken <- end.time - start.time
print(paste0("Execution time: ",format(time.taken,format = "%H %M %S")))
```

Uso: `annotate.R [options]`

El *input* principal es bien `--nodes` o bien `--genomes`.

Argumentos:

- **`-g, --genomes`**. Ruta de un archivo de texto que contenga una lista de genomas a anotar. El formato debe ser el siguiente, en consonancia con el de la base de datos de genomas de proteínas dada:

  ```
  RS_GCF_000281895.1

  RS_GCF_900100495.1

  RS_GCF_900104015.1

  (...)
  ```

- **`-n, --nodes`**. Ruta de un archivo de texto con información de los nodos de interés. El formato debe ser el siguiente:

  ```
  Node35562   k__Bacteria;p__Proteobacteria;c__Gammaproteobacteria;o__Enterobacteri
ales;f__Enterobacteriaceae;

  Node27828   k__Bacteria;p__Proteobacteria
  ```

- **`--skip_consensus`**. Si `TRUE`, eggNOG-mapper crea los archivos anotados pero no se crean modelos ni un consenso a partir de los mismos. Por defecto es `FALSE`.
- **`-m, --medium`**. Medio (e. g. 'M9', ' M9[glc]').
- **`--outputdir`**. Directorio de salida para los archivos de anotación y los modelos consenso.
- **`-o, --outputname`**. Nombre del modelo consenso creado (si el *input* es `-g`) o nombre común de los modelos consenso (si el *input* es `-n` y son varios nodos).
- **`--mediadb`**. Ruta del archivo de medios de cultivo.
- **`--checking`**. Si `TRUE`, solo se anotan aquellos genomas cuyas secuencias hayan sido detectadas en el experimento especificado. Solo se aplica si se hace alineamiento.
- **`-e, --experiment`**. Ruta del archivo con información del experimento de interés cuando `--checking` es `TRUE` (e. g. 'table.from_biom.tsv'). Solo es necesario si se hace alineamiento y `--checking` es `TRUE`. El archivo debe tener el siguiente formato:

  ```
  #OTU ID S1   S2   S3   (...)
  O1   000000   000001   000002   (...)
  ```

- `--nucmer`. Ruta del ejecutable de Nucmer (e. g. ‘./MUMmer3.23/nucmer’, ‘~/my_apps/nucmer’). Solo es necesario si se hace alineamiento.
- `--show-coords`. Ruta del ejecutable de show-coords (e. g. ‘./MUMmer3.23/show-coords’, ‘~/my_apps/show_coords’). Solo es necesario si se hace alineamiento.
- `--emapper_path`. Ruta del ejecutable de eggNOG-mapper (e. g. ‘./eggnog-mapper-my_version/emapper.py’).
- `--tree`. Nombre del archivo del árbol filogenético 16S. Solo es necesario si se hace alineamiento. Los nombres de los nodos deben modificarse para que coincidan con los del archivo de información de nodos (`-n, --nodes`). Esta modificación se puede hacer fácilmente por ejemplo usando la función `makeNodeLabel` del paquete `ape` de R. El árbol por defecto es ‘99_otus_nodes.tree’ (Greengenes gg_13_5), en la carpeta original del *script*.
- `--fasta`. Archivo multifasta de secuencias 16S de las hojas del árbol utilizado. Solo es necesario si se hace alineamiento. El archivo por defecto es’99_otus_nodes.tree’ (Greengenes gg_13_5) en la carpeta original del *script*.
- `--db16s`. Base de datos de secuencias 16S contra la cual Nucmer va a alinear las secuencias de las hojas del árbol. Solo es necesario si se hace alineamiento. Por defecto se usa ‘bac120_ssu_reps_r95.fna’ (GTDB https://data.ace.uq.edu.au/public/gtdb/data/releases/release95/95.0/genomic_files_reps/).
- `--dbproteins`. Base de datos de secuencias de aminoácidos. EggNOG-mapper tomará de aquí los genomas a anotar, ya sean dados por los *hits* de Nucmer (–nodes) o por el usuario (–genomes). Por defecto es ‘protein_faa_reps/bacteria/’ en la carpeta original del *script* (GTDB https://data.ace.uq.edu.au/public/gtdb/data/releases/release95/95.0/genomic_files_reps/).
- `--dmnd_db`. Ruta de una base de datos de Diamond personalizada. Especialmente útil cuando se utiliza una base de datos diferente a la estándar de eggNOG-mapper, como puede ser una creada con create_compatible_database.R.
- `--cores`. Número de cores a usar en la paralelización de procesos (`mclapply`). Por defecto son 4.

Descripción: En primer lugar se obtiene la lista de genomas a anotar. Esta lista puede tomarse directamente de los datos de entrada (archivo de texto dado con la opción `--genomes`) o bien realizando un alineamiento del nodo o los nodos especificados con `--nodos`. El proceso en este caso es similar al explicado para `modelado.R`, utilizando la función `find_alignment_hits`.

En segundo lugar, cada lista de genomas es anotada con eggNOG-mapper mediante la función `annotate` definida en `utils.R`. Este proceso está paralelizado internamente con `mcapply`.

Por último, si se activó la opción de crear consenso, se hace una llamada a `consenso_EGG.py` y finalmente a CarveMe.

Salida: Archivos de anotaciones de genomas para cada genoma de interés. Si `--skip_consensus` es `FALSE`, también se genera un modelo SBML-FCB2 consenso para cada nodo.

Requiere: R (3.5.0), CarveMe (1.4.0+), eggNOG-mapper (2.0.1), parallel, optparse. Si se utiliza la función de alineamiento con Nucmer, se requieren también NUCmer (cualquier versión; usamos la 3.1; MUMmer 3.23) y ape (se instala automáticamente).

---

create_compatible_database.R

```
#!/usr/bin/env Rscript

# ------------------------------------------------------------
# Crear una base de datos compatible con el diamond del PATH
# ------------------------------------------------------------
library("optparse")

option_list <- list(
  make_option(c("-o","--outputdir"), type="character", default=".",
              help="Output directory for the new database.", metavar="character"),
  make_option(c("-e","--emapper_path"), type="character", default=NULL,
              help="EggNOG-mapper directory. Contains the old database and the old Diam
ond version.", metavar="character"),
  make_option(c("-d","--diamond"), type="character", default="diamond",
              help="New Diamond version location. Default is 'diamond' (PATH).", metava
r="character"))

parser <- OptionParser(option_list=option_list)
opt <- parse_args(parser)

outputdir <- opt$outputdir
diamond   <- opt$diamond

if (is.null(opt$emapper_path)) {
  stop("Please specify the eggNOG-mapper path")
} else {
  emapper_path <- opt$emapper_path
}
old_db       <- paste0(emapper_path,"/data/eggnog_proteins.dmnd")
old_dmnd     <- paste0(emapper_path,"/bin/diamond")

make_compatible_database <- function(emapper_path) {
  print(paste0(old_dmnd," getseq --db ",old_db," > '",outputdir,"/eggnog_proteins.faa';
               ",diamond," makedb --in '",outputdir,"/eggnog_proteins.faa' --db ",output
dir,"/eggnog_proteins_compatible"))
}

make_compatible_database(emapper_path)

print(paste0("New Diamond database saved as: ",outputdir,"/eggnog_proteins_compatible.dm
nd"))
```

Al utilizar eggNOG-mapper junto a otra herramienta que use Diamond (como CarveMe), o simplemente por tener otra versión de Diamond instalada, es posible que se importe al entorno una versión de Diamond diferente a la que viene con eggNOG-mapper. En ese caso, podría haber incompatibilidad entre dicha versión de Diamond y la base de datos de Diamond incluida con el ejecutable de eggNOG-mapper (/data/eggnog_proteins.dmnd) y consecuentemente un fallo a la hora de anotar.

Este breve *script* sirve para obtener una base de datos Diamond en un formato distinto a la utilizada por eggNOG-mapper y compatible con la versión de Diamond que se esté utilizando o se quiera utilizar. Posteriormente se puede indicar a `annotate.R` que utilice esta nueva base de datos mediante `--dmnd_db`.

Uso: `create_compatible_databases.R [options]`

Argumentos:

- **`-o, --outputdir`**. Ruta del directorio de salida para la nueva base de datos.
- **`-e, --emapper_path`**. Ruta de la versión de eggNOG-mapper (contiene la versión antigua de

Diamond y su base de datos antigua).

- `-d, --diamond`. Versión de Diamond para la nueva base de datos. Si no se especifica, se usa la versión del PATH.

Descripción: Este *script* toma los datos de entrada y ejecuta un primer comando que convierte la base de datos antigua, en formato *.dmnd*, en un archivo *.faa*.

El segundo y último comando que ejecuta llama a la nueva versión de Diamond para crear, a partir del archivo .faa, una nueva base de datos en formato .dmnd. El archivo .faa se guarda también en la carpeta de salida.

Salida: /eggnog_proteins.faa y /eggnog_proteins_compatible.dmnd

Requiere: eggNOG-mapper (cualquier versión), Diamond (cualquier versión)

---

utils.R

```
#!/usr/bin/env Rscript
library(parallel)

nucmer <- function(nucmer_path, db_16S, fasta_16S) {
  # Alinea cada secuencia del archivo fasta indicado ("fasta_16S") con las secuencias
  # de la base de datos ("db_16S") y devuelve todos los hits en un archivo llamado
  # "./out.delta".
  system(paste(nucmer_path, db_16S, fasta_16S))
}




carve = function(line, taxonom, outputpath, db_protein_folder) {
  # Dado un string del tipo "<ID de hoja> <ID de genoma anotado>", crea un
  # modelo metabólico de dicha hoja a partir del archivo correspondiente al
  # genoma dado.
  leaf = strsplit(line, split=" ")[[1]][1]
  file = strsplit(line, split=" ")[[1]][2]
  outf = paste0(outputpath,leaf,".xml")
  if (file.exists(outf)) {
    print(paste0(outf," model already exists. Moving to the next one..."))
  } else {
    system(paste0("carve ",db_protein_folder,file,"_protein.faa ",
                  gram(taxonom)," -o ",outf)) # nombres de archivo = nombre de hoja
    print(paste0(outf," model created"))
  }
}




smetana = function(pair, modelfilepath="models/", output, coupling=TRUE, output_coupling
=NULL, generated_pairs_filename="generated_pairs.txt") {
  # Dado un string del tipo <hoja1> <hoja2>, lanza smetana para las hojas dadas,
  # si existen sus archivos. Si no existe alguno de los dos archivos o ninguno,
  # devuelve sus nombres. Si coupling==TRUE, se computan también los análisis sin
  # la opción de smetana "--no-coupling".
  # File creation
  if (coupling==TRUE & is.null(output_coupling)) {
    stop("A output name for the coupling results must be specified.")
  }
```

```
  filepath1 = paste0(modelfilepath,nodos[1],"/")
  filepath2 = paste0(modelfilepath,nodos[2],"/")
  m1 = pair[[1]]
  m2 = pair[[2]]
  if (file_test("-f",paste0(filepath1,m1,".xml")) & file_test("-f",paste0(filepath2,m2,"
.xml"))) {
    for (i in c("global","detailed")){
      output_filename=paste0(output,i,"/",m1,"_",m2,"_",medium)
      if (!file.exists(paste0(output_filename,"_",i,".tsv"))) { #solo crea el archivo s
i no existía ya
        print(paste0("Creating report ",output_filename,"_",i,"..."))
        system(paste0("smetana --",i," ",filepath1,m1,".xml"," ",filepath2,m2,".xml","
--flavor bigg -m ",medium,
                      " --mediadb ",mediadb," --molweight --no-coupling -o ",output_fil
ename))
        write(paste(m1,m2),file=paste0(output, generated_pairs_filename),append=TRUE) #
lista de parejas que se han analizado con Smetana
      } else {
        print(paste0(output_filename,"_",i,".tsv already exists. Moving to the next pai
r..."))
      }
    }
    if (coupling==TRUE) {  # se hace una ejecución más, pero solo para detailed.
      output_filename_c=paste0(output_coupling,i,"/",m1,"_",m2,"_",medium)
      if (!file.exists(paste0(output_filename,"_",i,".tsv"))) { #solo crea el archivo s
i no existía ya
        print(paste0("Creating report ",output_filename,"_",i,"..."))
        system(paste0("smetana --detailed"," ",filepath1,m1,".xml"," ",filepath2,m2,".x
ml"," --flavor bigg -m ",medium,
                      " --mediadb ",mediadb," --molweight -o ",output_filename_c))
      } else {
        print(paste0(output_filename_c,"_detailed.tsv already exists. Moving to the nex
t pair..."))
      }
    }
  } else {
    return(pair) # se devuelve
  }
}




emapper = function(input_fa, db_protein_folder, outputname, outputdir, emapper_path, cor
es) {
  system(paste0(emapper_path," -m diamond --cpu ",cores," --no_annot --no_file_comments
-i ",
                db_protein_folder, input_fa,"_protein.faa"," -o ",outputname," --output
_dir ",
                outputdir," --temp_dir /dev/shm --dmnd_db $PWD/",dmnd_db," --override")
)
  returned <- system(paste0(emapper_path," --annotate_hits_table ",outputdir,"/",outputn
ame,
                            ".emapper.seed_orthologs -o ",outputname," --output_dir ",o
utputdir," --override"))
  if (returned != 0) {
    stop("eggNOG-mapper returned non-zero status. If your Diamond version is different
from the eggNOG-mapper one
         (e.g. the CarveMe version is in the PATH) please create a new database with
```

```
        create_compatible_database.R
        and select it with --dmnd_db when next running annotate.R")
  }
}



check = function(nodos, exp, cores=4) {
  # Dada una pareja de nodos ("nodos", formato ape::phylo) y la ruta de un archivo con
  # datos experimentales ("exp"), determina y devuelve las combinaciones de OTUs
  # inter-nodo que coinciden en una misma muestra experimental
  exp = read.csv(exp,sep="\t",skip = 1,row.names=1)

  # Seleccionamos solo las hojas que estaban en el experimento, para agilizar el proceso
  lista_nodos <- mclapply(nodos, function(nodo) {nodo$tip.label[nodo$tip.label %in% rown
ames(exp)]},mc.cores=cores)

  # Obtengo todas las combinaciones de hojas para cada nodo
  pairs = expand.grid(lista_nodos[[1]], lista_nodos[[2]], KEEP.OUT.ATTRS = F)

  # Selecciono las parejas presentes en una misma muestra
  pairs[,3] = vector(mode="logical",length=length(pairs[,2])) # indicador de si coincid
en o no
  for (muestra in colnames(exp)){
    # anoto qué otus hay en cada muestra
    otus=rownames(subset(exp[muestra],exp[muestra]!=0))
    # indico para cada pareja si coinciden en esa muestra o en otra ya comprobada
    pairs[,3] = pairs[,3] | pairs[,1] %in% otus & pairs[,2] %in% otus
  }
  filtered_pairs  = subset(pairs,pairs[,3]==TRUE)[,0:2]
  return(filtered_pairs)
}



gram  = function(taxonom, gramneg = "gramneg.csv", grampos = "grampos.csv") {
  # Dado un string con la taxonomía en formato GreenGenes, devuelve si es
  # gram-positiva o gram-negativa. Si no se reconoce como G+ ni G-, devuelve
  # un string vacío.
  phylum  <- strsplit(taxonom, split=";")[[1]][2]
  gramneg <- levels(read.table(gramnegpath)[[1]])
  grampos <- levels(read.table(grampospath)[[1]])

  if (phylum %in% gramneg) {
    result <- "-u gramneg"
  }
  else if (phylum %in% grampos) {
    result <- "-u grampos"
  }
  else {
    result <- ""
  }
  return(result)
}
```

```
find_alignment_hits = function(filepath, node_16S, nucmer_path, db_16S, showcoords, core
s) {
  # Runs nucmer on each of the leaves of a given node (input is 16S sequences).
  # Creates 4 different files:

  #    - nucmer_unfiltered: table including all the best hits for every leaf before the
  #                         97% identity and 90% coverage filter.

  #    - nucmer_filtered: table including the best hits with over 97% identity and 90%
  #                       coverage. Some leaves may be filtered out at this point.
  #    - queries_v_hits: list of every leaf that passed the filter vs its best database
  #                      hit according to Nucmer. It's a two-column table

  #    - genomes: list of every database hit that passed the filter. One-column table.
  #

  # Returns a character vector with the leaf names and hits (the same list of the
  # queries_v_hits file).


  # Creo un archivo temporal en formato fasta que contenga las secuencias de ese nodo
  write(
    simplify2array(mclapply(colnames(node_16S), FUN=function(hoja){
      paste(paste(">",hoja,sep=""),as.character(node_16S[hoja][,]),sep="\n")},mc.cores=c
ores)),
    file="sec_temp.fasta")

  # Alineo cada una de sus hojas con la base de datos
  nucmer(nucmer_path, db_16S, "sec_temp.fasta")

  system(paste("mv out.delta",filepath))

  # Descarto hits con identidad por debajo de 97% con show-coords
  nucmer_res <- system(paste(showcoords," -c -l -I 97 ",filepath,
                             "out.delta",sep=""), intern = TRUE)
  # -c  Include percent coverage information
  # -l  Include the sequence length information
  # -I float    Set minimum percent identity to display

  # Creo archivo de mejores resultados sin filtrar por coverage
  header = nucmer_res[4:5]
  nucmer_res = gsub("|", "", nucmer_res[-(0:5)], fixed=T) # elimino separador
  nucmer_res = gsub("(?<=[\\s])\\s*|^\\s+|\\s+$", "", nucmer_res, perl=TRUE) # fusiono
espacios

  write(header, paste(filepath,"nucmer_unfiltered",sep=""))
  write(nucmer_res, paste(filepath,"nucmer_unfiltered",sep=""),append = TRUE) # guardo

  # Creo archivo de mejores resultados filtrados por %ID y coverage y lo guardo
  write(header, paste(filepath,"nucmer_filtered",sep=""))

  system(paste("awk '{ if ($10>=90) { print } }' ", # filtro por cobertura
               filepath,"nucmer_unfiltered > ",
               filepath,"nucmer_temp1",sep=""))
  system(paste("sort -r -k13,13 -k7,7 ", # ordeno por query y por %ID
               filepath,"nucmer_temp1 > ",
               filepath,"nucmer_temp2",sep=""))
```

```
  system(paste("awk -F ' ' -v q='' '{if ($13!=q) {q=$13; print}}' ", # escojo los mejore
s resultados
               filepath,"nucmer_temp2 >>",
               filepath,"nucmer_filtered",sep="")) #guardo

  # Lista filtrada de hits a modelar
  queries_v_hits = system(paste("awk -F ' ' -v q='' '{if ($13!=q) {q=$13; print $13,$12}
}' ",
                                filepath,"nucmer_temp2",sep=""), intern = TRUE)

  # guardo parejas como archivo (a modo de índice informativo)
  write(queries_v_hits, file=paste0(filepath, "queries_v_hits"))

  # guardo solo los genomas en un archivo (útil para anotar posteriormente por ejemplo)
  system(paste("awk -F ' ' -v q='' '{if ($13!=q) {q=$13; print $12}}' ",
               filepath,"nucmer_temp2 > ",filepath,"genomes",sep=""), intern = TRUE)

  # Borrado de archivos temporales
  system(paste("rm sec_temp.fasta ", filepath,"nucmer_temp*",sep=""))

  return(queries_v_hits)
}




annotate = function(genomes, outputdir, db_protein_folder, emapper_path, cores) {
  if (!file.exists(outputdir)){
    system(paste0("mkdir ",outputdir)) }

  N = length(genomes) # número total de genomas
  dump <- simplify2array(mclapply(genomes,FUN=function(genome) {
    print(paste0("Annotating genome ",genome," (total: ",N,")"))
    emapper(input_fa=genome,
            db_protein_folder = db_protein_folder,
            outputname=genome,
            outputdir=outputdir,
            emapper_path=emapper_path,
            cores=cores)
  },mc.cores=cores))
}
```

Uso: -

Argumentos: -

Descripción: Este *script* recoge las funciones más importantes de `modelado.R` y `annotate.R`. `nucmer`, `carve`, `emapper` y `smetana` lanzan estos 4 programas desde el terminal, preparando el comando a lanzar en cada caso e incluyendo en ocasiones mensajes de error si falta algún argumento. `annotate` es una función sencilla que se encarga de llamar a `emapper` para cada genoma.

Las funciones `carve` y `smetana` son un poco más complejas. En primer lugar, dado que deben ejecutar cientos o incluso miles de comandos según el caso (uno por cada modelo o análisis generado), los comandos están paralelizados con `mclapply` y `mcmapply`. Además, en el caso de `carve`, antes de ejecutar cada comando se comprueba si ya existe un modelo con ese nombre (generado en alguna ejecución previa de `modelado.R`), para evitar dedicar tiempo computacional a generar el mismo modelo dos o más veces. En el caso de `smetana` se comprueba antes de cada comando que no se haya generado anteriormente un análisis con el mismo nombre, por la misma razón.

`smetana` se encarga además de implementar o no un tercer análisis, el de los metabolitos acoplados al crecimiento, dependiendo de si la opción de `coupling` está activada.

La función `gram` es llamada desde `carve`. Esta función se encarga de, dado un filo taxonómico, devolver si es Gram-positivo o Gram-negativo. Así, `carve` utilizará el modelo universal más adecuado en cada caso. Se basa en dos archivos que contienen breves listas de filos Gram-positivos y Gram-negativos y que se encuentran en la carpeta original del *script*.

`check` es la función encargada de hacer la selección de modelos para la Estrategia 1. En primer lugar obtiene todas las posibles combinaciones de parejas entre el par de nodos dado. A continuación, recorre con un bucle todas las muestras del archivo del experimento dado y selecciona aquellas parejas que coinciden en al menos una.

`find_alignment_hits` es la función principal encargada del alineamiento, llamada en `modelado.R` y en `annotate.R`. Lo que hace es ejecutar `nucmer` para cada una de las hojas de cada nodo dado y aplica el filtro de calidad de identidad > 97% y cobertura (de la referencia/*hit*) > 90%. Devuelve un vector con el nombre de cada hoja (cuyo alineamiento haya superado el filtro de calidad) y su correspondiente *hit*. Este vector es guardado a modo de tabla en un archivo, “queries_v_hits”. También se generan los archivos “nucmer_unfiltered” y “nucmer_filtered”, con la salida de Nucmer antes y después del filtro de cobertura, y el archivo “genomes”, que consiste en una lista de todos los *hits* que han pasado el filtro. Este último es el tipo de archivo que se da como entrada a `annotate.R` usando `--genomes`.

Salida: -

Requiere: R (3.5.0), optparse, parallel. `nucmer` requiere el programa Nucmer (siendo válida cualquier versión; usamos la 3.1; MUMmer 3.23). `smetana` requiere Smetana (1.2.0). `carve` requiere CarveMe (1.4.0+). `emapper` requiere eggNOG-mapper (2.0.1).

RefrFBA.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Dec  4 12:24:06 2020

@author: Silvia Talavera Marcos
"""

from reframed import load_cbmodel
import os, sys
from reframed import FBA, Environment
from carveme.reconstruction.utils import load_media_db
import statistics as stats
import pandas as pd

# INPUT: RefrFBA.py input_folder/wd mediadb medium outputdir outputname

# IMPRIME la media y la desviación típica del crecimiento de todos los
#         modelos de la carpeta dada como entrada
# GENERA un archivo .csv con todas las tasas de crecimiento

wd = sys.argv[1]
mediadb = sys.argv[2]
medium = sys.argv[3]

if len(sys.argv) > 4:
    outputdir = sys.argv[4].rstrip("/") # si el usuario da el outputdir
```

```
    if len(sys.argv) > 5:
        outputname = sys.argv[5] # si el usuario da el nombre del output
    else:
        outputname = "report_reframed"+wd.split("/")[-1]+".csv"
else:
    outputdir = wd


models = []
for sbml in os.listdir(wd):
  if ".xml" in sbml:
    models.append(wd+"/"+sbml)

media_db = load_media_db(mediadb, compound_col="compound")

results = [] # lista de listas que finalmente dará un dataframe/csv con todos los datos

init_env = Environment.from_compounds(media_db[medium])
print("\nMedio: "+medium+"\n==============")
total=[]
n=0# para hacer la media de solution.fobj
for f in models:
    model = load_cbmodel(f,flavor="fbc2")
    init_env.apply(model)
    solution = FBA(model,   objective="Growth", get_values=False)
    if solution.fobj>0:
        n+=1
        total.append(solution.fobj)
    results.append([medium,f.split("/")[-1],solution.fobj])
try:
    media = sum(total)/n
    print("media del nodo: ", media)
    print("desviacion típica: ", stats.stdev(total),"\n")
except:
    pass

all_data=pd.DataFrame(results,columns=["Medio","Modelo","Crecimiento"])
all_data.to_csv(outputdir+"/"+outputname,index=False)
```

Uso: RefrFBA.py input_folder mediadb medium outputdir outputname

Los argumentos se indican sin *flags* en este *script*.

Argumentos:

- **`input_folder`**. Una carpeta que contenga todos los modelos SBML-FBC2 que se quiera analizar. Puede ser, por ejemplo, una de las carpetas generadas con `modelado.R`.
- **`mediadb`**. Base de datos de medios de cultivo a utilizar.
- **`medium`**. Medio en el que simular el crecimiento de los modelos dados.
- **`outputdir`**. Directorio de salida para el informe generado.
- **`outputname`**. Nombre del informe generado (incluyendo extensión *.csv*).

Descripción: En primer lugar se hacen los preparativos: a partir de la carpeta dada como `input_folder`, se crea una lista que contiene todos los modelos, filtrando por extensión *.xml*. También se carga el medio de cultivo y se inicializa como “entorno” con el método clase `Environment.from_compounds`, importado de CarveMe.

A continuación, se inicia un bucle en el que cada modelo de la lista será cargado con la función de CarveMe

`load_cbmodel`, se le aplicará el medio de cultivo y se incorporará a una simulación FBA con la función de ReFramed `FBA`. De cada simulación se guarda la tasa de crecimiento obtenida y, si es mayor que 0, se incorporará al cálculo de la tasa media de crecimiento para el total de los modelos. Todas las tasas de crecimiento se guardan en una lista de listas, `results`.

Por último, se imprime por la salida la tasa media de crecimiento y la desviación típica. Los datos de `results` se convierten a `DataFrame` de `pandas` y se guardan como *.csv* con el nombre indicado.

Salida: Archivo *.csv* con todas las tasas de crecimiento junto al nombre de cada modelo. Salida estándar: tasa media de crecimiento y desviación típica.

Requiere: Python (3.6+; en este Trabajo se usa 3.6.9). CarveMe (1.4.0+; incluye las funciones importadas, Pandas y ReFramed).

RefrFBA_E1.py

El *script* `RefrFBA.py` tiene una versión alternativa, `RefrFBA_E1.py`, que recibe como input una carpeta ligeramente distinta, con la siguiente estructura:

```
input_folder
├── cit
│   ├── Node28866
│   │   ├── 4370747.xml
│   │   └── (...)
│   └── Node35562
│       ├── 3944484.xml
│       └── (...)
├── glc
│   ├── Node27828
│   │   └── (...)
│   └── Node35562
│       └── (...)
└── leu
    ├── Node13821
    │   └── (...)
    └── Node28853
        └── (...)
```

`RefrFBA_E1` funciona de la misma manera, solo que el bucle va cambiando el medio especificado automáticamente para cada subcarpeta (M9[*cit*], M9[*glc*] o M9[*leu*]). El *.csv* final incluye todos los datos juntos, especificando el medio de cultivo y añadiendo también una columna con la tasa de crecimimento media. Se incluye el código en el desplegable a continuación.

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Dec  4 12:24:06 2020

@author: Silvia Talavera Marcos
"""

from reframed import load_cbmodel
import os
from reframed import FBA, Environment
from carveme.reconstruction.utils import load_media_db
import statistics as stats
import pandas as pd

wd = "/home/urihs/Desktop/TFM_private/08_reframed/sin_gapfill/"
```

```
models = {}
for medium in os.listdir(wd):
    models[medium] = {}
    for node in os.listdir(wd+medium):
        if "Node" in node:
            models[medium][node] = {}
        else:
            continue
        for sbml in os.listdir(wd+medium+"/"+node):
            if ".xml" in sbml:
                models[medium][node][sbml]=wd+medium+"/"+node+"/"+sbml

media_db = load_media_db("/home/urihs/Desktop/TFM_private/08_reframed/test_media.txt", c
ompound_col="compound")

results = [] # lista de listas que finalmente dará un dataframe/csv con todos los datos
for medium in models.keys():
    medio = "M9["+medium+"]"
    # medio = "LB"
    init_env = Environment.from_compounds(media_db[medio])
    print("\nMedio: "+medio+"\nNodos de: "+medium+"\n==============")
    media = 0
    listofkeys = list(models[medium].keys())
    listofkeys.sort()
    for node in listofkeys:
        media_prev = 0+media
        print("Nodo: "+node+"\n===============")
        total=[]
        n=0# para hacer la media de solution.fobj
        for f in models[medium][node]:
            model = load_cbmodel(models[medium][node][f],flavor="fbc2")
            init_env.apply(model)
            solution = FBA(model,   objective="Growth", get_values=False)
            print(solution)
            if solution.fobj!=0:
                n+=1
                total.append(solution.fobj)
            results.append([medium,node,f,solution.fobj])
        try:
            media = sum(total)/n
            print("media del nodo: ", media)
            print("desviacion típica: ", stats.stdev(total),"\n")
        except:
            pass
    try:
        print("RATIO : ",media_prev/media)
    except:
        pass

all_data=pd.DataFrame(results,columns=["Medio","Nodo","Modelo","Crecimiento"])
all_data.to_csv("report_reframed.csv",index=False)
```

parser.R

```
#!/usr/bin/env Rscript

library("optparse")
```

```
library("parallel")

# ==================
#     Take input
# ==================
### HELP ###
# parser.R -d/-g input_file -c cores --report_name reportname(can include path)

option_list <- list(
  make_option(c("-g", "--global"), type="character", default=NULL,
              help="Input folder with Smetana global results (e.g. ./my_results/NodeXXX
XX/smetana_results/global).", metavar="character"),
  make_option(c("-d", "--detailed"), type="character", default=NULL,
              help="Input folder with Smetana global results (e.g. ./my_results/NodeXXX
XX/smetana_results/global).", metavar="character"),
  make_option(c("-c","--cores"), type="numeric", default=4,
              help="Number of cores to use in parallelization", metavar = "character"),
  make_option(c("--report_name"), type="character", default="report.txt",
              help="Name of the output report file. Default: ./report.txt", metavar="ch
aracter")
)

parser <- OptionParser(option_list=option_list)
opt <- parse_args(parser)

global      <- opt$global
detailed    <- opt$detailed
cores       <- opt$cores
report_name <- opt$report_name

if (is.null(global)) {
  if (is.null(detailed)) {
    stop("Please specify an input folder with --global or --detailed.")
  } else {
    is.detailed <- TRUE
    is.global   <- FALSE
  }
} else if (is.null(detailed)) {
  is.global   <- TRUE
  is.detailed <- FALSE
} else {
  stop("Only --global or --detailed input is accepted. Please choose only one.")
}

if (is.global){
  # Open the dataset
  # ================
  filenames  <- list.files(global, pattern="*.tsv", full.names=TRUE)

  dataset    <- do.call("rbind", mclapply(filenames, FUN=function(filename)  {
      df <- read.table(filename, sep="\t", header=TRUE, na.strings = "n/a")
      rownames(df) <- tail(strsplit(filename,split="/")[[1]],n=1)
      return(df)
    } ,mc.cores=cores))


  # Report MRO=n/a files
  # ====================
  mro.is.na <- as.logical(is.na(dataset["mro"]))
```

```
  row_names  <- filenames
  not_growing <- row_names[mro.is.na]


  #  MRO values
  # =============
  growing <- dataset[!mro.is.na,]

  if (length(growing)==0) {
    write("MRO was n/a for all files. The co-culture can't grow.",file=report_name)
    stop ("MRO was n/a for all files. The co-culture can't grow.")
  } else {
    mean_mro <- mean(growing[,"mro"])
    min_mro  <-  min(growing[,"mro"])
    max_mro  <-  max(growing[,"mro"])
  }

  # MIP report
  # ==========
  mip.is.na <- as.logical(is.na(growing[,"mip"]))
  mip.is.not.na <- growing[!mip.is.na,]

  mean_mip <- mean(mip.is.not.na[,"mip"])
  min_mip  <-  min(mip.is.not.na[,"mip"])
  max_mip  <-  max(mip.is.not.na[,"mip"])


  # Print report
  # ============
  write("The following files have n/a MRO and MIP, which means they can't grow by themse
lves:\n",file=report_name)
  write(not_growing,file=report_name,append=TRUE)

  write("\n\nMRO (metabolic resource overlap) calculates how much the species compete fo
r the same metabolites", file=report_name, append=TRUE)
  write(paste("\nMean MRO:",   mean_mro, sep="\n"), file=report_name, append=TRUE)
  write(paste("\nMinimum MRO:", min_mro, sep="\n"), file=report_name, append=TRUE)
  write(paste("\nMaximum MRO:", max_mro, sep="\n"), file=report_name, append=TRUE)

  write("\n\nMIP (metabolic interaction potential) calculates how many metabolites the s
pecies can share to decrease their dependency on external resources", file=report_name,
append=TRUE)
  write(paste("\nMean MIP:",   mean_mip, sep="\n"), file=report_name, append=TRUE)
  write(paste("\nMinimum MIP:", min_mip, sep="\n"), file=report_name, append=TRUE)
  write(paste("\nMaximum MIP:", max_mip, sep="\n"), file=report_name, append=TRUE)

  write("\n\nFiles where MRO values are available:", file=report_name, append=TRUE)
  write.table(growing, file=report_name, sep="\t", quote=FALSE, append=TRUE)


} else {
  # Open the data files
  # ===================
  files_names   <- list.files(detailed, pattern="*.tsv", full.names=TRUE)

  files_text  <- mclapply(files_names, FUN=function(filename)  {
    read.csv(filename, sep="\t", header=TRUE)
  } ,mc.cores=cores)
```

```
  names(files_text) <- files_names


  # Report empty files
  # ==================
  is.empty  <- as.numeric(mclapply(files_text,FUN=function(x){length(x[,1])},mc.cores=co
res))==0

  empty     <- files_names[is.empty]
  not.empty <- files_names[!is.empty]


  # Analyze metabolites
  # ===================
  merged.files    <- do.call("rbind", files_text)
  rm(files_text) # free memory


  # Create node columns
  check_node      <- function(x,node_1_index,tags=c("first","second")){
    if (x %in% node_1_index) {result <- tags[1]} else {result <- tags[2]}
    return(result)
    }

  node_1_index    <- unique(sapply(files_names,FUN=function(filename){head(strsplit(file
name,split="_")[[1]],n=1)},USE.NAMES = FALSE))

  donor_node      <- sapply(merged.files[,"donor"], check_node, node_1_index, USE.NAMES
=FALSE)
  receiver_node   <- sapply(merged.files[,"receiver"],check_node, node_1_index, USE.NAME
S=FALSE)

  merged.files    <- cbind(merged.files, donor_node, receiver_node)

  # Exchanged in general
  all.metabolites <- setNames(aggregate(x=merged.files[,c(7,9)],by=list(merged.files$co
mpound),mean),
                              nm=c("compound","mus","smetana"))

  # Exchanged by node
  all.metabolites.by.node <- setNames(aggregate(x=merged.files[,c(7,9)],
                               by=list(merged.files$compound,merged.files$donor_node,
merged.files$receiver_node),
                               mean,),nm=c("compound","donor_node","receiver_node","mu
s","smetana"))

  # Ordered
  all.metabolites <- all.metabolites[order(all.metabolites$smetana,decreasing=TRUE),]
  all.metabolites.by.node <- all.metabolites.by.node[order(all.metabolites.by.node$smeta
na,decreasing=TRUE),]

  # Print report
  # ============
  write("The following files are empty, which means no exchange between species:\n",file
=report_name)
  write(empty,file=report_name,append=TRUE)
  write(paste0("\nThere are ",length(not.empty)," files which are not empty:"),file=repo
rt_name,append=TRUE)
  write(not.empty,file=report_name,append=TRUE)
```

```
  write("\n10 most exchanged metabolites:",file=report_name,append=TRUE)
  write.table(all.metabolites[1:10,],file=report_name,append=TRUE,quote=FALSE,sep="\t",r
ow.names=FALSE)

  write("\n10 most exchanged metabolites by donor node:",file=report_name,append=TRUE)
  write.table(all.metabolites.by.node[1:10,],file=report_name,append=TRUE,quote=FALSE,se
p="\t",row.names=FALSE)
  write("\n'first' node includes ",file=report_name,append=TRUE)
  write(node_1_index, file=report_name,append=TRUE)
  write("\nThe whole dataset is as follows:",file=report_name,append=TRUE)
  write.table(merged.files,file=report_name,append=TRUE,quote=FALSE,sep="\t",row.names=F
ALSE)
}
```

Uso: parser.R -d/-g input_file -c cores –report_name reportname

Argumentos:

- **`-g, --global`**. Carpeta que contenga informes del modo global de Smetana (e. g. ./my_results/NodeXXXXX/smetana_results/global).
- **`-d, --detailed`**. Carpeta que contenga informes del modo detallado de Smetana (e. g. ./my_results/NodeXXXXX/smetana_results/detailed).
- **`-c, --cores`**. Número de núcleos que usar en la paralelización.
- **`--report_name`**. Ruta y nombre del archivo de salida. Por defecto es `./report.txt`.

Descripción: Una vez leída la carpeta de entrada, se sigue un proceso distinto dependiendo de si es del modo global o del detallado.

Si es del modo global, primero se unen todos los archivos en un solo `data.frame`, leyéndolos con `mclapply` para mayor velocidad. Después se filtran y guardan (en `not_growing`) aquellas filas que contengan una MRO no disponible. Los nombres de las filas del `data.frame` coinciden con los de los modelos. Si la MRO es distinta de 0 en al menos un caso, el *script* continúa y obtiene el valor máximo, el medio y el mínimo de la MRO y la MIP. Finalmente se crea un informe que contiene una lista de aquellos archivos donde la MRO no estaba disponible, los valores de MRO y MIP y finalmente una lista completa de todos los resultados.

Si la carpeta dada es de resultados del modo detallado, primero se leen todos los archivos y se guardan en una lista. Esta lista se divide en dos una conteniendo los informes de Smetana vacíos y otra los que no están vacíos. Estos dos primeros pasos están paralelizados con `mclapply`.

Como los nombres predeterminados de los informes de Smetana (generados con `modelado.R`) incluyen los nombres de los dos modelos, uno de cada nodo, este *script* crea dos listas, `first` y `second`, que contienen los nombres de todos los modelos que pertenecen a cada uno de los nodos según su localización en el nombre del archivo. Esto se hace con la función `check_node`. A continuación se agrupan los metabolitos intercambiados de cada informe según el sentido en el que se intercambian, es decir, si van desde el nodo `first` hacia el `second` o viceversa (`all.metabolites.by.node`). También se crea una lista en la que se agrupan los metabolitos independientemente del sentido de intercambio (`all.metabolites`). Estas listas se ordenan de mayor a menor puntuación Smetana y se reportan en el informe final. También se incluye en el informe qué informes estaban vacíos.

Salida: Informe en formato de texto plano.

Requiere: R (3.5.0.), optparse, parallel

---

consenso.py

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Thu Nov 19 11:42:54 2020

@author: Silvia Talavera Marcos
"""

import os
import sys
import xml.etree.ElementTree as ET
import time, datetime

start = time.time()

# ================================================================================
# Leer todos los modelos
# ================================================================================
# INPUT: consenso.py input_folder output_folder outputname
# The models should not be gap-filled, as this modifies the fields
wd = sys.argv[1]
os.chdir(wd)

if len(sys.argv) > 2:
    outputdir = sys.argv[2].rstrip("/") # si el usuario da el outputdir

    if len(sys.argv) > 3:
        nodename = sys.argv[3] # si el usuario da el nombre del output
    else:
        nodename = wd.split("/")[-1]
else:
    outputdir = wd

models = [] # guardamos aquí los nombres de los archivos
for filename in os.listdir(wd):
    if ".xml" in filename:
        models.append(filename)


# ================================================================================
# Preparar el formato SBML
# ================================================================================
ET.register_namespace('', "http://www.sbml.org/sbml/level3/version1/core")
ET.register_namespace('fbc',"http://www.sbml.org/sbml/level3/version1/fbc/version2")

# ================================================================================
# Crear el "esqueleto" de nuestro modelo consenso. Usamos un modelo cualquiera.
# ================================================================================
try:
    first_model = ET.parse(models[0])
    consensus   = first_model
    cons_root   = consensus.getroot()
    cons_root[0].attrib['id'] = nodename
    cons_root[0][0][0][0].text = 'Description: This model is a consensus model from SBM
L FBC2 CarveMe output'
except:
    print("No se ha podido cargar ningún modelo.")

    raise(SystemExit(0))
```

```
# ================================================================================
# Hacer un conteo de cuántas veces aparece cada elemento en todos los
# modelos. *Evitamos cargar todos en memoria a la vez.*
# Nos quedaremos con aquellos que aparecen en el 80% o más de los modelos
# (4 de cada 5)
# ================================================================================

# Inauguramos una lista de reacciones de crecimiento (a partir de la cual obtendremos la
reacción de biomasa consenso)
growth = [first_model.getroot()[0][4][-2]]
cons_root[0][4].remove (cons_root[0][4][-2]) # y la eliminamos

# Hago una lista de diccionarios donde se hará el conteo
conteo = [{},{},{}]

# Los ID de cada elemento tienen diferente nombre según el campo
ID = ['id', 'id', '{http://www.sbml.org/sbml/level3/version1/fbc/version2}id']

# Inauguramos esta lista con el primer modelo
for n, campo in enumerate([2, 4, 6]):
    conteo[n] = {cons_root[0][campo][e].attrib[ID[n]]:1 for e in range(len(cons_root[0]
[campo]))}
    # <dicci>   <------------key------------------->
del(first_model)

# Vamos abriendo los demás modelos y contando
# Guardamos los elementos de los campos 2, 4 y 6 que no estaban ya
for m in models[1:]:
    model = ET.parse(m)
    model_root = model.getroot()
    # dejamos aparte la función de crecimiento
    growth.append(model_root[0][4][-2])
    model_root[0][4].remove(model_root[0][4][-2])

    for n, campo in enumerate([2, 4, 6]):
        for e in range(len(model_root[0][campo])):
            ident = model_root[0][campo][e].attrib[ID[n]]
            if ident not in conteo[n].keys():
                conteo[n][ident] = 1
                      #dic  #<-------key------------------------>
                cons_root[0][campo].append(model_root[0][campo][e])
            else:
                conteo[n][ident] += 1

    del(model) # !!! haciendo esto borro en memoria
    del(model_root)

# Filtro y elimino las entradas que están en menos de un 80% de modelos:
total = len(models)
for n, campo in enumerate([2,4,6]):
    removed = 0 # actualizamos el índice para evitar errores de indexación tras borrar
    for e in range(len(cons_root[0][campo])):
        ident = cons_root[0][campo][e-removed].attrib[ID[n]]
        if conteo[n][ident] < 0.80*total:
            cons_root[0][campo].remove (cons_root[0][campo][e-removed])
            removed+=1

# ================================================================================
# Por último añadimos la reacción de crecimiento que más se repite de la lista
```

```
# ================================================================================
raw_list = [tuple([tuple([str(i[n].attrib) for n in range(len(i))]) for i in item]) for
item in growth]

indexer={}
for index, i in enumerate(pure_list2):
    if i in myd.keys():
        pass
    else:
        indexer[i]=index

most_common = max(set(raw_list), key=pure_list.count) # Devuelve en formato tupla la fun
ción de crecimiento que más se repite
consensus_growth = growth[indexer[most_common]] # La tomamos en formato XML gracias a in
dexer
cons_root[0][4].append(consensus_growth)

# Guardamos el modelo
consensus.write(outputdir+"/"+nodename+"_consensus.xml", xml_declaration=True)

end = time.time()
print("Running time: ",str(datetime.timedelta(seconds = end - start)))
```

Uso: consenso.py input_folder output_folder outputname

Argumentos:

- `input_folder`. Una carpeta que contenga todos los modelos SBML-FBC2 a partir de los cuales se desee generar un consenso. Puede ser, por ejemplo, una de las carpetas generadas con `modelado.R`.
- `output_folder`. Carpeta de salida del consenso.
- `outputname`. Nombre del consenso (+ *consensus.xml*).

Descripción: Este *script* genera un modelo consenso en formato SBML-FBC2 a partir de archivos en el mismo formato. Está optimizado para modelos que no hayan sido sometidos a *gap-filling*, ya que esto modifica los campos del SBML. El formato SBML es un tipo de XML con los siguientes campos:

- root[0][0] –> notas del formato (notes)
- root[0][1] –> lista de compartimentos (listOfCompartments)
- **root[0][2] –> lista de compuestos (listOfSpecies)**
- root[0][3] –> lista de parámetros (listOfParameters)
- **root[0][4] –> lista de reacciones (listOfReactions)**
- root[0][5] –> lista de objetivos (listOfObjectives)
- **root[0][6] –> lista de productos génicos (listOfGeneProducts)**

Lo que hace este *script* es fijarse en los campos 2 (compuestos), 4 (reacciones) y 6 (productos génicos) e incluir en un archivo XML final (el consenso) todas las entradas de esos campos que estén en más de un 80% de modelos. Las reacciones del campo 4 es el que determina el crecimiento de cada modelo metabólico, mientras que el campo 6 es un campo propio del formato FBC2 que define productos génicos únicos para cada gen. Como los campos 2 y 6 dependen de las reacciones, también los tenemos en cuenta.

En primer lugar se crea un modelo XML vacío que hace de esqueleto del consenso. Este modelo está formado por todos los campos del primer modelo de la carpeta de entrada, excepto la reacción de biomasa (crecimiento) del campo 4. Paralelamente, creamos una variable (`growth`) donde se guardará, aparte, todas las reacciones de biomasa de todos los modelos con sus reactivos y coeficiente originales. Empezamos guardando la del primer modelo. También se crea una tercera variable, `conteo`, que es una

lista de diccionarios donde se apuntará cuántas veces aparece cada reactivo (campo 2), cada reacción que no sea la de crecimiento (campo 4) y cada producto génico (campo 6).

En segundo lugar, un bucle va recorriendo los demás modelos de la carpeta de entrada. A cada modelo abierto se le extrae la reacción de crecimiento y se guarda en `growth` y posteriormente se recorren sus campos 2, 4 y 6. Los campos 0 (descripción del modelo), 1, 3 y 5 no se cambian. Cada elemento `e` de cada campo (reactivo, reacción o producto génico en cada caso) se cuenta en el diccionario de `conteo`. Si no estaba ya incluido en el consenso, se incluye. Cada modelo se elimina de la memoria después de ser recorrido en el bucle.

Finalmente, un segundo bucle recorre los campos 2, 4 y 6 del consenso y elimina aquellas entradas cuyo número de apariciones en `conteo` sea menor del 80% del total de modelos. Por último, se selecciona la función de crecimiento más común entre todos los modelos y se añade al consenso final, que se guardará como *.xml*.

Salida: Modelo consenso en formato SBML-FBC2.

Requiere: Python (3.6+; en este Trabajo se usa 3.6.9).

---

consenso_EGG.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Nov 20 11:18:55 2020

@author: Silvia Talavera Marcos
"""

import os
import sys
from carveme.reconstruction.eggnog import load_eggnog_data
import datetime, time

start = time.time()

# ================================================================================
# Leer todos los modelos
# ================================================================================
# INPUT: consenso_EGG.py input_folder output_folder (outputname) (percentage)
# Only reads files that contain "annotations" in its name
# The path specified for the input_folder must be the realpath

wd = sys.argv[1].rstrip("/")

if len(sys.argv) > 2:
    outputdir = sys.argv[2].rstrip("/") # si el usuario da el outputdir

    if len(sys.argv) > 3:
        outputname = sys.argv[3].rstrip("/") # si el usuario da el nombre del output
        if len(sys.argv) > 4:
            perc = float(sys.argv[4]) # si el usuario da el porcentaje para filtrar
        else:
            perc = 0.80
    else:
        outputname = wd.split("/")[-1]


else:
```

```
    outputdir = wd

models = [] # guardamos aquí los nombres de los archivos

for filename in os.listdir(wd):
    if "annotations" in filename: # ignore seed orthologs files
        models.append(wd+"/"+filename)

if len(models) == 0:
    quit("The input file is empty.")
else:
    num_of_models = len(models)


# ================================================================================
# Crear un almacén de todas las reacciones a partir de un modelo cualquiera
# ================================================================================
all_reactions = load_eggnog_data(models[0],drop_unused_cols=False)

# Agrupamos por reacción para evitar repeticiones:
all_reactions = all_reactions.sort_values(by='score', ascending=False) \
                             .groupby('BiGG_gene', as_index=False).apply(lambda x: x.i
loc[0])
# Índice para acceder eficientemente a la fila de cada reacción:
reac2idx = {reac:idx for idx,reac in enumerate(all_reactions["BiGG_gene"])}
last_index = len(all_reactions)-1

# ================================================================================
#  Voy abriendo los demás y contando las apariciones de cada reacción
# ================================================================================

# Inicializamos el diccionario de conteo con las reacciones del primer modelo
conteo = {reac:1 for reac in all_reactions["BiGG_gene"]}
for model in models[1:]:
    # abrimos el modelo
    open_model = load_eggnog_data(model,drop_unused_cols=False)
    open_model = open_model.sort_values(by='score', ascending=False) \
                           .groupby('BiGG_gene', as_index=False).apply(lambda x: x.iloc
[0])

    for i, df_row in open_model.iterrows():
        reaction = df_row["BiGG_gene"]

        if reaction not in conteo.keys():
            conteo[reaction] = 1          # la contamos
            all_reactions = all_reactions.append(df_row,ignore_index=True) # la añadimo
s al almacén
            last_index += 1               # y anotamos su índice
            reac2idx[reaction] = last_index


        else:
            conteo[reaction] += 1         # la contamos
            # puntuación nueva ponderada (el resultado es la media para esa reacción):
            old_score = all_reactions.at[reac2idx[reaction],"score"]
            new_score = ( old_score *(conteo[reaction]-1) + df_row["score"] ) /conteo[r
eaction]
            all_reactions.at[reac2idx[reaction],"score"] = new_score # actualizar
```

```
    # cerramos el modelo (lo quitamos de memoria)
    del(open_model)


# ================================================================================
# Filtro y elimino las reacciones que están en menos de un 80% de modelos:
# ================================================================================
for r in reac2idx.keys():
    if conteo[r] < int(perc*num_of_models):
        all_reactions = all_reactions.drop(reac2idx[r],axis=0)
# ================================================================================
# Guardo el archivo de anotaciones consenso
# ================================================================================
if not os.path.exists(outputdir):
    os.mkdir(outputdir)
f = open(outputdir+"/"+outputname+".tsv","a+")

f.write("# emapper version: emapper-2.0.1 emapper DB: 2.0\n")
f.write("# consensus annotation file for "+outputname+"\n")
f.write("# time: "+str(datetime.datetime.now())+"\n")
f.write("#query_name    seed_eggNOG_ortholog    seed_ortholog_evalue    seed_ortholog_s
core best_tax_level  Preferred_name  GOs EC  KEGG_ko KEGG_Pathway    KEGG_Module KEGG_R
eaction   KEGG_rclass BRITE   KEGG_TC CAZy    BiGG_Reaction   taxonomic scope eggNOG OG
s  best eggNOG OG  COG Functional cat. eggNOG free text desc.\n")
all_reactions.to_csv(f,sep=" ",header=False, index=False)

f.close()

end = time.time()
print("Running time: ",end - start)
```

Uso: consenso_EGG.py input_folder output_folder (outputname) (percentage)

Argumentos:

- **`input_folder`**. Una carpeta que contenga todos los archivos de anotaciones de eggNOG-mapper a partir de los cuales se desee generar un consenso. Los archivos de anotaciones deben contener *annotations* en el nombre.
- **`output_folder`**. Carpeta de salida del consenso.
- **`outputname`**. Nombre del consenso (+ *.tsv*).
- **`percentage`**. Porcentaje de archivos en los que debe estar presente una anotación funcional para ser incluida en el consenso final. Por defecto es 80%. Debe indicarse con el formato “0.80”.

Descripción: Este *script* genera un archivo consenso de anotaciones funcionales a partir de una carpeta dada que contenga múltiples archivos de anotaciones funcionales, obtenidos con eggNOG-mapper. El formato de estos archivos es variable según las opciones indicadas a eggNOG-mapper; el formato compatible en este caso es aquel que se genera con *annotate.R* y que CarveMe es capaz de leer con su opción `--egg` para generar modelos.

En primer lugar, se lee la carpeta de entrada. Si hay al menos un modelo con *annotations* en el título, el *script* continúa. En segundo lugar, se inicializa con la función `load_eggnog_data` u n `DataFrame` de `pandas`, llamado `all_reactions`, a partir del primer modelo de la lista. También se inicializa un diccionario de conteo de reacciones.

`all_reactions` incluirá todas las reacciones diferentes que aparezcan en los modelos de la carpeta de entrada y se llena en un bucle que recorre los demás modelos. Se abre cada modelo, se seleccionan sus

reacciones únicas y se contabilizan en `conteo`. Como cada reacción tiene una puntuación asignada, utilizada por CarveMe para hacer modelos, cada vez que se repite una reacción se actualiza su puntuación haciendo una media ponderada. También se van anotando los índices de cada reacción para facilitar los pasos posteriores. Tras cada iteración, se elimina de memoria el último modelo abierto.

Por último, se aplica el filtro: se eliminan de `all_reactions` aquellas reacciones que no aparezcan en al menos el porcentaje especificado de archivos de anotaciones originales. La lista filtrada se guarda como `.tsv`. Posteriormente esta lista de anotaciones consensuada podrá ser utilizada por CarveMe para generar un modelo consenso.

Salida: Archivo de anotaciones funcionales *.tsv*.

Requiere: Python (3.6+; en este Trabajo se usa 3.6.9), CarveMe (1.4.0+, incluye pandas).