Supplementary Text S1: Assessing the completeness of Chlorophyta genomes

Fabricio Almeida-Silva[1,2] ***and Yves Van de Peer***[1,2,3,4]

[1]VIB-UGent Center for Plant Systems Biology, Ghent, Belgium
[2]Department of Plant Biotechnology and Bioinformatics, Ghent University, Ghent, Belgium
[3]College of Horticulture, Academy for Advanced Interdisciplinary Studies, Nanjing Agricultural University, Nanjing, China
[4]Center for Microbial Ecology and Genomics, Department of Biochemistry, Genetics and Microbiology, University of Pretoria, Pretoria, South Africa

2 February 2023

Contents

1 Overview . . . 2
2 Managing external dependencies with virtual environments . . . 2
3 Data acquisition . . . 2
4 Running BUSCO . . . 3
5 Visualizing summary statistics . . . 4
Session info . . . 6
References . . . 9

```r
library(here)
library(cogeqc)
library(tidyverse)
library(Herper)

set.seed(123) # for reproducibility
options(timeout = 6000) # to load files from the web
```

1 Overview

Here, we will use *cogeqc* to assess the completeness of Chlorophyta genomes available on Pico-PLAZA 3.0 (Van Bel et al. 2018) using Best Universal Single-Copy Orthologs (BUSCOs).

2 Managing external dependencies with virtual environments

Here, for convenience, we will install BUSCO in a Conda environment for use with *cogeqc*. For that, we will use the Bioconductor package *Herper*

Below, you can find the code to install miniconda in a directory of your choice (here, "~/Documents") and create a virtual environment containing a BUSCO installation.

```r
# Path to where BUSCO will be installed and env name
my_miniconda <- file.path("~/Documents", "miniconda")
env <- "cogeqc_env"

# Create env named `cogeqc_env` with BUSCO in it
install_CondaTools(
    tools = "busco==5.3.0",
    env = env,
    channels = c("conda-forge", "bioconda"),
    pathToMiniConda = my_miniconda
)
```

3 Data acquisition

Now, we will load all genomes directly from PLAZA as `DNAStringSet` objects and export them to a single directory of FASTA files, so we can run BUSCO in batch mode.

```r
# Links to Chlorophyta genomes from Pico-PLAZA 3.0
base_url <- "ftp://ftp.psb.ugent.be/pub/plaza/plaza_pico_03/Genomes/"
links <- paste0(
    base_url,
    c("mpu.fasta.gz", "mrcc299.fasta.gz", "olu.fasta.gz", "ome.fasta.gz",
      "orcc809.fasta.gz", "ota.fasta.gz", "bprrcc1105.fasta.gz",
      "cre.fasta.gz", "vca.fasta.gz", "cvu.fasta.gz", "acg.fasta.gz",
      "pse3.fasta.gz", "prcc4223.fasta.gz", "cnc64a.fasta.gz",
      "hsp.fasta.gz", "apr.fasta.gz")
)

# Load all genomes
genomes <- lapply(links, Biostrings::readDNAStringSet)
names(genomes) <- basename(links)

# Write all genomes to a subdirectory of tempdir
genomes_path <- file.path(tempdir(), "genomes")
if(!dir.exists(genomes_path)) { fs::dir_create(genomes_path) }

write <- lapply(seq_along(genomes), function(x) {
    Biostrings::writeXStringSet(
        x = genomes[[x]],
        filepath = file.path(genomes_path, names(genomes)[x])
    )
    return(NULL)
})

dir(genomes_path)
##  [1] "acg.fasta.gz"       "apr.fasta.gz"       "bprrcc1105.fasta.gz"
##  [4] "cnc64a.fasta.gz"    "cre.fasta.gz"       "cvu.fasta.gz"
##  [7] "hsp.fasta.gz"       "mpu.fasta.gz"       "mrcc299.fasta.gz"
## [10] "olu.fasta.gz"       "ome.fasta.gz"       "orcc809.fasta.gz"
## [13] "ota.fasta.gz"       "prcc4223.fasta.gz"  "pse3.fasta.gz"
## [16] "vca.fasta.gz"
```

4 Running BUSCO

Now that all genomes are stored as FASTA files in /tmp/RtmpcuLMqJ/genomes, we can assess their completeness with BUSCO.

```
# See all possible lineage datasets
with_CondaEnv(
    env, list_busco_datasets(), my_miniconda
)

# Run BUSCO using chlorophyta_odb10 as the lineage data set
busco <- with_CondaEnv(
    env,
    run_busco(
        sequence = genomes_path,
        outlabel = "chlorophyta_busco",
        mode = "genome",
        lineage = "chlorophyta_odb10",
        outpath = tempdir(),
        download_path = tempdir()
    ),
    my_miniconda
)
```

```
# Read and parse the output
outdir <- file.path(tempdir(), "chlorophyta_busco")
busco_summary <- read_busco(outdir)
save(
    busco_summary,
    file = here::here("products", "result_files", "busco_summary.rda"),
    compress = "xz"
)
```

The parsed BUSCO output (as returned by `read_busco()`) looks like this:

```
load(here("products", "result_files", "busco_summary.rda"))
head(busco_summary)
##         Class Frequency           Lineage             File
## 1 Complete_SC      94.1 chlorophyta_odb10    pse3.fasta.gz
## 2 Complete_SC      95.1 chlorophyta_odb10     cre.fasta.gz
## 3 Complete_SC      96.8 chlorophyta_odb10     olu.fasta.gz
## 4 Complete_SC      98.7 chlorophyta_odb10 mrcc299.fasta.gz
## 5 Complete_SC      91.8 chlorophyta_odb10     apr.fasta.gz
## 6 Complete_SC      86.4 chlorophyta_odb10     acg.fasta.gz
```

5 Visualizing summary statistics

Finally, let's visualize summary BUSCO stats:

```
# Manually create tree based on Pico-PLAZA's tree
c_branches <- function(b1, b2) {
    x <- paste0("(", b1, ",", b2, ")")
}

ostreococcus_root <- "((((Ostreococcus_lucimarinus, Ostreococcus_sp_RCC809), Ostreococcus_tauri), Ostreococcu
micromonas <- "(Micromonas_pusilla_strain_CCMP1545, Micromonas_sp_RCC299)"
chlamydomonadales <- "(Volvox_carteri, Chlamydomonas_reinhardtii)"
picochlorum <- "(Picochlorum_sp_SENEW3, Picochlorum_RCC4223)"
chlorellales <- "((Helicosporidium_sp, Auxenochlorella_protothecoides), Chlorella_sp_NC64A)"
trebouxiophyceae <- c_branches(
    "(Coccomyxa_subellipsoidea_C-169, Asterochloris_sp_Cgr/DA1pho_v2)",
    c_branches(picochlorum, chlorellales)
)

chlo_tree <- c_branches(
    c_branches(
        ostreococcus_root, micromonas
    ),
    c_branches(
        chlamydomonadales, trebouxiophyceae
    )
)
chlo_tree <- paste0(chlo_tree, ";")
```

```r
# Read tree as a phylo object and clean species names
chlo_tree <- treeio::read.tree(text = chlo_tree)
chlo_tree$tip.label <- gsub("_", " ", chlo_tree$tip.label)

# Plot species tree and get species order from tree topology
p_tree <- plot_species_tree(chlo_tree, xlim = c(0, 12))
taxa_order <- rev(ggtree::get_taxa_name(p_tree))

# Plot BUSCO summary stats
p_busco <- busco_summary %>%
    mutate(File = str_replace_all(File, "\\.fasta.*", "")) %>%
    mutate(File = str_replace_all(
        File,
        c(
            "pse3" = "Picochlorum sp SENEW3",
            "cre" = "Chlamydomonas reinhardtii",
            "olu" = "Ostreococcus lucimarinus",
            "mrcc299" = "Micromonas sp RCC299",
            "apr" = "Auxenochlorella protothecoides",
            "acg" = "Asterochloris sp Cgr/DA1pho v2",
            "cvu" = "Coccomyxa subellipsoidea C-169",
            "bprrcc1105" = "Bathycoccus prasinos",
            "orcc809" = "Ostreococcus sp RCC809",
            "prcc4223" = "Picochlorum RCC4223",
            "ota" = "Ostreococcus tauri",
            "hsp" = "Helicosporidium sp",
            "mpu" = "Micromonas pusilla strain CCMP1545",
            "vca" = "Volvox carteri",
            "ome" = "Ostreococcus mediterraneus",
            "cnc64a" = "Chlorella sp NC64A"
        )
    )) %>%
    mutate(File = factor(File, taxa_order)) %>%
    plot_busco() +
    theme(axis.text.y = element_blank()) +
    labs(y = "")

# Combining phylogeny with BUSCO plot
combined <- patchwork::wrap_plots(p_tree, p_busco)
combined
```

Except for *Helicosporidium sp.*, Chlorophyta genomes on Pico-PLAZA 3.0 have a high quality, as denoted by their high completeness.

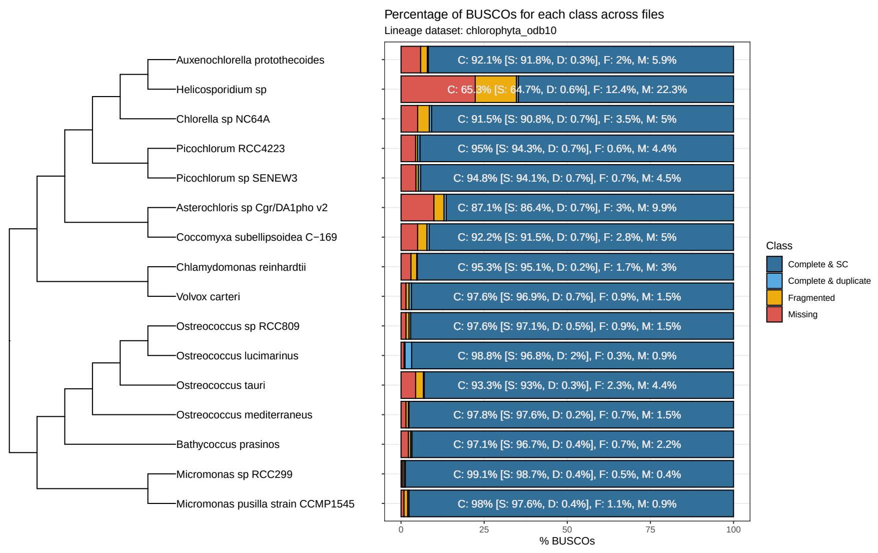


Figure 1: BUSCO scores for Chlorophyta genomes on Pico-PLAZA 3.0.

Session info

This document was created under the following conditions:

```
sessioninfo::session_info()
## - Session info ---------------------------------------------------------------
##  setting  value
##  version  R version 4.2.2 Patched (2022-11-10 r83330)
##  os       Ubuntu 20.04.5 LTS
##  system   x86_64, linux-gnu
##  ui       X11
##  language (EN)
##  collate  en_US.UTF-8
##  ctype    en_US.UTF-8
##  tz       Europe/Brussels
##  date     2023-02-02
##  pandoc   2.19.2 @ /usr/lib/rstudio/resources/app/bin/quarto/bin/tools/ (via rmarkdown)
##
## - Packages -------------------------------------------------------------------
##  package          * version    date (UTC) lib source
##  ape                5.6-2      2022-03-02 [1] CRAN (R 4.2.0)
##  aplot              0.1.8      2022-10-09 [1] CRAN (R 4.2.1)
##  assertthat         0.2.1      2019-03-21 [1] CRAN (R 4.2.0)
##  backports          1.4.1      2021-12-13 [1] CRAN (R 4.2.0)
##  beeswarm           0.4.0      2021-06-01 [1] CRAN (R 4.2.2)
##  BiocGenerics       0.42.0     2022-04-26 [1] Bioconductor
##  BiocManager        1.30.18    2022-05-18 [1] CRAN (R 4.2.0)
##  BiocStyle        * 2.25.0     2022-06-15 [1] Github (Bioconductor/BiocStyle@7150c28)
##  Biostrings         2.64.1     2022-08-18 [1] Bioconductor
##  bitops             1.0-7      2021-04-24 [1] CRAN (R 4.2.0)
##  bookdown           0.29      2022-09-12 [1] CRAN (R 4.2.1)
##  broom              1.0.1     2022-08-29 [1] CRAN (R 4.2.1)
##  cellranger         1.1.0     2016-07-27 [1] CRAN (R 4.2.0)
##  cli                3.4.1     2022-09-23 [1] CRAN (R 4.2.1)
##  cogeqc           * 1.3.1     2023-01-24 [1] Bioconductor
##  colorspace         2.0-3     2022-02-21 [1] CRAN (R 4.2.0)
##  crayon             1.5.2     2022-09-29 [1] CRAN (R 4.2.1)
##  DBI                1.1.3     2022-06-18 [1] CRAN (R 4.2.0)
##  dbplyr             2.2.1     2022-06-27 [1] CRAN (R 4.2.1)
##  digest             0.6.29    2021-12-01 [1] CRAN (R 4.2.0)
##  dplyr            * 1.0.10    2022-09-01 [1] CRAN (R 4.2.1)
##  ellipsis           0.3.2     2021-04-29 [1] CRAN (R 4.2.0)
##  evaluate           0.17      2022-10-07 [1] CRAN (R 4.2.1)
##  fansi              1.0.3     2022-03-24 [1] CRAN (R 4.2.0)
##  farver             2.1.1     2022-07-06 [1] CRAN (R 4.2.1)
##  fastmap            1.1.0     2021-01-25 [1] CRAN (R 4.2.0)
##  forcats          * 0.5.2     2022-08-19 [1] CRAN (R 4.2.1)
##  fs                 1.5.2     2021-12-08 [1] CRAN (R 4.2.0)
##  gargle             1.2.1     2022-09-08 [1] CRAN (R 4.2.1)
##  generics           0.1.3     2022-07-05 [1] CRAN (R 4.2.1)
##  GenomeInfoDb       1.32.4    2022-09-06 [1] Bioconductor
##  GenomeInfoDbData   1.2.8     2022-05-06 [1] Bioconductor
##  ggbeeswarm         0.7.1     2022-12-16 [1] CRAN (R 4.2.2)
##  ggfun              0.0.8     2022-11-07 [1] CRAN (R 4.2.1)
##  ggplot2          * 3.4.0     2022-11-04 [1] CRAN (R 4.2.1)
##  ggplotify          0.1.0     2021-09-02 [1] CRAN (R 4.2.0)
##  ggtree             3.7.1.001 2022-11-10 [1] Github (YuLab-SMU/ggtree@b7ef83e)
##  glue               1.6.2     2022-02-24 [1] CRAN (R 4.2.0)
##  googledrive        2.0.0     2021-07-08 [1] CRAN (R 4.2.0)
##  googlesheets4      1.0.1     2022-08-13 [1] CRAN (R 4.2.1)
##  gridGraphics       0.5-1     2020-12-13 [1] CRAN (R 4.2.0)
##  gtable             0.3.1     2022-09-01 [1] CRAN (R 4.2.1)
##  haven              2.5.1     2022-08-22 [1] CRAN (R 4.2.1)
##  here             * 1.0.1     2020-12-13 [1] CRAN (R 4.2.0)
##  Herper           * 1.1.2     2022-05-18 [1] Github (RockefellerUniversity/Herper@5bceeb4)
##  hms                1.1.2     2022-08-19 [1] CRAN (R 4.2.1)
##  htmltools          0.5.3     2022-07-18 [1] CRAN (R 4.2.1)
##  httr               1.4.4     2022-08-17 [1] CRAN (R 4.2.1)
##  igraph             1.3.5     2022-09-22 [1] CRAN (R 4.2.1)
##  IRanges            2.30.1    2022-08-18 [1] Bioconductor
##  jsonlite           1.8.3     2022-10-21 [1] CRAN (R 4.2.1)
##  knitr              1.40      2022-08-24 [1] CRAN (R 4.2.1)
##  labeling           0.4.2     2020-10-20 [1] CRAN (R 4.2.0)
##  lattice            0.20-45   2021-09-22 [1] CRAN (R 4.2.0)
##  lazyeval           0.2.2     2019-03-15 [1] CRAN (R 4.2.0)
##  lifecycle          1.0.3     2022-10-07 [1] CRAN (R 4.2.1)
##  lubridate          1.8.0     2021-10-07 [1] CRAN (R 4.2.0)
##  magrittr           2.0.3     2022-03-30 [1] CRAN (R 4.2.0)
##  Matrix             1.5-1     2022-09-13 [1] CRAN (R 4.2.1)
##  modelr             0.1.9     2022-08-19 [1] CRAN (R 4.2.1)
##  munsell            0.5.0     2018-06-12 [1] CRAN (R 4.2.0)
## nlme           3.1-160  2022-10-10 [1] CRAN (R 4.2.1)
## patchwork      1.1.2    2022-08-19 [1] CRAN (R 4.2.1)
## pillar         1.8.1    2022-08-19 [1] CRAN (R 4.2.1)
## pkgconfig      2.0.3    2019-09-22 [1] CRAN (R 4.2.0)
## plyr           1.8.7    2022-03-24 [1] CRAN (R 4.2.0)
## png            0.1-7    2013-12-03 [1] CRAN (R 4.2.0)
## purrr        * 0.3.5    2022-10-06 [1] CRAN (R 4.2.1)
## R6             2.5.1    2021-08-19 [1] CRAN (R 4.2.0)
## Rcpp           1.0.9    2022-07-08 [1] CRAN (R 4.2.1)
## RCurl          1.98-1.9 2022-10-03 [1] CRAN (R 4.2.1)
## readr        * 2.1.3    2022-10-01 [1] CRAN (R 4.2.1)
## readxl         1.4.1    2022-08-17 [1] CRAN (R 4.2.1)
## reprex         2.0.2    2022-08-17 [1] CRAN (R 4.2.1)
## reshape2       1.4.4    2020-04-09 [1] CRAN (R 4.2.0)
## reticulate   * 1.26     2022-08-31 [1] CRAN (R 4.2.1)
## rjson          0.2.21   2022-01-09 [1] CRAN (R 4.2.0)
## rlang          1.0.6    2022-09-24 [1] CRAN (R 4.2.1)
## rmarkdown      2.17     2022-10-07 [1] CRAN (R 4.2.1)
## rprojroot      2.0.3    2022-04-02 [1] CRAN (R 4.2.0)
## rstudioapi     0.14     2022-08-22 [1] CRAN (R 4.2.1)
## rvest          1.0.3    2022-08-19 [1] CRAN (R 4.2.1)
## S4Vectors      0.34.0   2022-04-26 [1] Bioconductor
## scales         1.2.1    2022-08-20 [1] CRAN (R 4.2.1)
## sessioninfo    1.2.2    2021-12-06 [1] CRAN (R 4.2.0)
## stringi        1.7.8    2022-07-11 [1] CRAN (R 4.2.1)
## stringr      * 1.4.1    2022-08-20 [1] CRAN (R 4.2.1)
## tibble       * 3.1.8    2022-07-22 [1] CRAN (R 4.2.1)
## tidyr        * 1.2.1    2022-09-08 [1] CRAN (R 4.2.1)
## tidyselect     1.2.0    2022-10-10 [1] CRAN (R 4.2.1)
## tidytree       0.4.1    2022-09-26 [1] CRAN (R 4.2.1)
## tidyverse    * 1.3.2    2022-07-18 [1] CRAN (R 4.2.1)
## treeio         1.23.0   2022-11-10 [1] Github (GuangchuangYu/treeio@db85803)
## tzdb           0.3.0    2022-03-28 [1] CRAN (R 4.2.0)
## utf8           1.2.2    2021-07-24 [1] CRAN (R 4.2.0)
## vctrs          0.5.0    2022-10-22 [1] CRAN (R 4.2.1)
## vipor          0.4.5    2017-03-22 [1] CRAN (R 4.2.1)
## withr          2.5.0    2022-03-03 [1] CRAN (R 4.2.0)
## xfun           0.33     2022-09-12 [1] CRAN (R 4.2.1)
## xml2           1.3.3    2021-11-30 [1] CRAN (R 4.2.0)
## XVector        0.36.0   2022-04-26 [1] Bioconductor
## yaml           2.3.5    2022-02-21 [1] CRAN (R 4.2.0)
## yulab.utils    0.0.5    2022-06-30 [1] CRAN (R 4.2.1)
## zlibbioc       1.42.0   2022-04-26 [1] Bioconductor
##
## [1] /home/faalm/R/x86_64-pc-linux-gnu-library/4.2
## [2] /usr/local/lib/R/site-library
## [3] /usr/lib/R/site-library
## [4] /usr/lib/R/library
##
## ------------------------------------------------------------------------------
```

References

Van Bel, Michiel, Tim Diels, Emmelien Vancaester, Lukasz Kreft, Alexander Botzki, Yves Van de Peer, Frederik Coppens, and Klaas Vandepoele. 2018. "PLAZA 4.0: An Integrative Resource for Functional, Evolutionary and Comparative Plant Genomics." *Nucleic Acids Research* 46 (D1): D1190–96.

Supplementary Text S2: Assessing Orthobench orthogroups

Fabricio Almeida-Silva[1,2] ***and Yves Van de Peer***[1,2,3,4]

[1]VIB-UGent Center for Plant Systems Biology, Ghent, Belgium
[2]Department of Plant Biotechnology and Bioinformatics, Ghent University, Ghent, Belgium
[3]College of Horticulture, Academy for Advanced Interdisciplinary Studies, Nanjing Agricultural University, Nanjing, China
[4]Center for Microbial Ecology and Genomics, Department of Biochemistry, Genetics and Microbiology, University of Pretoria, Pretoria, South Africa

7 August 2023

Contents

1 Overview 2
2 Data acquisition 2
3 Orthogroup assessment 4
Session info 5
References 8

1 Overview

Here, we will assess the orthogroups from the Orthobench data set (Trachana et al. 2011; Emms and Kelly 2020) using orthogroup scores.

```
set.seed(123)
options(timeout = 6000)

library(cogeqc)
library(tidyverse)
library(httr)
library(here)
library(biomartr)
```

2 Data acquisition

The data set can be obtained in this GitHub repository. First, let's create a data frame containing all gene IDs for each species.

```
# Repo's URL
url <- "https://api.github.com/repos/davidemms/Open_Orthobench/git/trees/master?recursive=2"

# Get all file paths
paths <- GET(url) |> content()

# Get only file paths for sequence files
paths <- unlist(lapply(paths$tree, function(x) x$path))
sequence_paths <- data.frame(
    Path = paths
) |>
    separate(Path, c("base", "folder", "filename"), "/") |>
    filter(folder == "Input") |>
    filter(str_detect(filename, "\\.fa")) |>
    mutate(
        download_path = file.path(
            "https://github.com/davidemms/Open_Orthobench/raw/master",
            base, folder, filename
        )
    )

# Create a data frame of gene IDs per species
genes_per_species <- Reduce(rbind, lapply(seq_len(nrow(sequence_paths)), function(x) {

    species <- gsub(".pep.*", "", sequence_paths$filename[x])
    gene_ids <- names(
        Biostrings::readAAStringSet(sequence_paths$download_path[x])
    )

    species_and_genes <- data.frame(
        Species = species,
        Gene = gene_ids
    )

    return(species_and_genes)
}))
```

Now, we will get all reference orthogroups from Orthobench and reshape them so that they look like the standard orthogroup data frame in **cogeqc** (with columns `Orthogroup`, `Species`, and `Gene`).

```
# Get URL to each orthogroup
og_paths <- data.frame(
    Path = paths
) |>
    separate(Path, c("base", "folder", "filename"), "/") |>
    filter(folder == "RefOGs") |>
    filter(str_detect(filename, "\\.txt")) |>
    mutate(
        download_path = file.path(
            "https://github.com/davidemms/Open_Orthobench/raw/master",
            base, folder, filename
        )
    )

# Read orthogroups and reformat them as cogeqc's orthogroup data frame
reference_ogs <- Reduce(rbind, lapply(seq_len(nrow(og_paths)), function(x) {

    og_name <- gsub(".txt", "", og_paths$filename[x])
    og_genes <- readLines(og_paths$download_path[x])

    og_df <- data.frame(
        Orthogroup = og_name,
        Gene = og_genes
    ) |>
        left_join(genes_per_species) |>
        dplyr::select(Orthogroup, Species, Gene)

    return(og_df)
}))
```

Finally, we will use the biomartr package (Drost and Paszkowski 2017) to obtain protein domain annotation for each species from Ensembl.

```
annotation_list <- lapply(unique(reference_ogs$Species), function(x) {

    species_id <- paste0(
        tolower(substr(x, 1, 1)), # first letter of genus
        gsub(".*_", "", gsub("\\..*", "", x)) # entire specific epithet
    )

    if(startsWith(x, "Canis")) {
        species_id <- "clfamiliaris"
    }

    genes <- reference_ogs |>
        filter(Species == x) |>
        pull(Gene)

    annot <- biomart(
        genes = genes,
        mart = "ENSEMBL_MART_ENSEMBL",
        dataset = paste0(species_id, "_gene_ensembl"),
        attributes = "interpro",
        filters = "ensembl_peptide_id"
    ) |>
        dplyr::select(Gene = ensembl_peptide_id, Annotation = interpro)

    return(annot)
})
names(annotation_list) <- unique(reference_ogs$Species)

# Remove empty elements (i.e., species for which we could not obtain annotation)
empty <- sapply(annotation_list, nrow) == 0
annotation_list <- annotation_list[!empty]
```

3 Orthogroup assessment

Now that we have the orthogroup data frame and the annotation list, we can calculate homogeneity scores.

```
p_orthobench_homogeneity <- reference_ogs |>
    ## Remove species for which we could not obtain domain annotation
    filter(Species %in% names(annotation_list)) |>
    ## Calculate homogeneity scores
    left_join(Reduce(rbind, annotation_list)) |>
    calculate_H(correct_overclustering = FALSE) |>
    dplyr::filter(!is.na(Score)) |>
    ## Plot a histogram of scores
    ggplot(aes(x = Score, y = ..count../sum(..count..))) +
    geom_histogram(fill = "grey60", color = "black", bins = 60) +
    labs(
        title = "Homogeneity scores for Orthobench's reference orthogroups",
        y = "Relative frequency", x = "Homogeneity scores"
    ) +
    xlim(0, 1) +
    theme_bw()

p_orthobench_homogeneity
```

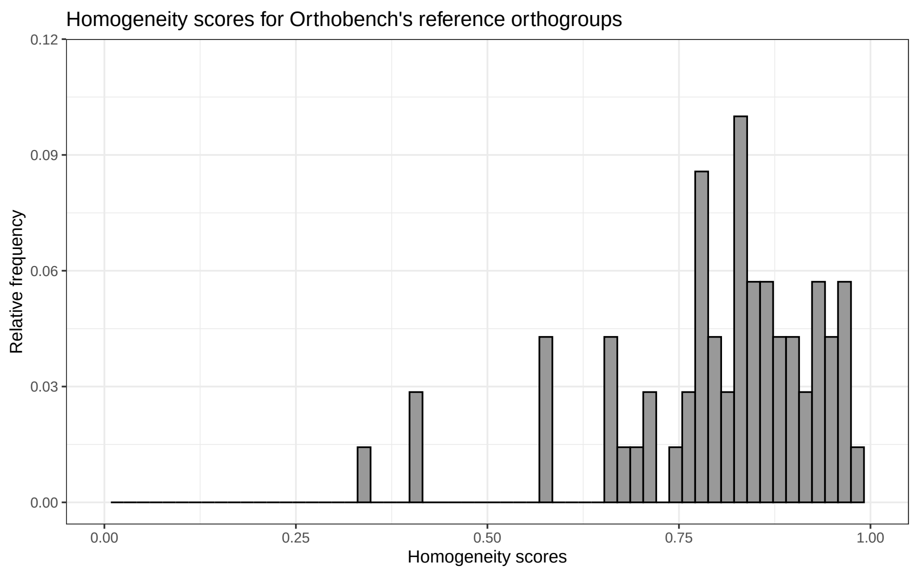


Figure 1: Homogeneity scores for Orthobench's reference orthogroups.

The plot shows that homogeneity scores for reference orthogroups tend to be very close to 1, as expected, which validates the rationale behind our approach. Of note, most orthogroups do not reach perfect homogeneity, probably due to domain gains and losses throughout their evolution, but they are still very close to 1. In summary, our findings here demonstrate that reference-quality orthogroups should indeed have homogeneity scores as close to 1 as possible, and users should seek a similar distribution when inferring orthogroups for their own data sets.

Session info

This document was created under the following conditions:

```
## - Session info --------------------------------------------------------------
##  setting  value
##  version  R version 4.3.0 (2023-04-21)
##  os       Ubuntu 20.04.5 LTS
##  system   x86_64, linux-gnu
##  ui       X11
##  language (EN)
##  collate  en_US.UTF-8
##  ctype    en_US.UTF-8
##  tz       Europe/Brussels
##  date     2023-08-07
##  pandoc   3.1.1 @ /usr/lib/rstudio/resources/app/bin/quarto/bin/tools/ (via rmarkdown)
##
## - Packages -------------------------------------------------------------------
##  package          * version   date (UTC) lib source
##  AnnotationDbi      1.62.0    2023-04-25 [1] Bioconductor
##  ape                5.7-1     2023-03-13 [1] CRAN (R 4.3.0)
##  aplot              0.1.10    2023-03-08 [1] CRAN (R 4.3.0)
##  beeswarm           0.4.0     2021-06-01 [1] CRAN (R 4.3.0)
##  Biobase            2.60.0    2023-04-25 [1] Bioconductor
##  BiocFileCache      2.8.0     2023-04-25 [1] Bioconductor
##  BiocGenerics       0.46.0    2023-04-25 [1] Bioconductor
##  BiocManager        1.30.21.1 2023-07-18 [1] CRAN (R 4.3.0)
##  BiocStyle        * 2.29.1    2023-08-04 [1] Github (Bioconductor/BiocStyle@7c0e093)
##  biomaRt            2.56.0    2023-04-25 [1] Bioconductor
##  biomartr         * 1.0.3     2023-05-07 [1] CRAN (R 4.3.0)
##  Biostrings         2.68.0    2023-04-25 [1] Bioconductor
##  bit                4.0.5     2022-11-15 [1] CRAN (R 4.3.0)
##  bit64              4.0.5     2020-08-30 [1] CRAN (R 4.3.0)
##  bitops             1.0-7     2021-04-24 [1] CRAN (R 4.3.0)
##  blob               1.2.4     2023-03-17 [1] CRAN (R 4.3.0)
##  bookdown           0.34      2023-05-09 [1] CRAN (R 4.3.0)
##  cachem             1.0.8     2023-05-01 [1] CRAN (R 4.3.0)
##  cli                3.6.1     2023-03-23 [1] CRAN (R 4.3.0)
##  cogeqc           * 1.4.0     2023-04-25 [1] Bioconductor
##  colorspace         2.1-0     2023-01-23 [1] CRAN (R 4.3.0)
##  crayon             1.5.2     2022-09-29 [1] CRAN (R 4.3.0)
##  curl               5.0.0     2023-01-12 [1] CRAN (R 4.3.0)
##  data.table         1.14.8    2023-02-17 [1] CRAN (R 4.3.0)
##  DBI                1.1.3     2022-06-18 [1] CRAN (R 4.3.0)
##  dbplyr             2.3.2     2023-03-21 [1] CRAN (R 4.3.0)
##  digest             0.6.33    2023-07-07 [1] CRAN (R 4.3.0)
##  dplyr            * 1.1.2     2023-04-20 [1] CRAN (R 4.3.0)
##  evaluate           0.21      2023-05-05 [1] CRAN (R 4.3.0)
##  fansi              1.0.4     2023-01-22 [1] CRAN (R 4.3.0)
##  farver             2.1.1     2022-07-06 [1] CRAN (R 4.3.0)
##  fastmap            1.1.1     2023-02-24 [1] CRAN (R 4.3.0)
##  filelock           1.0.2     2018-10-05 [1] CRAN (R 4.3.0)
##  forcats          * 1.0.0     2023-01-29 [1] CRAN (R 4.3.0)
##  generics           0.1.3     2022-07-05 [1] CRAN (R 4.3.0)
##  GenomeInfoDb       1.36.0    2023-04-25 [1] Bioconductor
##  GenomeInfoDbData   1.2.10    2023-04-28 [1] Bioconductor
##  ggbeeswarm         0.7.2     2023-04-29 [1] CRAN (R 4.3.0)
##  ggfun              0.0.9     2022-11-21 [1] CRAN (R 4.3.0)
##  ggplot2          * 3.4.1     2023-02-10 [1] CRAN (R 4.3.0)
##  ggplotify          0.1.0     2021-09-02 [1] CRAN (R 4.3.0)
##  ggtree             3.8.0     2023-04-25 [1] Bioconductor
##  glue               1.6.2     2022-02-24 [1] CRAN (R 4.3.0)
##  gridGraphics       0.5-1     2020-12-13 [1] CRAN (R 4.3.0)
##  gtable             0.3.3     2023-03-21 [1] CRAN (R 4.3.0)
##  here             * 1.0.1     2020-12-13 [1] CRAN (R 4.3.0)
##  hms                1.1.3     2023-03-21 [1] CRAN (R 4.3.0)
##  htmltools          0.5.5     2023-03-23 [1] CRAN (R 4.3.0)
##  httr             * 1.4.5     2023-02-24 [1] CRAN (R 4.3.0)
##  igraph             1.4.2     2023-04-07 [1] CRAN (R 4.3.0)
##  IRanges            2.34.0    2023-04-25 [1] Bioconductor
##  jsonlite           1.8.7     2023-06-29 [1] CRAN (R 4.3.0)
##  KEGGREST           1.40.0    2023-04-25 [1] Bioconductor
##  knitr              1.43      2023-05-25 [1] CRAN (R 4.3.0)
## labeling         0.4.2     2020-10-20 [1] CRAN (R 4.3.0)
## lattice          0.20-45   2021-09-22 [4] CRAN (R 4.2.0)
## lazyeval         0.2.2     2019-03-15 [1] CRAN (R 4.3.0)
## lifecycle        1.0.3     2022-10-07 [1] CRAN (R 4.3.0)
## lubridate      * 1.9.2     2023-02-10 [1] CRAN (R 4.3.0)
## magrittr         2.0.3     2022-03-30 [1] CRAN (R 4.3.0)
## memoise          2.0.1     2021-11-26 [1] CRAN (R 4.3.0)
## munsell          0.5.0     2018-06-12 [1] CRAN (R 4.3.0)
## nlme             3.1-162   2023-01-31 [4] CRAN (R 4.2.2)
## patchwork        1.1.2     2022-08-19 [1] CRAN (R 4.3.0)
## pillar           1.9.0     2023-03-22 [1] CRAN (R 4.3.0)
## pkgconfig        2.0.3     2019-09-22 [1] CRAN (R 4.3.0)
## plyr             1.8.8     2022-11-11 [1] CRAN (R 4.3.0)
## png              0.1-8     2022-11-29 [1] CRAN (R 4.3.0)
## prettyunits      1.1.1     2020-01-24 [1] CRAN (R 4.3.0)
## progress         1.2.2     2019-05-16 [1] CRAN (R 4.3.0)
## purrr          * 1.0.1     2023-01-10 [1] CRAN (R 4.3.0)
## R6               2.5.1     2021-08-19 [1] CRAN (R 4.3.0)
## rappdirs         0.3.3     2021-01-31 [1] CRAN (R 4.3.0)
## Rcpp             1.0.10    2023-01-22 [1] CRAN (R 4.3.0)
## RCurl            1.98-1.12 2023-03-27 [1] CRAN (R 4.3.0)
## readr          * 2.1.4     2023-02-10 [1] CRAN (R 4.3.0)
## reshape2         1.4.4     2020-04-09 [1] CRAN (R 4.3.0)
## rlang            1.1.1     2023-04-28 [1] CRAN (R 4.3.0)
## rmarkdown        2.23      2023-07-01 [1] CRAN (R 4.3.0)
## rprojroot        2.0.3     2022-04-02 [1] CRAN (R 4.3.0)
## RSQLite          2.3.1     2023-04-03 [1] CRAN (R 4.3.0)
## rstudioapi       0.14      2022-08-22 [1] CRAN (R 4.3.0)
## S4Vectors        0.38.0    2023-04-25 [1] Bioconductor
## scales           1.2.1     2022-08-20 [1] CRAN (R 4.3.0)
## sessioninfo      1.2.2     2021-12-06 [1] CRAN (R 4.3.0)
## stringi          1.7.12    2023-01-11 [1] CRAN (R 4.3.0)
## stringr        * 1.5.0     2022-12-02 [1] CRAN (R 4.3.0)
## tibble         * 3.2.1     2023-03-20 [1] CRAN (R 4.3.0)
## tidyr          * 1.3.0     2023-01-24 [1] CRAN (R 4.3.0)
## tidyselect       1.2.0     2022-10-10 [1] CRAN (R 4.3.0)
## tidytree         0.4.2     2022-12-18 [1] CRAN (R 4.3.0)
## tidyverse      * 2.0.0     2023-02-22 [1] CRAN (R 4.3.0)
## timechange       0.2.0     2023-01-11 [1] CRAN (R 4.3.0)
## treeio           1.24.1    2023-05-31 [1] Bioconductor
## tzdb             0.3.0     2022-03-28 [1] CRAN (R 4.3.0)
## utf8             1.2.3     2023-01-31 [1] CRAN (R 4.3.0)
## vctrs            0.6.3     2023-06-14 [1] CRAN (R 4.3.0)
## vipor            0.4.5     2017-03-22 [1] CRAN (R 4.3.0)
## withr            2.5.0     2022-03-03 [1] CRAN (R 4.3.0)
## xfun             0.39      2023-04-20 [1] CRAN (R 4.3.0)
## XML              3.99-0.14 2023-03-19 [1] CRAN (R 4.3.0)
## xml2             1.3.4     2023-04-27 [1] CRAN (R 4.3.0)
## XVector          0.40.0    2023-04-25 [1] Bioconductor
## yaml             2.3.7     2023-01-23 [1] CRAN (R 4.3.0)
## yulab.utils      0.0.6     2022-12-20 [1] CRAN (R 4.3.0)
##  zlibbioc           1.46.0    2023-04-25 [1] Bioconductor
##
##  [1] /home/faalm/R/x86_64-pc-linux-gnu-library/4.3
##  [2] /usr/local/lib/R/site-library
##  [3] /usr/lib/R/site-library
##  [4] /usr/lib/R/library
##
## ------------------------------------------------------------------------------
```

References

Drost, Hajk-Georg, and Jerzy Paszkowski. 2017. "Biomartr: Genomic Data Retrieval with r." *Bioinformatics* 33 (8): 1216–17.

Emms, David M, and Steven Kelly. 2020. "Benchmarking Orthogroup Inference Accuracy: Revisiting Orthobench." *Genome Biology and Evolution* 12 (12): 2258–66.

Trachana, Kalliopi, Tomas A Larsson, Sean Powell, Wei-Hua Chen, Tobias Doerks, Jean Muller, and Peer Bork. 2011. "Orthology Prediction Methods: A Quality Assessment Using Curated Protein Families." *Bioessays* 33 (10): 769–80.

Supplementary Text S3: Assessing orthogroup inference in public databases

Fabricio Almeida-Silva[1,2] ***and Yves Van de Peer***[1,2,3,4]

[1]VIB-UGent Center for Plant Systems Biology, Ghent, Belgium
[2]Department of Plant Biotechnology and Bioinformatics, Ghent University, Ghent, Belgium
[3]College of Horticulture, Academy for Advanced Interdisciplinary Studies, Nanjing Agricultural University, Nanjing, China
[4]Center for Microbial Ecology and Genomics, Department of Biochemistry, Genetics and Microbiology, University of Pretoria, Pretoria, South Africa

7 August 2023

Contents

1 Overview . . . 2

2 Calculating orthogroup scores . . . 2

2.1 PLAZA Dicots 5.0 . . . 2

2.2 OrthoDB, eggNOG, and HOGENOM . . . 4

3 Comparing homogeneity scores . . . 4

Session info . . . 7

References . . . 10

```
library(cogeqc)
library(here)
library(tidyverse)
library(ggpubr)

set.seed(123) # for reproducibility
source(here("code", "utils.R"))
```

1 Overview

Here, we will use the protein domain-based approach in *cogeqc* to assess gene families from different sources, namely:

- PLAZA Dicots 5.0 (Van Bel et al. 2022)
- OrthoDB (Kuznetsov et al. 2023)
- eggNOG (Hernandez-Plaza et al. 2023)
- HOGENOM (Penel et al. 2009)

2 Calculating orthogroup scores

To make comparison possible, we will *Arabidopsis thaliana* domain annotation as a proxy, as this species is present in all of the aforementioned databases. For that, we will use the function `calculate_H()` from *cogeqc*.

Orthogroups assignments from OrthoDB, eggNOG, InParanoid, PhylomeDB, and HOGENOM will be obtained from UniProt.

2.1 PLAZA Dicots 5.0

Below, we will obtain orthogroups and *A. thaliana*'s domain annotation from PLAZA 5.0, and then we will calculate homogeneity scores for each orthogroup.

```
# Obtain gene families from PLAZA
fams_plaza <- readr::read_tsv(
    paste0(
        "https://ftp.psb.ugent.be/pub/plaza/plaza_public_dicots_05/",
        "GeneFamilies/genefamily_data.HOMFAM.csv.gz"
    ), show_col_types = FALSE, skip = 2
) %>%
    filter(species == "ath") %>%
    as.data.frame()
names(fams_plaza) <- c("Orthogroup", "Species", "Gene")
head(fams_plaza)
##     Orthogroup Species      Gene
## 1 HOM05D000001     ath AT1G02310
## 2 HOM05D000001     ath AT1G03510
## 3 HOM05D000001     ath AT1G03540
## 4 HOM05D000001     ath AT1G04020
## 5 HOM05D000001     ath AT1G04840
## 6 HOM05D000001     ath AT1G05750

# Obtain domain anotation for A. thaliana
ath_interpro <- readr::read_tsv(
    paste0(
        "https://ftp.psb.ugent.be/pub/plaza/plaza_public_dicots_05/",
        "InterPro/interpro.ath.csv.gz"
    ), show_col_types = FALSE, skip = 8
) %>%
    select(1,3)
names(ath_interpro) <- c("Gene", "Annotation")
head(ath_interpro)
## # A tibble: 6 x 2
##   Gene      Annotation
##   <chr>     <chr>
## 1 AT1G01010 IPR036093
## 2 AT1G01010 IPR003441
## 3 AT1G01010 IPR036093
## 4 AT1G01020 IPR007290
## 5 AT1G01020 IPR007290
## 6 AT1G01030 IPR003340

# Combining everything and calculating homogeneity scores
fam_df_plaza <- merge(fams_plaza, ath_interpro)
head(fam_df_plaza)
##        Gene   Orthogroup Species Annotation
## 1 AT1G01010 HOM05D000010     ath  IPR036093
## 2 AT1G01010 HOM05D000010     ath  IPR003441
## 3 AT1G01010 HOM05D000010     ath  IPR036093
## 4 AT1G01020 HOM05D006082     ath  IPR007290
## 5 AT1G01020 HOM05D006082     ath  IPR007290
## 6 AT1G01030 HOM05D000466     ath  IPR015300

H_summary <- function(ortho_df = NULL) {
    H <- calculate_H(ortho_df)
    mean_H <- round(mean(H$Score), 2)
    median_H <- round(median(H$Score), 2)
    result_list <- list(H = H, mean_score = mean_H, median_score = median_H)
    return(result_list)
}

H_plaza <- H_summary(fam_df_plaza)
head(H_plaza$H)
##     Orthogroup     Score
## 1 HOM05D000001  283.3132
## 2 HOM05D000002  129.9598
## 3 HOM05D000003  889.1268
## 4 HOM05D000004    0.0000
## 5 HOM05D000005 1135.8799
## 6 HOM05D000006 2820.8337
```

2.2 OrthoDB, eggNOG, and HOGENOM

Orthogroup assignments from these databases will be obtained from UniProt (Consortium 2021).

```r
# Get list of proteins - from primary transcripts only
ath_proteome <- Biostrings::readAAStringSet(
    paste0(
        "https://ftp.uniprot.org/pub/databases/uniprot/",
        "current_release/knowledgebase/reference_proteomes/Eukaryota/",
        "UP000006548/UP000006548_3702.fasta.gz"
    )
)
ath_proteins <- names(ath_proteome)
ath_proteins <- sapply(strsplit(ath_proteins, split = "\\|"), `[`, 2)

# Extract phylogenomic information for all genes
source(here::here("code", "utils.R"))
fams_uniprot <- extract_ogs_uniprot(ath_proteins)

fams_orthodb <- fams_uniprot[, c("Gene", "OrthoDB")] %>% drop_na()
fams_eggnog <- fams_uniprot[, c("Gene", "eggNOG")] %>% drop_na()
fams_hogenom <- fams_uniprot[, c("Gene", "HOGENOM")] %>% drop_na()

#----Calculate homogeneity scores for each database----------------------------
# OrthoDB
fams_df_orthodb <- merge(fams_orthodb, ath_interpro)
names(fams_df_orthodb)[2] <- "Orthogroup"
H_orthodb <- H_summary(fams_df_orthodb)

# eggNOG
fams_df_eggnog <- merge(fams_eggnog, ath_interpro)
names(fams_df_eggnog)[2] <- "Orthogroup"
H_eggnog <- H_summary(fams_df_eggnog)

# HOGENOM
fams_df_hogenom <- merge(fams_hogenom, ath_interpro)
names(fams_df_hogenom)[2] <- "Orthogroup"
H_hogenom <- H_summary(fams_df_hogenom)
```

3 Comparing homogeneity scores

Finally, let's compare homogeneity scores and visualize their distributions. First, let's combine all data frames of homogeneity scores into a single data frame.

```r
H_combined <- bind_rows(
    H_plaza$H %>% mutate(Source = "PLAZA"),
    H_orthodb$H %>% mutate(Source = "OrthoDB"),
    H_eggnog$H %>% mutate(Source = "eggNOG"),
    H_hogenom$H %>% mutate(Source = "HOGENOM")
)
```

```
save(
    H_combined,
    file = here::here("products", "result_files", "H_combined.rda"),
    compress = "xz"
)
```

Now, let's compare the distributions of homogeneity scores for each database to see if there are any differences. For that, we will calculate P-values from a Wilcoxon test with Wicoxon effect sizes (r). The Wilcoxon effect size is calculated as the Z statistic divided by the square root of the sample size.

```
# Scale scores to maximum, so that they range from 0 to 1
H_combined$Score <- H_combined$Score / max(H_combined$Score)
head(H_combined)
##      Orthogroup      Score Source
## 1 HOM05D000001 0.10043599  PLAZA
## 2 HOM05D000002 0.04607143  PLAZA
## 3 HOM05D000003 0.31520000  PLAZA
## 4 HOM05D000004 0.00000000  PLAZA
## 5 HOM05D000005 0.40267523  PLAZA
## 6 HOM05D000006 1.00000000  PLAZA

# Quick exploration of means and medians
H_combined %>%
    group_by(Source) %>%
    summarise(mean = mean(Score), median = median(Score))
## # A tibble: 4 x 3
##   Source   mean median
##   <chr>   <dbl>  <dbl>
## 1 HOGENOM 0.603  0.609
## 2 OrthoDB 0.578  0.567
## 3 PLAZA   0.610  0.6
## 4 eggNOG  0.565  0.546

# Compare homogeneity scores - all vs all
db_wilcox <- compare(H_combined, "Score ~ Source")

db_wilcox |>
    filter_comparison() |>
    knitr::kable(
        caption = "Mann-Whitney U test for differences in orthogroup scores with Wilcoxon effect sizes.",
        digits = 10
    )
```

We can see that there are diffences in mean. In summary:

1. eggNOG orthogroups have lower scores than every other source
2. HOGENOM orthogroups have higher scores than OrthoDB, but lower than PLAZA.
3. PLAZA orthogroup scores are higher than every other database.

Table 1: Mann-Whitney U test for differences in orthogroup scores with Wilcoxon effect sizes.

group1	group2	n1	n2	padj	effsize	magnitude
eggNOG	HOGENOM	3092	3257	0.0e+00	0.11102956	small
eggNOG	OrthoDB	3092	3201	8.5e-09	0.07197679	small
eggNOG	PLAZA	3092	3503	0.0e+00	0.09434683	small
HOGENOM	OrthoDB	3257	3201	0.0e+00	0.09071787	small
HOGENOM	PLAZA	3257	3503	3.0e-03	0.03402611	small
OrthoDB	PLAZA	3201	3503	7.0e-10	0.07526911	small

However, the effect sizes are very small, suggesting that significant differences could be due to large sample sizes, as P-values are highly affected by sample sizes.

Now, let's visualize the distributions with significant differences highlighted. Here, we will only display comparison bars for comparisons with $P < 0.05$ and effect sizes > 0.1.

```r
# Comparisons to be made
comps <- list(
    c("HOGENOM", "eggNOG")
)

# Change order of levels according to comparison results
H_combined$Source <- factor(
    H_combined$Source, levels = rev(c(
        "PLAZA", "HOGENOM", "OrthoDB", "eggNOG"
    ))
)

# Visualize distributions with significant differences highlighted
distros <- ggviolin(
    H_combined, y = "Score", x = "Source",
    orientation = "horiz", trim = TRUE, add = c("boxplot", "mean"),
    fill = "Source", add.params = list(fill = "white"), palette = "jama"
) +
    ggpubr::stat_compare_means(
        comparisons = comps,
        label = "p.signif",
        method = "wilcox.test"
    ) +
    theme(legend.position = "none") +
    labs(y = "Scaled homogeneity scores", x = "Source of orthogroups",
         title = "Distribution of mean homogeneity scores for orthogroups",
         subtitle = "Scores were calculated based on *A. thaliana* genes") +
    theme(plot.subtitle = ggtext::element_markdown())

distros
```

To conclude, despite some significant differences, all databases perform equally well in their orthogroup definition. The observed differences in means could be due to large sample sizes, as indicated by very low effect sizes, and to the different species composition of the database.

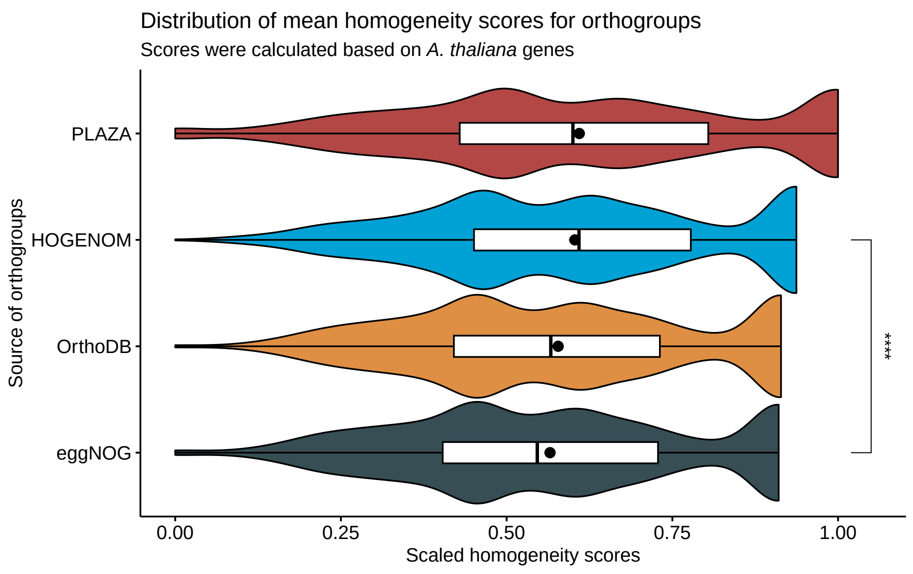


Figure 1: Distribution of mean orthogroup scores.

Session info

This document was created under the following conditions:

```
sessioninfo::session_info()
## - Session info ---------------------------------------------------------------
##  setting  value
##  version  R version 4.3.0 (2023-04-21)
##  os       Ubuntu 20.04.5 LTS
##  system   x86_64, linux-gnu
##  ui       X11
##  language (EN)
##  collate  en_US.UTF-8
##  ctype    en_US.UTF-8
##  tz       Europe/Brussels
##  date     2023-08-07
##  pandoc   3.1.1 @ /usr/lib/rstudio/resources/app/bin/quarto/bin/tools/ (via rmarkdown)
##
## - Packages -------------------------------------------------------------------
##  package            * version   date (UTC) lib source
##  abind                1.4-5     2016-07-21 [1] CRAN (R 4.3.0)
##  ape                  5.7-1     2023-03-13 [1] CRAN (R 4.3.0)
##  aplot                0.1.10    2023-03-08 [1] CRAN (R 4.3.0)
##  backports            1.4.1     2021-12-13 [1] CRAN (R 4.3.0)
##  beeswarm             0.4.0     2021-06-01 [1] CRAN (R 4.3.0)
##  BiocGenerics         0.46.0    2023-04-25 [1] Bioconductor
##  BiocManager          1.30.21.1 2023-07-18 [1] CRAN (R 4.3.0)
##  BiocStyle          * 2.29.1    2023-08-04 [1] Github (Bioconductor/BiocStyle@7c0e093)
##  Biostrings           2.68.0    2023-04-25 [1] Bioconductor
##  bit                  4.0.5     2022-11-15 [1] CRAN (R 4.3.0)
## bit64              4.0.5    2020-08-30 [1] CRAN (R 4.3.0)
## bitops             1.0-7    2021-04-24 [1] CRAN (R 4.3.0)
## bookdown           0.34     2023-05-09 [1] CRAN (R 4.3.0)
## broom              1.0.4    2023-03-11 [1] CRAN (R 4.3.0)
## car                3.1-2    2023-03-30 [1] CRAN (R 4.3.0)
## carData            3.0-5    2022-01-06 [1] CRAN (R 4.3.0)
## cli                3.6.1    2023-03-23 [1] CRAN (R 4.3.0)
## codetools          0.2-19   2023-02-01 [4] CRAN (R 4.2.2)
## cogeqc           * 1.4.0    2023-04-25 [1] Bioconductor
## coin               1.4-2    2021-10-08 [1] CRAN (R 4.3.0)
## colorspace         2.1-0    2023-01-23 [1] CRAN (R 4.3.0)
## commonmark         1.9.0    2023-03-17 [1] CRAN (R 4.3.0)
## crayon             1.5.2    2022-09-29 [1] CRAN (R 4.3.0)
## curl               5.0.0    2023-01-12 [1] CRAN (R 4.3.0)
## digest             0.6.33   2023-07-07 [1] CRAN (R 4.3.0)
## dplyr            * 1.1.2    2023-04-20 [1] CRAN (R 4.3.0)
## evaluate           0.21     2023-05-05 [1] CRAN (R 4.3.0)
## fansi              1.0.4    2023-01-22 [1] CRAN (R 4.3.0)
## farver             2.1.1    2022-07-06 [1] CRAN (R 4.3.0)
## fastmap            1.1.1    2023-02-24 [1] CRAN (R 4.3.0)
## forcats          * 1.0.0    2023-01-29 [1] CRAN (R 4.3.0)
## generics           0.1.3    2022-07-05 [1] CRAN (R 4.3.0)
## GenomeInfoDb       1.36.0   2023-04-25 [1] Bioconductor
## GenomeInfoDbData   1.2.10   2023-04-28 [1] Bioconductor
## ggbeeswarm         0.7.2    2023-04-29 [1] CRAN (R 4.3.0)
## ggfun              0.0.9    2022-11-21 [1] CRAN (R 4.3.0)
## ggplot2          * 3.4.1    2023-02-10 [1] CRAN (R 4.3.0)
## ggplotify          0.1.0    2021-09-02 [1] CRAN (R 4.3.0)
## ggpubr           * 0.6.0    2023-02-10 [1] CRAN (R 4.3.0)
## ggsci              3.0.0    2023-03-08 [1] CRAN (R 4.3.0)
## ggsignif           0.6.4    2022-10-13 [1] CRAN (R 4.3.0)
## ggtext             0.1.2    2022-09-16 [1] CRAN (R 4.3.0)
## ggtree             3.8.0    2023-04-25 [1] Bioconductor
## glue               1.6.2    2022-02-24 [1] CRAN (R 4.3.0)
## gridGraphics       0.5-1    2020-12-13 [1] CRAN (R 4.3.0)
## gridtext           0.1.5    2022-09-16 [1] CRAN (R 4.3.0)
## gtable             0.3.3    2023-03-21 [1] CRAN (R 4.3.0)
## here             * 1.0.1    2020-12-13 [1] CRAN (R 4.3.0)
## hms                1.1.3    2023-03-21 [1] CRAN (R 4.3.0)
## htmltools          0.5.5    2023-03-23 [1] CRAN (R 4.3.0)
## igraph             1.4.2    2023-04-07 [1] CRAN (R 4.3.0)
## IRanges            2.34.0   2023-04-25 [1] Bioconductor
## jsonlite           1.8.7    2023-06-29 [1] CRAN (R 4.3.0)
## knitr              1.43     2023-05-25 [1] CRAN (R 4.3.0)
## labeling           0.4.2    2020-10-20 [1] CRAN (R 4.3.0)
## lattice            0.20-45  2021-09-22 [4] CRAN (R 4.2.0)
## lazyeval           0.2.2    2019-03-15 [1] CRAN (R 4.3.0)
## libcoin            1.0-9    2021-09-27 [1] CRAN (R 4.3.0)
## lifecycle          1.0.3    2022-10-07 [1] CRAN (R 4.3.0)
## lubridate        * 1.9.2    2023-02-10 [1] CRAN (R 4.3.0)
## magrittr           2.0.3    2022-03-30 [1] CRAN (R 4.3.0)
##  markdown           1.6       2023-04-07 [1] CRAN (R 4.3.0)
##  MASS               7.3-58.2  2023-01-23 [4] CRAN (R 4.2.2)
##  Matrix             1.5-1     2022-09-13 [4] CRAN (R 4.2.1)
##  matrixStats        1.0.0     2023-06-02 [1] CRAN (R 4.3.0)
##  modeltools         0.2-23    2020-03-05 [1] CRAN (R 4.3.0)
##  multcomp           1.4-25    2023-06-20 [1] CRAN (R 4.3.0)
##  munsell            0.5.0     2018-06-12 [1] CRAN (R 4.3.0)
##  mvtnorm            1.1-3     2021-10-08 [1] CRAN (R 4.3.0)
##  nlme               3.1-162   2023-01-31 [4] CRAN (R 4.2.2)
##  patchwork          1.1.2     2022-08-19 [1] CRAN (R 4.3.0)
##  pillar             1.9.0     2023-03-22 [1] CRAN (R 4.3.0)
##  pkgconfig          2.0.3     2019-09-22 [1] CRAN (R 4.3.0)
##  plyr               1.8.8     2022-11-11 [1] CRAN (R 4.3.0)
##  purrr            * 1.0.1     2023-01-10 [1] CRAN (R 4.3.0)
##  R6                 2.5.1     2021-08-19 [1] CRAN (R 4.3.0)
##  Rcpp               1.0.10    2023-01-22 [1] CRAN (R 4.3.0)
##  RCurl              1.98-1.12 2023-03-27 [1] CRAN (R 4.3.0)
##  readr            * 2.1.4     2023-02-10 [1] CRAN (R 4.3.0)
##  reshape2           1.4.4     2020-04-09 [1] CRAN (R 4.3.0)
##  rlang              1.1.1     2023-04-28 [1] CRAN (R 4.3.0)
##  rmarkdown          2.23      2023-07-01 [1] CRAN (R 4.3.0)
##  rprojroot          2.0.3     2022-04-02 [1] CRAN (R 4.3.0)
##  rstatix            0.7.2     2023-02-01 [1] CRAN (R 4.3.0)
##  rstudioapi         0.14      2022-08-22 [1] CRAN (R 4.3.0)
##  S4Vectors          0.38.0    2023-04-25 [1] Bioconductor
##  sandwich           3.0-2     2022-06-15 [1] CRAN (R 4.3.0)
##  scales             1.2.1     2022-08-20 [1] CRAN (R 4.3.0)
##  sessioninfo        1.2.2     2021-12-06 [1] CRAN (R 4.3.0)
##  stringi            1.7.12    2023-01-11 [1] CRAN (R 4.3.0)
##  stringr          * 1.5.0     2022-12-02 [1] CRAN (R 4.3.0)
##  survival           3.5-3     2023-02-12 [4] CRAN (R 4.2.2)
##  TH.data            1.1-2     2023-04-17 [1] CRAN (R 4.3.0)
##  tibble           * 3.2.1     2023-03-20 [1] CRAN (R 4.3.0)
##  tidyr            * 1.3.0     2023-01-24 [1] CRAN (R 4.3.0)
##  tidyselect         1.2.0     2022-10-10 [1] CRAN (R 4.3.0)
##  tidytree           0.4.2     2022-12-18 [1] CRAN (R 4.3.0)
##  tidyverse        * 2.0.0     2023-02-22 [1] CRAN (R 4.3.0)
##  timechange         0.2.0     2023-01-11 [1] CRAN (R 4.3.0)
##  treeio             1.24.1    2023-05-31 [1] Bioconductor
##  tzdb               0.3.0     2022-03-28 [1] CRAN (R 4.3.0)
##  utf8               1.2.3     2023-01-31 [1] CRAN (R 4.3.0)
##  vctrs              0.6.3     2023-06-14 [1] CRAN (R 4.3.0)
##  vipor              0.4.5     2017-03-22 [1] CRAN (R 4.3.0)
##  vroom              1.6.3     2023-04-28 [1] CRAN (R 4.3.0)
##  withr              2.5.0     2022-03-03 [1] CRAN (R 4.3.0)
##  xfun               0.39      2023-04-20 [1] CRAN (R 4.3.0)
##  xml2               1.3.4     2023-04-27 [1] CRAN (R 4.3.0)
##  XVector            0.40.0    2023-04-25 [1] Bioconductor
##  yaml               2.3.7     2023-01-23 [1] CRAN (R 4.3.0)
##  yulab.utils        0.0.6     2022-12-20 [1] CRAN (R 4.3.0)
##  zlibbioc           1.46.0    2023-04-25 [1] Bioconductor
##  zoo                1.8-12    2023-04-13 [1] CRAN (R 4.3.0)
##
##  [1] /home/faalm/R/x86_64-pc-linux-gnu-library/4.3
##  [2] /usr/local/lib/R/site-library
##  [3] /usr/lib/R/site-library
##  [4] /usr/lib/R/library
##
## ------------------------------------------------------------------------------
```

References

Consortium, The UniProt. 2021. "UniProt: The Universal Protein Knowledgebase in 2021." *Nucleic Acids Research* 49 (D1): D480–89.

Hernandez-Plaza, Ana, Damian Szklarczyk, Jorge Botas, Carlos P Cantalapiedra, Joaquin Giner-Lamia, Daniel R Mende, Rebecca Kirsch, et al. 2023. "eggNOG 6.0: Enabling Comparative Genomics Across 12 535 Organisms." *Nucleic Acids Research* 51 (D1): D389–94.

Kuznetsov, Dmitry, Fredrik Tegenfeldt, Mose Manni, Mathieu Seppey, Matthew Berkeley, Evgenia V Kriventseva, and Evgeny M Zdobnov. 2023. "OrthoDB V11: Annotation of Orthologs in the Widest Sampling of Organismal Diversity." *Nucleic Acids Research* 51 (D1): D445–51.

Penel, Simon, Anne-Muriel Arigon, Jean-François Dufayard, Anne-Sophie Sertier, Vincent Daubin, Laurent Duret, Manolo Gouy, and Guy Perrière. 2009. "Databases of Homologous Gene Families for Comparative Genomics." In *BMC Bioinformatics*, 10:1–13. 6. BioMed Central.

Van Bel, Michiel, Francesca Silvestri, Eric M Weitz, Lukasz Kreft, Alexander Botzki, Frederik Coppens, and Klaas Vandepoele. 2022. "PLAZA 5.0: Extending the Scope and Power of Comparative and Functional Genomics in Plants." *Nucleic Acids Research* 50 (D1): D1468–74.

Supplementary Text S4: Assessing orthogroup inference for Brassicaceae genomes

Fabricio Almeida-Silva [1,2] ***and Yves Van de Peer*** [1,2,3,4]

[1]VIB-UGent Center for Plant Systems Biology, Ghent, Belgium
[2]Department of Plant Biotechnology and Bioinformatics, Ghent University, Ghent, Belgium
[3]College of Horticulture, Academy for Advanced Interdisciplinary Studies, Nanjing Agricultural University, Nanjing, China
[4]Center for Microbial Ecology and Genomics, Department of Biochemistry, Genetics and Microbiology, University of Pretoria, Pretoria, South Africa

8 August 2023

Contents

1 Overview . . . 2
2 Orthogroup inference . . . 2
2.1 Exploratory analysis of orthogroup inference results . . . 3
3 Orthogroup assessment . . . 6
4 Comparing and visualizing homogeneity statistics . . . 7
4.1 Global distributions . . . 8
4.2 The effect of species choice . . . 10
4.3 The effect of mcl inflation parameters . . . 12
4.4 The effect of DIAMOND mode (default vs ultra) . . . 14
5 Functional analysis of homogeneous and heterogeneous gene families . . . 17
6 Is there an association between OG score and OG gene length? . . . 24
Session info . . . 29
References . . . 33

```r
library(here)
library(cogeqc)
library(tidyverse)
library(ggpubr)
library(rstatix)
library(clusterProfiler)
library(enrichplot)
library(patchwork)
library(dplyr)

source(here("code", "utils.R"))
```

1 Overview

Here, we will compare the protein domain-based approach in `cogeqc` to assess the impact of multiple combinations of parameters in OrthoFinder (Emms and Kelly 2019) in the accuracy of orthogroup inference. The data set used here will be a collection of Brassicaceae genomes. The parameters we will change are:

1. Program (`-S` option)

- DIAMOND
- DIAMOND ultrasensitive

2. MCL inflation parameter (`-I`)

- 1
- 1.5 (default)
- 2
- 3

2 Orthogroup inference

To start, we will load the proteome data and export each proteome as a FASTA file in the `data` directory, so we can pass it to OrthoFinder.

```r
# Load proteomes
load(here("data", "brassicaceae_proteomes.rda"))

# Write files to data/
lapply(seq_along(brassicaceae_proteomes), function(x) {
    outfile <- here("data", paste0(names(brassicaceae_proteomes)[x], ".fasta"))
    Biostrings::writeXStringSet(
        brassicaceae_proteomes[[x]], outfile
    )
})
```

Now, we can run OrthoFinder for each combination of parameters. Here, we created 2 different bash scripts for each DIAMOND mode. They are:

- `of_diamond.sh`: code to run DIAMOND (default mode) for different inflation parameters;

- `of_diamond_ultra.sh`: code to run DIAMOND in ultrasensitive mode for different inflation parameters

The 2 files can be run with:

```
bash of_diamond.sh
bash of_diamond_ultra.sh
```

The *Orthogroups.tsv* files were all moved to the directory `products/result_files`.

2.1 Exploratory analysis of orthogroup inference results

Now that we have the *Orthogroups.tsv* files from OrthoFinder, let's load them.

```
# Extract tar.xz file
tarfile <- here("products", "result_files", "Orthogroups.tar.xz")
outdir <- tempdir()

system2("tar", args = c("-xf", tarfile, "--directory", outdir))

# Get path to OrthoFinder output
og_files <- list.files(
    path = outdir,
    pattern = "Orthogroups.*", full.names = TRUE
)

# Read and parse files
ogs <- lapply(og_files, function(x) {
    og <- read_orthogroups(x)
    og <- og %>%
        mutate(Species = stringr::str_replace_all(Species, "\\.", "")) %>%
        mutate(Gene = str_replace_all(
            Gene, c(
                "\\.[0-9]$" = "",
                "\\.[0-9]\\.p$" = "",
                "\\.t[0-9]$" = "",
                "\\.g$" = ""
            )
        ))
    return(og)
})
og_names <- gsub("\\.tsv", "", basename(og_files))
og_names <- gsub("Orthogroups_", "", og_names)

names(ogs) <- og_names
```

Let's explore OG sizes for each combination of parameters and filter orthogroups by size to remove orthogroups that are artificially large.

```
# Visualize OG sizes
og_sizes_plot <- patchwork::wrap_plots(
    plot_og_sizes(ogs$default_1) + ggtitle("Default, mcl = 1"),
    plot_og_sizes(ogs$default_1_5) + ggtitle("Default, mcl = 1.5") +
        theme(axis.text.y = element_blank()),
    plot_og_sizes(ogs$default_2) + ggtitle("Default, mcl = 2") +
        theme(axis.text.y = element_blank()),
    plot_og_sizes(ogs$default_3) + ggtitle("Default, mcl = 3") +
        theme(axis.text.y = element_blank()),
    plot_og_sizes(ogs$ultra_1) + ggtitle("Ultra, mcl = 1") +
        theme(axis.text.y = element_blank()),
    plot_og_sizes(ogs$ultra_1_5) + ggtitle("Ultra, mcl = 1.5") +
        theme(axis.text.y = element_blank()),
    plot_og_sizes(ogs$ultra_2) + ggtitle("Ultra, mcl = 2") +
        theme(axis.text.y = element_blank()),
    plot_og_sizes(ogs$ultra_3) + ggtitle("Ultra, mcl = 3") +
        theme(axis.text.y = element_blank()),
    nrow = 1, ncol = 8
)

og_sizes_plot
```

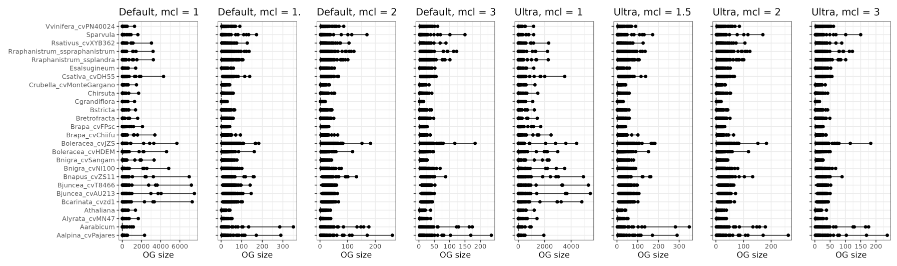


Figure 1: Orthogroup sizes for each run.

Expectedly, OrthoFinder runs with mcl inflation parameters of 1 lead to very large orthogroups, including some orthogroups with thousands of genes.

Now, let's explore the percentage of orthogroups with >200, >100, and >50 genes in each OrthoFinder run.

```
# Calculate OG sizes for each run
og_sizes <- lapply(ogs, function(x) {
    sizes <- as.matrix(table(x$Orthogroup, x$Species))
    total <- rowSums(sizes)

    sizes_df <- data.frame(unclass(sizes))
    sizes_df$Total <- total
    return(sizes_df)
})

# What is the percentage of OGs with >=100 genes? And with >50 genes?
percentage_size <- function(size_df, n = 100) {
    return(sum(size_df$Total > n) / nrow(size_df) * 100)
}

percentages <- data.frame(
    Mode = names(og_sizes),
    P200 = unlist(lapply(og_sizes, percentage_size, n = 200)),
    P100 = unlist(lapply(og_sizes, percentage_size, n = 100)),
    P50 = unlist(lapply(og_sizes, percentage_size, n = 50)),
    OGs = unlist(lapply(og_sizes, nrow))
)

# Reorder rows from lowest to highest mcl inflation
orders <- c(
    "default_1", "default_1_5", "default_2", "default_3",
    "ultra_1", "ultra_1_5", "ultra_2", "ultra_3"
)
percentages <- percentages[orders, ]

# Visual exploration
percentage_plot <- percentages %>%
    tidyr::pivot_longer(cols = !Mode) %>%
    mutate(name = str_replace_all(
        name,
        c(
            "OGs" = "Number of OGs",
            "P200" = "% OGs with >200 genes",
            "P100" = "% OGs with >100 genes",
            "P50" = "% OGs with >50 genes"
        )
    )) %>%
    ggplot(., aes(y = Mode, x = value)) +
    geom_col(aes(fill = Mode), show.legend = "none") +
    scale_fill_manual(
        values = c("ultra_3" = "#08519C", "ultra_2" = "#3182BD",
                   "ultra_1_5" = "#6BAED6", "ultra_1" = "#BDD7E7",
                   "default_3" = "#006D2C", "default_2" = "#31A354",
                   "default_1_5" = "#74C476", "default_1" = "#BAE4B3")
    ) +
    facet_wrap(~name, ncol = 4, scales = "free_x") +
    theme_bw() +
    labs(
        x = "", y = "OrthoFinder mode",
        title = "Relationship between the number of orthogroups and orthogroup size per OrthoFinder mode"
    )

percentage_plot
```

It is very clear that increasing the mcl inflation increases the number of orthogroups, but decreases the percentage of OGs with more than 100 and 50 genes.

Finally, let's remove OGs with >=200 genes to remove noise.

```
# Filter OGs
ogs_filtered <- lapply(seq_along(ogs), function(x) {

    # Which OGs less than 200 genes?
```

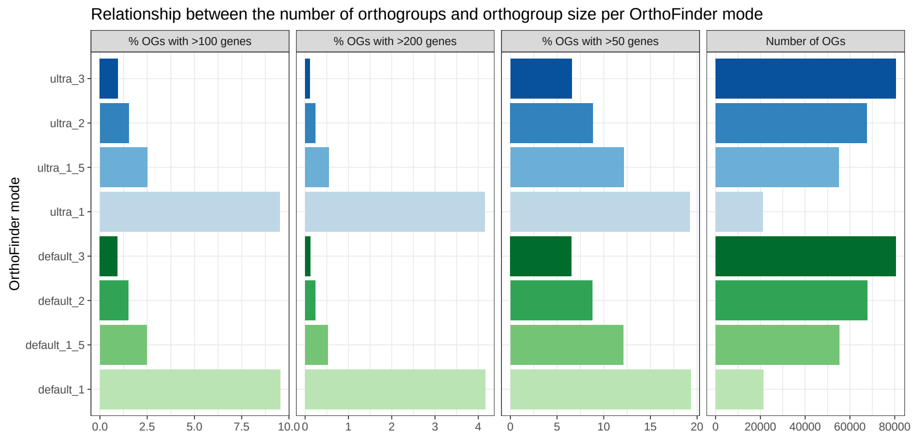


Figure 2: Percentage of orthogroups with $>$50, $>$100, and $>$200 genes for each run.

```
    og_keep <- rownames(og_sizes[[x]][og_sizes[[x]]$Total < 200, ])

    fogs <- ogs[[x]][ogs[[x]]$Orthogroup %in% og_keep, ]
    return(fogs)
})
names(ogs_filtered) <- names(ogs)
```

3 Orthogroup assessment

Now, let's get InterPro domain annotation for the following species to assess orthogroups:

- *A. thaliana*
- *A. arabicum*
- *A. lyrata*
- *B. carinata*
- *C. rubella*
- *C. hirsuta*
- *S. parvula*

```
# Define function to read functional annotation from PLAZA 5.0
read_annotation <- function(url, cols = c(1, 3)) {
    annot <- readr::read_tsv(url, show_col_types = FALSE, skip = 8) %>%
        select(cols)
    names(annot)[1:2] <- c("Gene", "Annotation")
    return(annot)
}
```

```
# Get Interpro annotation
base <- "https://ftp.psb.ugent.be/pub/plaza/plaza_public_dicots_05/InterPro/"
interpro <- list(
    Athaliana = read_annotation(paste0(base, "interpro.ath.csv.gz")),
    Aarabicum = read_annotation(paste0(base, "interpro.aar.csv.gz")),
    Alyrata_cvMN47 = read_annotation(paste0(base, "interpro.aly.csv.gz")),
    Bcarinata_cvzd1 = read_annotation(paste0(base, "interpro.bca.csv.gz")),
    Crubella_cvMonteGargano = read_annotation(paste0(base, "interpro.cru.csv.gz")),
    Chirsuta = read_annotation(paste0(base, "interpro.chi.csv.gz")),
    Sparvula = read_annotation(paste0(base, "interpro.spa.csv.gz"))
)
interpro <- lapply(interpro, as.data.frame)

# Calculate homogeneity scores
species_annotation <- names(interpro)
og_assessment <- lapply(seq_along(ogs_filtered), function(x) {

    message("Working on mode ", names(ogs_filtered)[x])
    orthogroups <- ogs_filtered[[x]]
    orthogroups <- orthogroups[orthogroups$Species %in% species_annotation, ]

    res <- assess_orthogroups(orthogroups, interpro)
    res$Mode <- factor(
        names(ogs_filtered)[x],
        levels = c(
            "ultra_3", "ultra_2", "ultra_1_5", "ultra_1",
            "default_3", "default_2", "default_1_5", "default_1"
        )
    )
    return(res)
})
og_assessment <- Reduce(rbind, og_assessment)

# Save homogeneity stats
save(
    og_assessment, compress = "xz",
    file = here("products", "result_files", "og_assessment_brassicaceae.rda")
)
```

4 Comparing and visualizing homogeneity statistics

Here, we will compare and visualize how the homogeneity scores are affected by:

- different species choice
- different mcl inflation values
- different DIAMOND modes (default and ultra)

Quick exploration of median and mean homogeneity:

```
load(here("products", "result_files", "og_assessment_brassicaceae.rda"))

# Scale value to the maximum so that values range from 0 to 1
og_assessment$Median_score <- og_assessment$Median_score /
    max(og_assessment$Median_score)

# Mean
mean_og <- og_assessment %>%
    group_by(Mode) %>%
    summarise(mean = mean(Median_score))

# Median
median_og <- og_assessment %>%
    group_by(Mode) %>%
    summarise(median = median(Median_score))

mean_and_median_og <- inner_join(mean_og, median_og) |>
    dplyr::rename(Mean = mean, Median = median)

knitr::kable(mean_and_median_og, caption = "Mean and median OG scores.", digits = 3)
```

Table 1: Mean and median OG scores.

Mode	Mean	Median
ultra_3	0.640	0.640
ultra_2	0.631	0.635
ultra_1_5	0.620	0.628
ultra_1	0.425	0.424
default_3	0.639	0.640
default_2	0.631	0.635
default_1_5	0.620	0.628
default_1	0.425	0.423

4.1 Global distributions

Here, we will compare and visualize all distros considering different DIAMOND modes and mcl inflation values. To start, let's perform Wilcoxon tests for all combinations of modes and obtain effect sizes.

```
# Relevel 'Mode' factor
og_assessment$Mode <- factor(
    og_assessment$Mode,
    levels = c(
        "ultra_3", "ultra_2", "ultra_1_5", "ultra_1",
        "default_3", "default_2", "default_1_5", "default_1"
    )
)

# Comparing all vs all
comp_global <- compare(og_assessment, "Median_score ~ Mode")
comp_global |>
    filter_comparison() |>
    knitr::kable(
        caption = "Mann-Whitney U test for differences in orthogroup scores with Wilcoxon effect sizes.",
        digits = 10
    )
```

Table 2: Mann-Whitney U test for differences in orthogroup scores with Wilcoxon effect sizes.

group1	group2	n1	n2	padj	effsize	magnitude
ultra_3	ultra_2	19738	18575	0.00e+00	0.04120347	small
ultra_3	ultra_1_5	19738	16898	0.00e+00	0.06125964	small
ultra_3	ultra_1	19738	5534	0.00e+00	0.34087185	moderate
ultra_3	default_3	19738	19765	5.00e-10	0.03113134	small
ultra_3	default_2	19738	18633	0.00e+00	0.04197169	small
ultra_3	default_1_5	19738	16975	0.00e+00	0.06233198	small
ultra_3	default_1	19738	5587	0.00e+00	0.34258513	moderate
ultra_2	ultra_1_5	18575	16898	0.00e+00	0.04340346	small
ultra_2	ultra_1	18575	5534	0.00e+00	0.33536590	moderate
ultra_2	default_3	18575	19765	0.00e+00	0.04018176	small
ultra_2	default_2	18575	18633	2.55e-08	0.02855053	small
ultra_2	default_1_5	18575	16975	0.00e+00	0.04451653	small
ultra_2	default_1	18575	5587	0.00e+00	0.33742024	moderate
ultra_1_5	ultra_1	16898	5534	0.00e+00	0.32401575	moderate
ultra_1_5	default_3	16898	19765	0.00e+00	0.05737302	small
ultra_1_5	default_2	16898	18633	0.00e+00	0.04259062	small
ultra_1_5	default_1_5	16898	16975	1.72e-06	0.02554732	small
ultra_1_5	default_1	16898	5587	0.00e+00	0.32541386	moderate
ultra_1	default_3	5534	19765	0.00e+00	0.34014551	moderate
ultra_1	default_2	5534	18633	0.00e+00	0.33456485	moderate
ultra_1	default_1_5	5534	16975	0.00e+00	0.32260079	moderate
ultra_1	default_1	5534	5587	2.70e-02	0.01928759	small
default_3	default_2	19765	18633	0.00e+00	0.04084430	small
default_3	default_1_5	19765	16975	0.00e+00	0.06124558	small
default_3	default_1	19765	5587	0.00e+00	0.34147823	moderate
default_2	default_1_5	18633	16975	0.00e+00	0.04366051	small
default_2	default_1	18633	5587	0.00e+00	0.33616829	moderate
default_1_5	default_1	16975	5587	0.00e+00	0.32420710	moderate

As we can see, using mcl = 1 leads to much smaller homogeneity scores as compared to every other mcl value. For mcl values >=1.5, there are differences, but they are likely due to large sample sizes, as indicated by small effect sizes.

The default OrthoFinder mode (default DIAMOND, mcl = 1.5) leads to higher homogeneity as compared to runs using mcl = 1, both in default and ultrasensitive DIAMOND modes. The difference between the default mode and runs with higher mcl values are negligible.

Now, let's visualize the distributions and compare the default OrthoFinder mode with every other mode, highlighting significant differences ($P < 0.05$) with effect size > 0.1.

```r
# Visualize
global_comps <- list(
    c("default_1_5", "ultra_1"),
    c("default_1_5", "default_1")
)

p_distros_global <- ggviolin(
    og_assessment, y = "Median_score", x = "Mode",
    orientation = "horiz", trim = TRUE,
    add = c("boxplot", "mean"),
    fill = "Mode", add.params = list(fill = "white")
) +
    scale_fill_manual(
        values = c("ultra_3" = "#08519C", "ultra_2" = "#3182BD",
                   "ultra_1_5" = "#6BAED6", "ultra_1" = "#BDD7E7",
                   "default_3" = "#006D2C", "default_2" = "#31A354",
                   "default_1_5" = "#74C476", "default_1" = "#BAE4B3")
    ) +
    stat_compare_means(
        comparisons = global_comps, label = "p.signif",
        method = "wilcox.test"
    ) +
    theme(legend.position = "none") +
    labs(y = "Scaled homogeneity scores", x = "OrthoFinder modes",
         title = "Distribution of mean homogeneity scores for orthogroups") +
    theme(plot.subtitle = ggtext::element_markdown())

p_distros_global
```

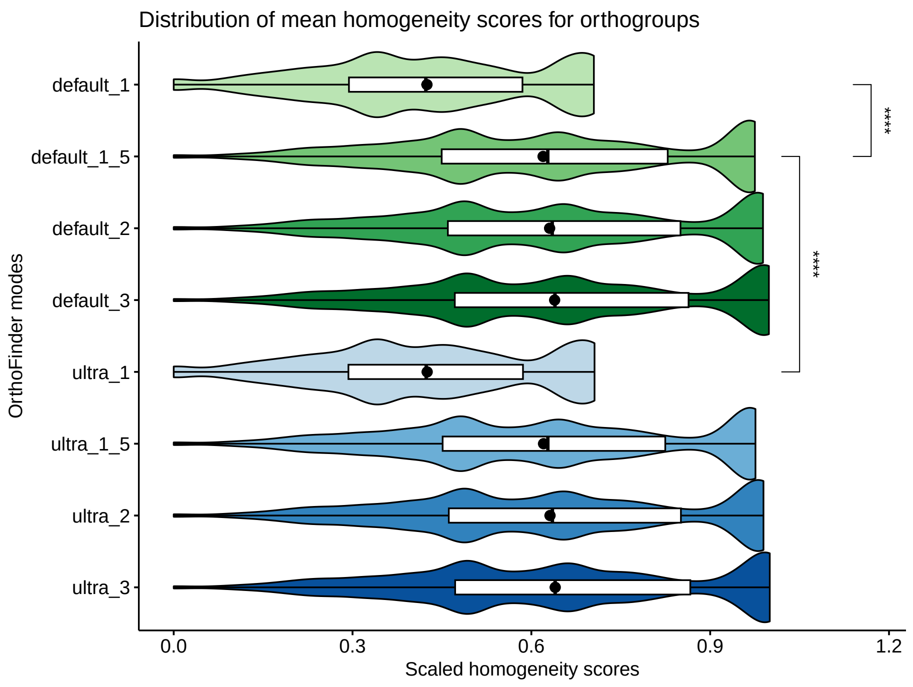


Figure 3: Distribution of mean orthogroup scores for each OrthoFinder run.

4.2 The effect of species choice

Here, we will compare the distributions of orthogroups scores using each species individually to see if the species choice has an impact on the conclusions.

```r
og_species_long <- Reduce(rbind, lapply(2:8, function(x) {

    var <- names(og_assessment)[x]
    species_name <- gsub("_.*", "", var)

    long_df <- og_assessment[, c("Orthogroups", var, "Mode")]
    names(long_df) <- c("OGs", "Score", "Mode")
    long_df$Score <- long_df$Score / max(long_df$Score, na.rm = TRUE)
    long_df$Species <- species_name

    return(long_df)
}))

og_species_long <- og_species_long[!is.na(og_species_long$Score), ]
og_species_long <- og_species_long |>
    mutate(
        Species = str_replace_all(
            Species,
            c(
                "Aarabicum" = "A. arabicum",
                "Alyrata" = "A. lyrata",
                "Athaliana" = "A. thaliana",
                "Bcarinata" = "B. carinata",
                "Chirsuta" = "C. hirsuta",
                "Crubella" = "C. rubella",
                "Sparvula" = "S. parvula"
            )
        )
    )


p_distros_by_species <- ggviolin(
    og_species_long,
    y = "Score", x = "Mode",
    orientation = "horiz", trim = TRUE,
    add = c("boxplot", "mean"), facet.by = "Species", nrow = 1,
    fill = "Mode", add.params = list(fill = "white")
) +
    scale_fill_manual(
        values = c(
            "ultra_3" = "#08519C", "ultra_2" = "#3182BD",
            "ultra_1_5" = "#6BAED6", "ultra_1" = "#BDD7E7",
            "default_3" = "#006D2C", "default_2" = "#31A354",
            "default_1_5" = "#74C476", "default_1" = "#BAE4B3"
        )
    ) +
    theme(legend.position = "none") +
    labs(
        y = "Scaled homogeneity scores", x = "OrthoFinder modes",
        title = "Distribution of OG scores for each species"
    ) +
    scale_x_discrete(
        labels = c(
            "default_1" = "Default, 1",
            "default_1_5" = "Default, 1.5",
            "default_2" = "Default, 2",
            "default_3" = "Default, 3",
            "ultra_1" = "Ultra, 1",
            "ultra_1_5" = "Ultra, 1.5",
            "ultra_2" = "Ultra, 2",
            "ultra_3" = "Ultra, 3"
        )
    ) +
    theme(axis.text.x = element_text(angle = 60, vjust = 0.5))

p_distros_by_species
```

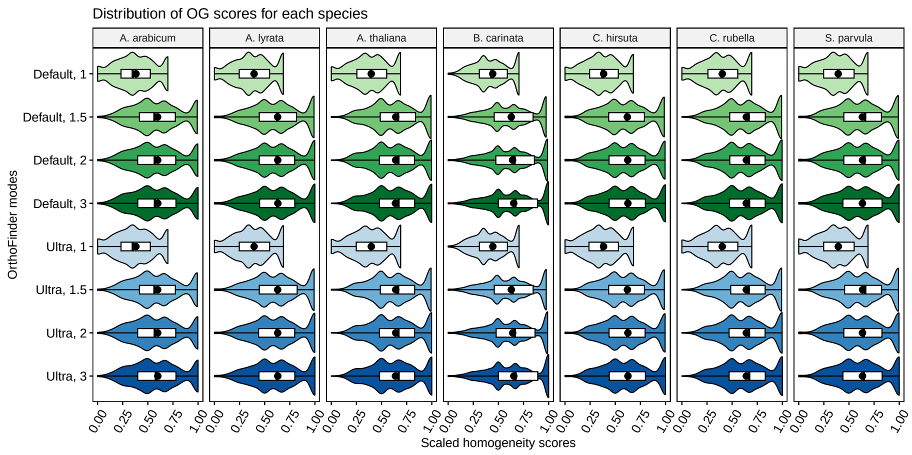


Figure 4: Distribution of orthogroup scores for each OrthoFinder run calculated for each species separately.

We conclude that the species choice does not affect the comparisons of orthogroup scores among OrthoFinder runs.

4.3 The effect of mcl inflation parameters

Here, we will explore the impact of changing mcl inflation parameters in the homogeneity of orthogroups.

```
    # Process data to include information on DIAMOND mode and mcl
    og_modes <- og_assessment %>%
        mutate(diamond = str_replace_all(Mode, "_.*", "")) %>%
        mutate(mcl = str_replace_all(Mode, c("default_" = "", "ultra_" = ""))) %>%
        mutate(mcl = str_replace_all(mcl, "_", ".")) %>%
        mutate(mcl = as.numeric(mcl))

    # Obtain P-values from Wilcoxon tests and effect sizes
    comp_mcl_default <- og_modes %>%
        filter(diamond == "default") %>%
        compare(., "Median_score ~ mcl")

    comp_mcl_default |>
        filter_comparison() |>
        knitr::kable(
            caption = "Mann-Whitney U test for differences in orthogroup scores between runs with different mcl p
            digits = 10
        )
```

Table 3: Mann-Whitney U test for differences in orthogroup scores between runs with different mcl parameters and standard DIAMOND mode. Effect sizes represent Wilcoxon effect sizes.

group1	group2	n1	n2	padj	effsize	magnitude
1	1.5	5587	16975	0	0.32420710	moderate
1	2	5587	18633	0	0.33616829	moderate
1	3	5587	19765	0	0.34147823	moderate
1.5	2	16975	18633	0	0.04366051	small
1.5	3	16975	19765	0	0.06124558	small
2	3	18633	19765	0	0.04084430	small

```
comp_mcl_ultra <- og_modes %>%
    filter(diamond == "ultra") %>%
    compare(., "Median_score ~ mcl")

comp_mcl_ultra |>
    filter_comparison() |>
    knitr::kable(
        caption = "Mann-Whitney U test for differences in orthogroup scores between runs with different mcl p
        digits = 10
    )
```

Table 4: Mann-Whitney U test for differences in orthogroup scores between runs with different mcl parameters and ultra-sensitive DIAMOND mode. Effect sizes represent Wilcoxon effect sizes.

group1	group2	n1	n2	padj	effsize	magnitude
1	1.5	5534	16898	0	0.32401575	moderate
1	2	5534	18575	0	0.33536590	moderate
1	3	5534	19738	0	0.34087185	moderate
1.5	2	16898	18575	0	0.04340346	small
1.5	3	16898	19738	0	0.06125964	small
2	3	18575	19738	0	0.04120347	small

In line with what we demonstrated in the global distributions, the Wilcoxon tests show that mcl = 1 leads to much lower homogeneity scores than all other mcl values, regardless of the DIAMOND mode. Additionally, increasing mcl values leads to increased homogeneity scores (i.e., homogeneity scores follow the order of mcl $3 > 2 > 1.5 > 1$), but differences among mcl values $>=1.5$ are negligible, as indicated by small effect sizes. Thus, low P-values could be due to large sample sizes.

Now, let's visualize the distributions.

```r
# List of comparisons to be made
mcl_comp <- list(
    c("1", "1.5"), c("1", "2"), c("1", "3"), c("1.5", "3")
)

# Plot
p_distros_mcl <- og_assessment %>%
    mutate(diamond = str_replace_all(Mode, "_.*", "")) %>%
    mutate(mcl = str_replace_all(Mode, c("default_" = "", "ultra_" = ""))) %>%
    mutate(mcl = str_replace_all(mcl, "_", ".")) %>%
    mutate(mcl = as.numeric(mcl)) %>%
    ggviolin(., x = "mcl", y = "Median_score", trim = TRUE,
             add = c("boxplot", "mean"), facet.by = "diamond",
             fill = "Mode", add.params = list(fill = "white")) +
    theme(legend.position = "none") +
    scale_fill_manual(
        values = c("ultra_3" = "#08519C", "ultra_2" = "#3182BD",
                   "ultra_1_5" = "#6BAED6", "ultra_1" = "#BDD7E7",
                   "default_3" = "#006D2C", "default_2" = "#31A354",
                   "default_1_5" = "#74C476", "default_1" = "#BAE4B3")
    ) +
    stat_compare_means(
        comparisons = mcl_comp, label = "p.signif",
        method = "wilcox.test"
    ) +
    labs(
        y = "Scaled homogeneity scores", x = "MCL inflation parameters",
        title = "Effect of MCL inflation values on orthogroup inference",
        subtitle = "Panels represent DIAMOND sensitivity modes"
    )

p_distros_mcl
```

4.4 The effect of DIAMOND mode (default vs ultra)

Here, we will investigate whether changing the DIAMOND mode (default vs ultrasensitive) in OrthoFinder affects orthogroup homogeneity.

```r
# Compare median scores
mcl1 <- og_modes %>%
    filter(mcl == 1) %>%
    compare(., "Median_score ~ diamond") |>
    filter_comparison()

mcl1_5 <- og_modes %>%
    filter(mcl == 1.5) %>%
    compare(., "Median_score ~ diamond") |>
    filter_comparison()

mcl2 <- og_modes %>%
```

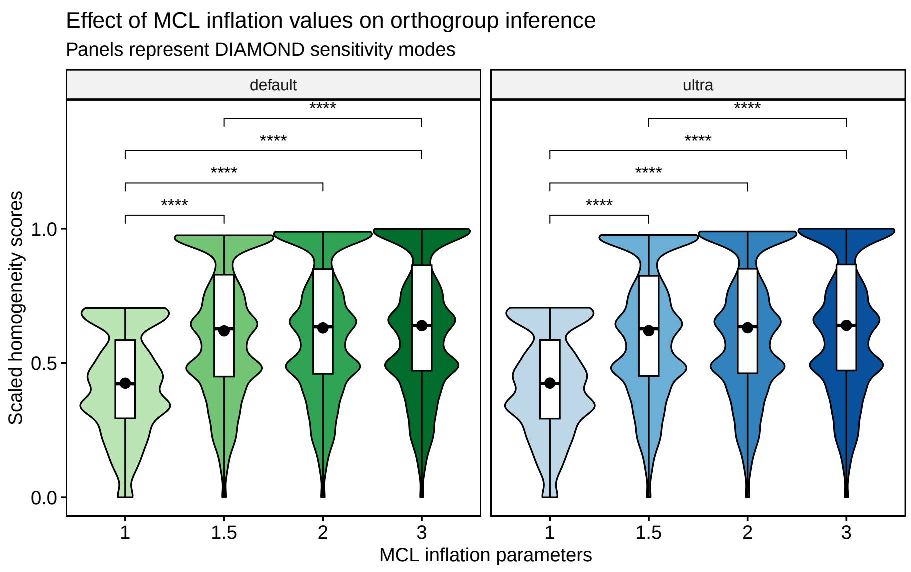


Figure 5: Effect of MCL inflation values on orthogroup scores.

```
    filter(mcl == 2) %>%
    compare(., "Median_score ~ diamond") |>
    filter_comparison()

mcl3 <- og_modes %>%
    filter(mcl == 3) %>%
    compare(., "Median_score ~ diamond") |>
    filter_comparison()

bind_rows(
    mcl1 |> mutate(mcl = 1),
    mcl1_5 |> mutate(mcl = 1.5),
    mcl2 |> mutate(mcl = 2),
    mcl3 |> mutate(mcl = 3)
) |>
    knitr::kable(
        caption = "Mann-Whitney U test for differences in orthogroup scores between runs with different DIAM
        digits = 10
    )
```

Table 5: Mann-Whitney U test for differences in orthogroup scores between runs with different DIAMOND modes for each mcl value. Effect sizes represent Wilcoxon effect sizes.

group1	group2	n1	n2	padj	effsize	magnitude	mcl
default	ultra	5587	5534	2.10e-02	0.01928759	small	1.0
default	ultra	16975	16898	1.29e-06	0.02554732	small	1.5
default	ultra	18633	18575	1.82e-08	0.02855053	small	2.0
default	ultra	19765	19738	3.00e-10	0.03113134	small	3.0

Again, we can see that there are significant P-values, but very small effect sizes, indicating no difference resulting from the DIAMOND mode. Thus, users can run the default mode of DIAMOND, which is way faster, without any loss of biological signal for orthogroup inference.

Let's visualize the distributions.

```
# Plot
p_distros_diamond <- og_modes %>%
    ggviolin(., x = "diamond", y = "Median_score", trim = TRUE,
             add = c("boxplot", "mean"), facet.by = "mcl", ncol = 4,
             fill = "Mode", add.params = list(fill = "white")) +
    theme(legend.position = "none") +
    scale_fill_manual(
        values = c("ultra_3" = "#08519C", "ultra_2" = "#3182BD",
                   "ultra_1_5" = "#6BAED6", "ultra_1" = "#BDD7E7",
                   "default_3" = "#006D2C", "default_2" = "#31A354",
                   "default_1_5" = "#74C476", "default_1" = "#BAE4B3")
    ) +
    labs(y = "Scaled homogeneity scores", x = "DIAMOND mode",
         title = "Effect of DIAMOND sensitivity mode on orthogroup inference",
         subtitle = "Panels represent MCL inflation parameters") +
    theme(plot.subtitle = ggtext::element_markdown())

p_distros_diamond
```

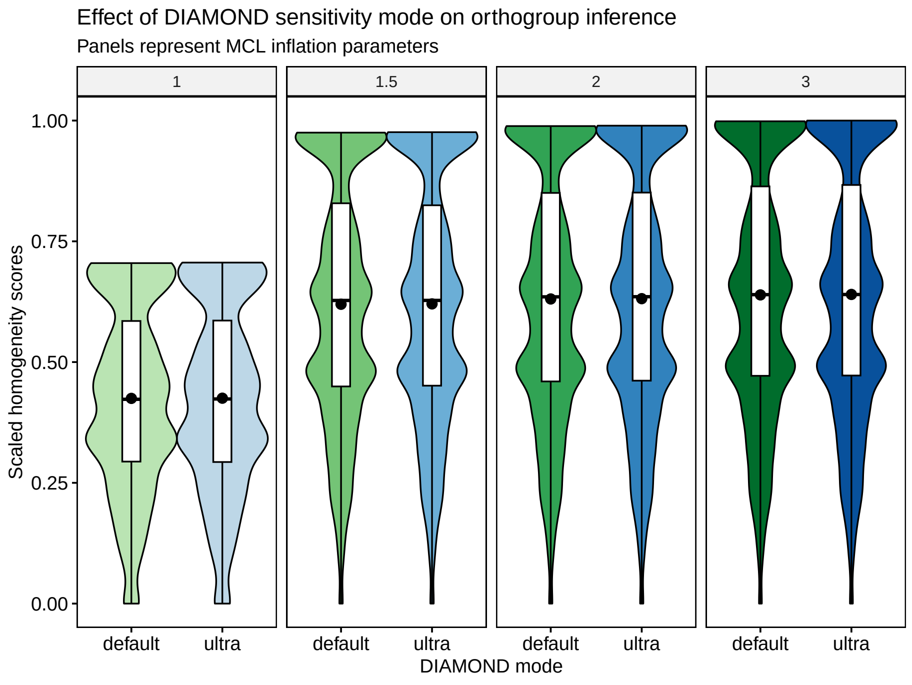


Figure 6: Effect of DIAMOND mode on orthogroup scores.

5 Functional analysis of homogeneous and heterogeneous gene families

By looking at the global distributions of homogeneity scores, we can see that all distributions have a similar shape. This pattern suggests that some gene families tend to be more homogeneous (scores close to 1), while others tend to include domains that are not shared by all members. The latter can be, for instance, rapidly evolving families that gain or lose domains at faster rates.

To explore what these groups of families contain, we will perform a functional enrichment analysis each group. First of anything, let's plot the distribution for the default OrthoFinder mode and highlight the groups.

```
# Plot distro with groups
p_distros_groups <- og_assessment %>%
    filter(Mode == "default_1_5") %>%
    ggplot(aes(x = Median_score)) +
    geom_density(fill = "grey80", color = "black") +
    ggpubr::theme_pubr() +
    labs(
        y = "Density", x = "Orthogroup scores",
        title = "Distribution of mean homogeneity scores for orthogroups",
        subtitle = "Scores for the default OrthoFinder mode"
    ) +
    geom_vline(xintercept = 0.56, color = "firebrick", linetype = 2) +
    geom_vline(xintercept = 0.87, color = "firebrick", linetype = 2)

p_distros_groups
```

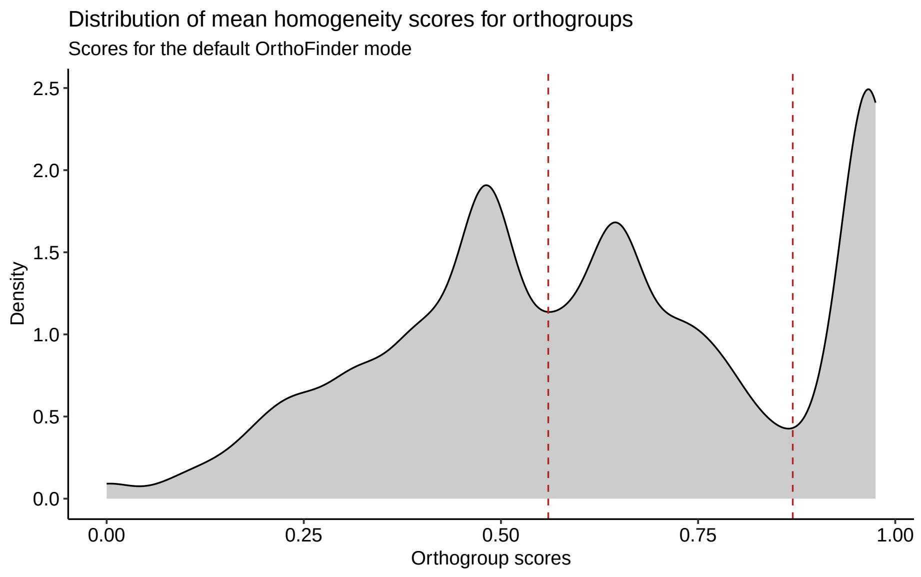


Figure 7: Distribution of mean homogeneity scores for orthogroups

Now, let's get vectors of genes in orthogroups from each of the groups highlighted in the figure above.

```r
species <- c(
    "Athaliana", "Aarabicum", "Alyrata_cvMN47", "Bcarinata_cvzd1",
    "Crubella_cvMonteGargano", "Chirsuta", "Sparvula"
)

# Get genes and orthogroups (default mode)
genes_ogs <- ogs_filtered$default_1_5

# Keep only species for which we have functional annotation info
genes_ogs <- genes_ogs[genes_ogs$Species %in% species, c(1, 3)]

# Get background genes (all genes in OGs)
background <- genes_ogs$Gene

# Find orthogroups for each group
## G1: 0 - 0.56
g1 <- og_assessment %>%
    filter(Mode == "default_1_5") %>%
    mutate(Median_score = Median_score / max(Median_score)) %>%
    filter(Median_score <= 0.56) %>%
    select(Orthogroups) %>%
    inner_join(., genes_ogs, by = c("Orthogroups" = "Orthogroup")) %>%
    pull(Gene)

## G2: 0.56 - 0.87
g2 <- og_assessment %>%
    filter(Mode == "default_1_5") %>%
    mutate(Median_score = Median_score / max(Median_score)) %>%
    filter(Median_score > 0.56 & Median_score <= 0.87) %>%
    select(Orthogroups) %>%
    inner_join(., genes_ogs, by = c("Orthogroups" = "Orthogroup")) %>%
    pull(Gene)


## G3: 0.87 - 1
g3 <- og_assessment %>%
    filter(Mode == "default_1_5") %>%
    mutate(Median_score = Median_score / max(Median_score)) %>%
    filter(Median_score > 0.87) %>%
    select(Orthogroups) %>%
    inner_join(., genes_ogs, by = c("Orthogroups" = "Orthogroup")) %>%
    pull(Gene)
```

Next, we need to get functional annotation from PLAZA.

```r
options(timeout = 6000)
plaza_species <- c("ath", "aar", "aly", "bca", "cru", "chi", "spa")

# GO annotation
bgo <- "https://ftp.psb.ugent.be/pub/plaza/plaza_public_dicots_05/GO/"
go <- lapply(plaza_species, function(x) {
    y <- read_annotation(paste0(bgo, "go.", x, ".csv.gz"), c(1, 3, 8))
    term2gene <- y[, c(2, 1)] %>% distinct(., .keep_all = TRUE)
    term2name <- y[, c(2, 3)] %>% distinct(., .keep_all = TRUE)
    res <- list(
        TERM2GENE = as.data.frame(term2gene),
        TERM2NAME = as.data.frame(term2name)
    )
    return(res)
})
go_gene <- Reduce(rbind, lapply(go, function(x) return(x$TERM2GENE)))
go_des <- Reduce(rbind, lapply(go, function(x) return(x$TERM2NAME)))

## Remove non-BP terms
ath_bp <- file.path(tempdir(), "ath_bp.rds")
download.file(
    "https://jokergoo.github.io/rGREAT_genesets/genesets/bp_athaliana_eg_gene_go_genesets.rds",
    destfile = ath_bp
)
gobp <- readRDS(ath_bp)
gobp <- names(gobp)
go_gene <- go_gene[go_gene$Annotation %in% gobp, ]
go_des <- go_des[go_des$Annotation %in% gobp, ]
rm(gobp)

# MapMan annotation
bmm <- "https://ftp.psb.ugent.be/pub/plaza/plaza_public_dicots_05/MapMan/"
mm <- lapply(plaza_species, function(x) {
    y <- read_annotation(paste0(bmm, "mapman.", x, ".csv.gz"), c(3:5))
    term2gene <- y[, c(2, 1)] %>% distinct(., .keep_all = TRUE)
    term2name <- y[, c(2, 3)] %>% distinct(., .keep_all = TRUE)
    res <- list(
        TERM2GENE = as.data.frame(term2gene),
        TERM2NAME = as.data.frame(term2name)
    )
    return(res)
})
mm_gene <- Reduce(rbind, lapply(mm, function(x) return(x$TERM2GENE)))
mm_des <- Reduce(rbind, lapply(mm, function(x) return(x$TERM2NAME))) %>%
    mutate(desc = str_replace_all(desc, ".*\\.", ""))

# InterPro
bi <- "https://ftp.psb.ugent.be/pub/plaza/plaza_public_dicots_05/InterPro/"
ip <- lapply(plaza_species, function(x) {
    y <- read_annotation(paste0(bi, "interpro.", x, ".csv.gz"), c(1, 3, 4))
    term2gene <- y[, c(2, 1)] %>% distinct(., .keep_all = TRUE)
    term2name <- y[, c(2, 3)] %>% distinct(., .keep_all = TRUE)
    res <- list(
        TERM2GENE = as.data.frame(term2gene),
        TERM2NAME = as.data.frame(term2name)
    )
    return(res)
```

```r
})
ip_gene <- Reduce(rbind, lapply(ip, function(x) return(x$TERM2GENE)))
ip_des <- Reduce(rbind, lapply(ip, function(x) return(x$TERM2NAME)))
```

Now, we can finally perform the enrichment analyses.

```r
# Perform enrichment analyses
library(clusterProfiler)

tgene <- list(
    GO = go_gene,
    MapMan = mm_gene,
    InterPro = ip_gene
)
tname <- list(
    GO = go_des,
    MapMan = mm_des,
    InterPro = ip_des
)

## G1
g1_sea <- Reduce(rbind, lapply(seq_along(tgene), function(x) {
    return(as.data.frame(enricher(
        g1, universe = background,
        TERM2GENE = tgene[[x]], TERM2NAME = tname[[x]]
    ))[, 1:6])
}))

## G2
g2_sea <- Reduce(rbind, lapply(seq_along(tgene), function(x) {
    return(as.data.frame(enricher(
        g2, universe = background,
        TERM2GENE = tgene[[x]], TERM2NAME = tname[[x]]
    ))[, 1:6])
}))

## G3
g3_sea <- Reduce(rbind, lapply(seq_along(tgene), function(x) {
    return(as.data.frame(enricher(
        g3, universe = background,
        TERM2GENE = tgene[[x]], TERM2NAME = tname[[x]]
    ))[, 1:6])
}))

# Combine SEA results in a single data frame and export it as a .tsv file
## Combine data frames
sea_res <- rbind(
    g1_sea %>% mutate(group = "G1"),
    g2_sea %>% mutate(group = "G2"),
    g3_sea %>% mutate(group = "G3")
)
```

```r
## Export .tsv
write_tsv(
    sea_res,
    file = here("products", "tables", "enrichment_bygroup.tsv")
)
```

The complete enrichment results are stored in the table `enrichment_bygroup.tsv`. To make visualization and interpretation easier, we will perform semantic similarity analysis to group redundant terms and get a global view of processes associated with each cluster.

Here, we will only use GO terms from the category "Biological Process".

```r
# Semantic similarity analysis for GO-BP terms
## G1
g1_summary <- pairwise_termsim(enricher(
    g1, universe = background,
    TERM2GENE = go_gene, TERM2NAME = go_des
))

## G2
g2_summary <- pairwise_termsim(enricher(
    g2, universe = background,
    TERM2GENE = go_gene, TERM2NAME = go_des
))

## G3
g3_summary <- pairwise_termsim(enricher(
    g3, universe = background,
    TERM2GENE = go_gene, TERM2NAME = go_des
))

# Save objects
save(
    g1_summary, compress = "xz",
    file = here("products", "result_files", "g1_summary.rda")
)

save(
    g2_summary, compress = "xz",
    file = here("products", "result_files", "g2_summary.rda")
)

save(
    g3_summary, compress = "xz",
    file = here("products", "result_files", "g3_summary.rda")
)
```

Now, let's plot the results.

```r
# Tree plot
p_tree_g1 <- treeplot(g1_summary, nWords = 0) +
    ggsci::scale_fill_jama() +
    ggtitle("Group 1")
p_tree_g1$layers[[4]] <- NULL


p_tree_g2 <- treeplot(g2_summary, nCluster = 7, nWords = 0) +
    ggsci::scale_fill_jama() +
    ggtitle("Group 2")
p_tree_g2$layers[[4]] <- NULL


p_tree_g3 <- treeplot(g3_summary, nWords = 0) +
    ggsci::scale_fill_jama() +
    ggtitle("Group 3")
p_tree_g3$layers[[4]] <- NULL


# Replace P.adj with -log10(P.adj)
p_tree_g1$data$color <- -log10(p_tree_g1$data$color)
p_tree_g2$data$color <- -log10(p_tree_g2$data$color)
p_tree_g3$data$color <- -log10(p_tree_g3$data$color)


# Combine plots in one, with shared legends
rcol <- range(
    c(
        p_tree_g1$data$color, p_tree_g2$data$color, p_tree_g3$data$color
    ),
    na.rm = TRUE
)
rsize <- range(
    c(
        p_tree_g1$data$count, p_tree_g2$data$count, p_tree_g2$data$count
    ),
    na.rm = TRUE
)

wrap_plots(p_tree_g1, p_tree_g2, p_tree_g3) +
    plot_layout(guides = "collect") &
    scale_color_continuous(name = "-Log10(P)", limits = signif(rcol, 2)) &
    scale_size_continuous(name = "Gene count", limits = rsize) &
    theme(legend.position = "bottom")
```

```
# Dot plot
p_dot_g1 <- dotplot(g1_summary, showCategory = 20) + ggtitle("Group 1")
p_dot_g2 <- dotplot(g2_summary, showCategory = 20) + ggtitle("Group 2")
p_dot_g3 <- dotplot(g3_summary, showCategory = 20) + ggtitle("Group 3")

# Replace P.adj with -log10(P.adj)
p_dot_g1$data$p.adjust <- -log10(p_dot_g1$data$p.adjust)
p_dot_g2$data$p.adjust <- -log10(p_dot_g2$data$p.adjust)
p_dot_g3$data$p.adjust <- -log10(p_dot_g3$data$p.adjust)
```

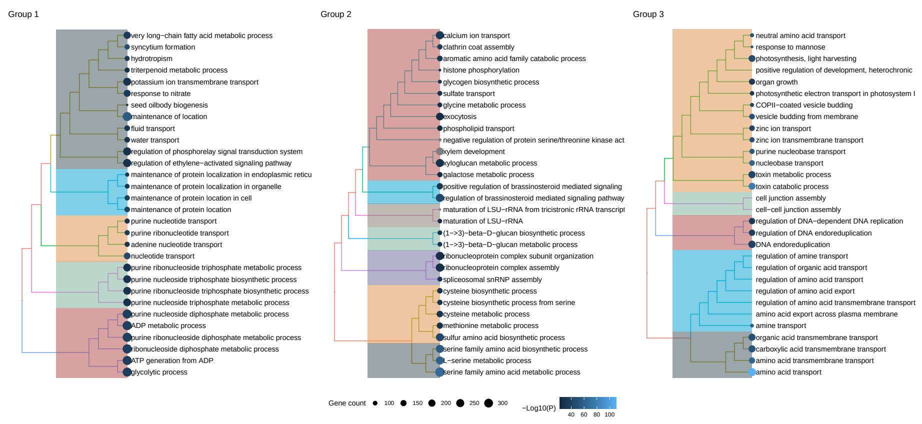


Figure 8: Tree plot of functional terms associated with each orthogroup cluster.

```r
# Combine plots in one, keep shared legend
rcol <- range(
    c(
        p_dot_g1$data$p.adjust, p_dot_g2$data$p.adjust,
        p_dot_g3$data$p.adjust
    ),
    na.rm = TRUE
)
rsize <- range(
    c(
        p_dot_g1$data$Count, p_dot_g2$data$Count, p_dot_g3$data$Count
    ),
    na.rm = TRUE
)

wrap_plots(p_dot_g1, p_dot_g2, p_dot_g3) +
    plot_layout(guides = "collect") &
    scale_color_continuous(name = "-Log10(P)", limits = signif(rcol, 2)) &
    scale_size_continuous(name = "Gene count", limits = rsize) &
    theme(legend.position = "bottom")
```

The plots show that genes associated to particular biological processes tend to be clustered in the same orthogroup (group 3, scores closer to 1), while genes associated to other biological processes tend to be more dispersed across orthogroups (groups 1 and 2, scores closer to 1), possibly because they are evolving faster and, hence, have lower sequence similarity among themselves. In details, these genes and processes are:

- **Group 1:** ATP production, water and K+ transport, seed oilbody biogenesis, and response to nitrate and ethylene.
- **Group 2:** sulfur amino acid metabolsm, spliceosome biogenesis, beta-1,3-glucan biosynthesis, response to brassinosteroids, xylem development, exocytosis, and calcium and sulfate transport.
- **Group 3:** photosynthesis, zinc and amino acid transport, DNA replication, endocytosis, cell-cell junction assembly, and toxin catabolism.

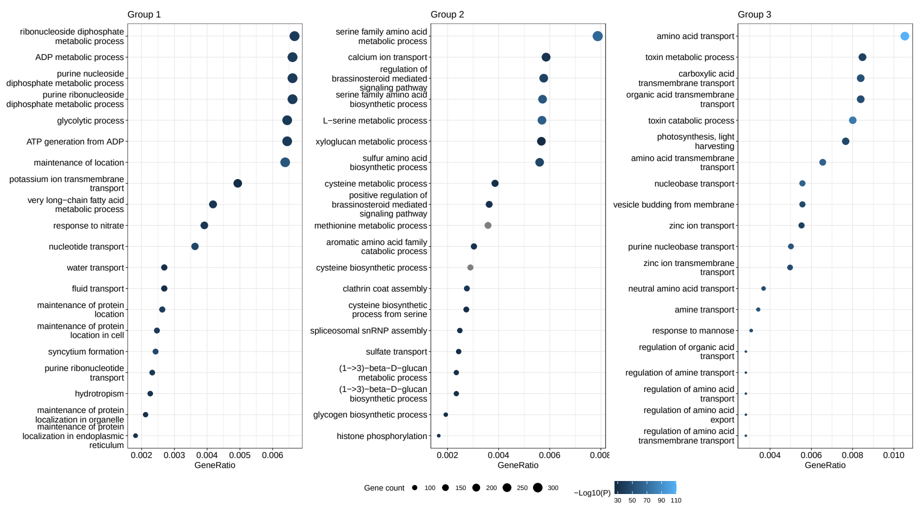


Figure 9: Dotplot of functional terms associated with each orthogroup cluster.

6 Is there an association between OG score and OG gene length?

Emms and Kelly (2015) have demonstrated a gene length bias that influences the accuracy of orthogroup detection. This is because short sequences cannot produce large bit scores or low e-values, and long sequences produce many hits with scores better than those for the best hits of short sequences (Emms and Kelly 2015). OrthoFinder implements a score transform that claims to eliminate such bias. But does it remove the bias completely?

To answer this question, we will use homogeneity scores for the default OrthoFinder run (default DIAMOND mode, mcl = 1.5).

First of all, let's calculate the mean and median gene length for each orthogroup.

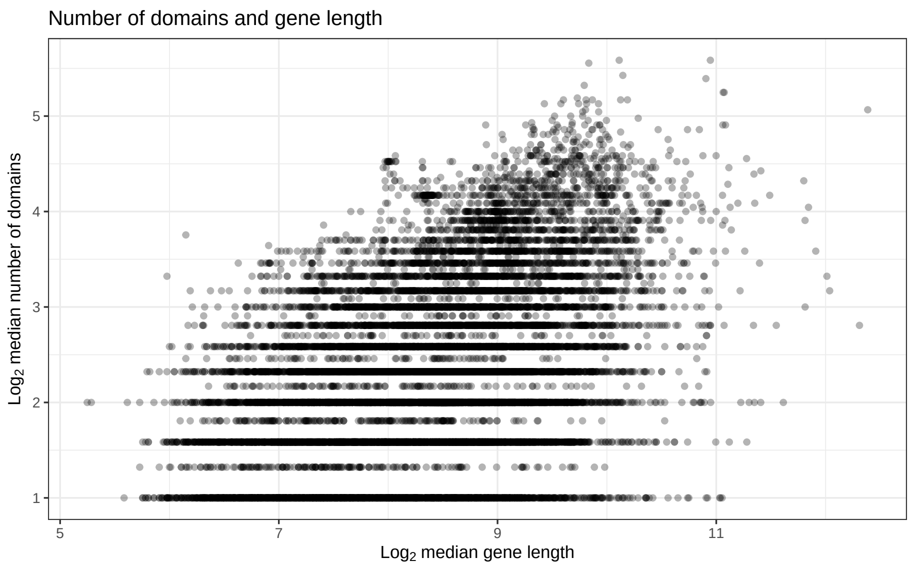
```r
# Combine proteomes into a single AAStringSet object and clean gene names
names(brassicaceae_proteomes) <- NULL
proteomes <- do.call(c, brassicaceae_proteomes)
rm(brassicaceae_proteomes)

names(proteomes) <- gsub("\\\t.*", "", names(proteomes))
names(proteomes) <- gsub(" .*", "", names(proteomes))
names(proteomes) <- gsub("\\.[0-9]$", "", names(proteomes))
names(proteomes) <- gsub("\\.[0-9]\\.p$", "", names(proteomes))
names(proteomes) <- gsub("\\.t[0-9]$", "", names(proteomes))
names(proteomes) <- gsub("\\.g$", "", names(proteomes))


# Load only orthogroups from the default OrthoFinder run
og <- read_orthogroups(file.path(tempdir(), "Orthogroups_default_1_5.tsv")) %>%
    mutate(Gene = str_replace_all(
        Gene, c(
            "\\\t.*" = "",
            "\\.[0-9]$" = "",
            "\\.[0-9]\\.p$" = "",
            "\\.t[0-9]$" = "",
            "\\.g$" = ""
        )
    )) %>%
    dplyr::select(Orthogroup, Gene)

# Calculate mean gene lengths for each orthogroup
gene_lengths <- data.frame(
    Gene = names(proteomes),
    Length = Biostrings::width(proteomes)
)

og_gene_lengths <- og %>%
    inner_join(., gene_lengths) %>%
    group_by(Orthogroup) %>%
    summarise(
        mean_length = mean(Length),
        median_length = median(Length)
    )

# Add homogeneity scores to data frame of mean gene length per orthogroup
og_length_and_scores <- og_assessment %>%
    dplyr::filter(Mode == "default_1_5") %>%
    dplyr::select(Orthogroups, Mean_H, Median_H) %>%
    inner_join(., og_gene_lengths, by = c("Orthogroups" = "Orthogroup"))
```

Now, since the number of domains in a protein correlates with its length, let's also calculate the median number of domains in an orthogroup.

```
# Calculate median number of domains for each orthogroup
og_domain_count <- Reduce(rbind, interpro) |>
    dplyr::count(Gene) |>
    inner_join(og, by = "Gene") |>
    group_by(Orthogroup) |>
    summarise(
        median_ndomains = median(n)
    )

og_length_and_scores <- left_join(
    og_length_and_scores, og_domain_count, by = c("Orthogroups" = "Orthogroup")
)

# Save data
save(
    og_length_and_scores,
    file = here("products", "result_files", "og_length_and_scores.rda"),
    compress = "xz"
)
```

Next, we will investigate if the number of domains can be a confounder in associations between the orthogroup score and gene length.

```r
# Explore associations between the number of domains and gene length
p_length_domains <- ggplot(
    og_length_and_scores,
    aes(y = log2(median_ndomains + 1), x = log2(median_length + 1))
) +
    geom_point(alpha = 0.3) +
    theme_bw() +
    labs(
        title = "Number of domains and gene length",
        x = expression(Log[2] ~ "median gene length"),
        y = expression(Log[2] ~ "median number of domains")
    )

cor_length_domains <- cor.test(
    log2(og_length_and_scores$median_length + 1),
    log2(og_length_and_scores$median_ndomains + 1),
    method = "spearman",
    exact = FALSE
)

# Show plot and correlation test statistics
p_length_domains
```



```
cor_length_domains
##
##  Spearman's rank correlation rho
##
## data:  log2(og_length_and_scores$median_length + 1) and log2(og_length_and_scores$median_ndomains + 1)
## S = 4.7564e+11, p-value < 2.2e-16
## alternative hypothesis: true rho is not equal to 0
## sample estimates:
##       rho
## 0.4165488
```

The figure and test above show that there is indeed a moderate correlation $(\rho \approx 0.4, P < 0.001)$ between gene length and number of domains. Because of that, we will use partial Spearman's correlation to measure the association between orthogroup scores and gene length while controlling for the number of domains.

```
# Calculate partial Spearman's correlations
## Without accounting for the number of domains
cor1 <- ppcor::pcor(
    data.frame(
        Length = log2(og_length_and_scores$median_length + 1),
        Score = log2(og_length_and_scores$Median_H + 1)
    ),
    method = "spearman"
)
cor1
## $estimate
##            Length      Score
## Length  1.0000000 -0.1903208
## Score  -0.1903208  1.0000000
##
## $p.value
##               Length         Score
## Length  0.000000e+00 3.370622e-138
## Score  3.370622e-138  0.000000e+00
##
## $statistic
##           Length     Score
## Length   0.00000 -25.25673
## Score  -25.25673   0.00000
##
## $n
## [1] 16975
##
## $gp
## [1] 0
##
## $method
## [1] "spearman"

## Accounting for the number of domains
cor2 <- ppcor::pcor.test(
    log2(og_length_and_scores$median_length + 1),
    log2(og_length_and_scores$Median_H + 1),
    log2(og_length_and_scores$median_ndomains + 1),
    method = "spearman"
)
cor2
```

```
##     estimate      p.value statistic     n gp   Method
## 1 0.08593287 3.417431e-29  11.23661 16975  1 spearman
```

The tests show a weak correlation between orthogroup scores and gene length. When the number of domains is included as a covariate, we find no correlation at all, indicating that OrthoFinder's normalization score is effective.

Finally, let's plot the data and add the test statistics.

```
p_association_length_homogeneity <- og_length_and_scores %>%
    mutate(
        logH = log10(Median_H + 1),
        logLength = log10(median_length + 1)
    ) %>%
    ggscatter(
        ., x = "logLength", y = "logH", alpha = 0.3,
        color = "black", size = 1
    ) +
    annotate(
        "text",
        x = 1.71, y = 0.055,
        label = paste(
            "rho", "==", signif(cor1$estimate[1, 2], 2)
        ),
        parse = TRUE
    ) +
    annotate(
        "text",
        x = 1.71, y = 0.035,
        label = paste(
            "rho[partial]", "==", signif(cor2$estimate, 2)
        ),
        parse = TRUE
    ) +
    annotate(
        "text",
        x = 1.71, y = 0.015,
        label = paste("P", "<", 2.2e-16), parse = TRUE
    ) +
    labs(
        title = "Relationship between OG score and gene length",
        x = expression(Log[10] ~ "median gene length"),
        y = expression(Log[10] ~ "median homogeneity score")
    )

p_association_length_homogeneity
```

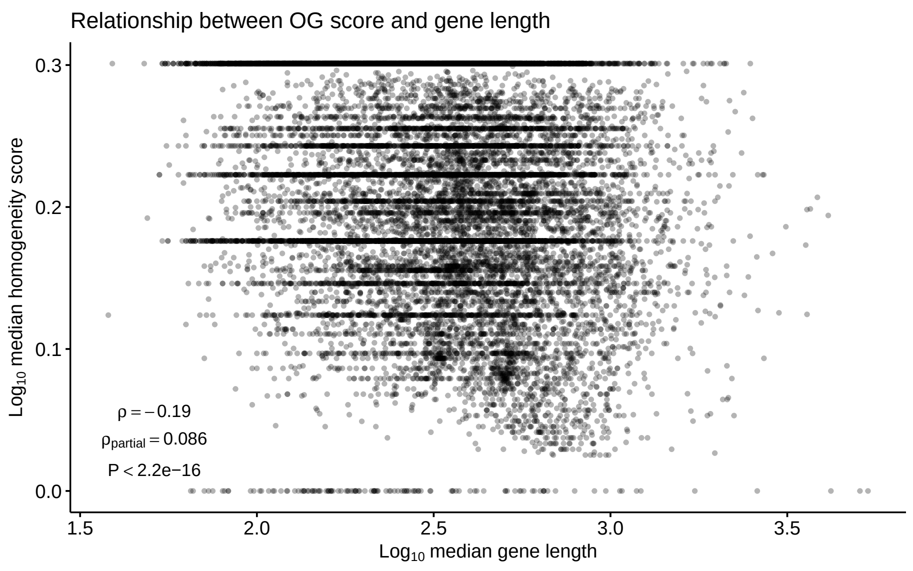


Figure 10: Relationship between sequence length and orthogroup scores.

Session info

This document was created under the following conditions:

```
sessioninfo::session_info()
## - Session info ---------------------------------------------------------------
##  setting  value
##  version  R version 4.3.0 (2023-04-21)
##  os       Ubuntu 20.04.5 LTS
##  system   x86_64, linux-gnu
##  ui       X11
##  language (EN)
##  collate  en_US.UTF-8
##  ctype    en_US.UTF-8
##  tz       Europe/Brussels
##  date     2023-08-08
##  pandoc   3.1.1 @ /usr/lib/rstudio/resources/app/bin/quarto/bin/tools/ (via rmarkdown)
##
## - Packages -------------------------------------------------------------------
##  package          * version   date (UTC) lib source
##  abind              1.4-5     2016-07-21 [1] CRAN (R 4.3.0)
##  AnnotationDbi      1.62.0    2023-04-25 [1] Bioconductor
##  ape                5.7-1     2023-03-13 [1] CRAN (R 4.3.0)
##  aplot              0.1.10    2023-03-08 [1] CRAN (R 4.3.0)
##  backports          1.4.1     2021-12-13 [1] CRAN (R 4.3.0)
##  beeswarm           0.4.0     2021-06-01 [1] CRAN (R 4.3.0)
##  Biobase            2.60.0    2023-04-25 [1] Bioconductor
##  BiocGenerics       0.46.0    2023-04-25 [1] Bioconductor
##  BiocManager        1.30.21.1 2023-07-18 [1] CRAN (R 4.3.0)
##  BiocParallel       1.34.0    2023-04-25 [1] Bioconductor
##  BiocStyle        * 2.29.1    2023-08-04 [1] Github (Bioconductor/BiocStyle@7c0e093)
##  Biostrings         2.68.0    2023-04-25 [1] Bioconductor
##  bit                4.0.5     2022-11-15 [1] CRAN (R 4.3.0)
##  bit64              4.0.5     2020-08-30 [1] CRAN (R 4.3.0)
##  bitops             1.0-7     2021-04-24 [1] CRAN (R 4.3.0)
##  blob               1.2.4     2023-03-17 [1] CRAN (R 4.3.0)
##  bookdown           0.34      2023-05-09 [1] CRAN (R 4.3.0)
##  broom              1.0.4     2023-03-11 [1] CRAN (R 4.3.0)
##  cachem             1.0.8     2023-05-01 [1] CRAN (R 4.3.0)
##  car                3.1-2     2023-03-30 [1] CRAN (R 4.3.0)
##  carData            3.0-5     2022-01-06 [1] CRAN (R 4.3.0)
##  cli                3.6.1     2023-03-23 [1] CRAN (R 4.3.0)
##  clusterProfiler  * 4.8.1     2023-05-03 [1] Bioconductor
##  codetools          0.2-19    2023-02-01 [4] CRAN (R 4.2.2)
##  cogeqc           * 1.4.0     2023-04-25 [1] Bioconductor
##  coin               1.4-2     2021-10-08 [1] CRAN (R 4.3.0)
##  colorspace         2.1-0     2023-01-23 [1] CRAN (R 4.3.0)
##  commonmark         1.9.0     2023-03-17 [1] CRAN (R 4.3.0)
##  cowplot            1.1.1     2020-12-30 [1] CRAN (R 4.3.0)
##  crayon             1.5.2     2022-09-29 [1] CRAN (R 4.3.0)
##  data.table         1.14.8    2023-02-17 [1] CRAN (R 4.3.0)
##  DBI                1.1.3     2022-06-18 [1] CRAN (R 4.3.0)
##  digest             0.6.33    2023-07-07 [1] CRAN (R 4.3.0)
##  DOSE               3.26.1    2023-05-03 [1] Bioconductor
##  downloader         0.4       2015-07-09 [1] CRAN (R 4.3.0)
##  dplyr            * 1.1.2     2023-04-20 [1] CRAN (R 4.3.0)
##  enrichplot       * 1.20.0    2023-04-25 [1] Bioconductor
##  evaluate           0.21      2023-05-05 [1] CRAN (R 4.3.0)
##  fansi              1.0.4     2023-01-22 [1] CRAN (R 4.3.0)
##  farver             2.1.1     2022-07-06 [1] CRAN (R 4.3.0)
##  fastmap            1.1.1     2023-02-24 [1] CRAN (R 4.3.0)
##  fastmatch          1.1-3     2021-07-23 [1] CRAN (R 4.3.0)
##  fgsea              1.26.0    2023-04-25 [1] Bioconductor
##  forcats          * 1.0.0     2023-01-29 [1] CRAN (R 4.3.0)
##  generics           0.1.3     2022-07-05 [1] CRAN (R 4.3.0)
##  GenomeInfoDb       1.36.0    2023-04-25 [1] Bioconductor
##  GenomeInfoDbData   1.2.10    2023-04-28 [1] Bioconductor
##  ggbeeswarm         0.7.2     2023-04-29 [1] CRAN (R 4.3.0)
##  ggforce            0.4.1     2022-10-04 [1] CRAN (R 4.3.0)
##  ggfun              0.0.9     2022-11-21 [1] CRAN (R 4.3.0)
##  ggnewscale         0.4.8     2022-10-06 [1] CRAN (R 4.3.0)
##  ggplot2          * 3.4.1     2023-02-10 [1] CRAN (R 4.3.0)
##  ggplotify          0.1.0     2021-09-02 [1] CRAN (R 4.3.0)
##  ggpubr           * 0.6.0     2023-02-10 [1] CRAN (R 4.3.0)
##  ggraph             2.1.0     2022-10-09 [1] CRAN (R 4.3.0)
##  ggrepel            0.9.3     2023-02-03 [1] CRAN (R 4.3.0)
##  ggsci              3.0.0     2023-03-08 [1] CRAN (R 4.3.0)
##  ggsignif           0.6.4     2022-10-13 [1] CRAN (R 4.3.0)
##  ggtext             0.1.2     2022-09-16 [1] CRAN (R 4.3.0)
##  ggtree             3.8.0     2023-04-25 [1] Bioconductor
##  glue               1.6.2     2022-02-24 [1] CRAN (R 4.3.0)
##  GO.db               3.17.0    2023-05-02 [1] Bioconductor
##  GOSemSim            2.26.0    2023-04-25 [1] Bioconductor
##  graphlayouts        1.0.0     2023-05-01 [1] CRAN (R 4.3.0)
##  gridExtra           2.3       2017-09-09 [1] CRAN (R 4.3.0)
##  gridGraphics        0.5-1     2020-12-13 [1] CRAN (R 4.3.0)
##  gridtext            0.1.5     2022-09-16 [1] CRAN (R 4.3.0)
##  gson                0.1.0     2023-03-07 [1] CRAN (R 4.3.0)
##  gtable              0.3.3     2023-03-21 [1] CRAN (R 4.3.0)
##  HDO.db              0.99.1    2023-06-20 [1] Bioconductor
##  here              * 1.0.1     2020-12-13 [1] CRAN (R 4.3.0)
##  hms                 1.1.3     2023-03-21 [1] CRAN (R 4.3.0)
##  htmltools           0.5.5     2023-03-23 [1] CRAN (R 4.3.0)
##  httr                1.4.5     2023-02-24 [1] CRAN (R 4.3.0)
##  igraph              1.4.2     2023-04-07 [1] CRAN (R 4.3.0)
##  IRanges             2.34.0    2023-04-25 [1] Bioconductor
##  jsonlite            1.8.7     2023-06-29 [1] CRAN (R 4.3.0)
##  KEGGREST            1.40.0    2023-04-25 [1] Bioconductor
##  knitr               1.43      2023-05-25 [1] CRAN (R 4.3.0)
##  labeling            0.4.2     2020-10-20 [1] CRAN (R 4.3.0)
##  lattice             0.20-45   2021-09-22 [4] CRAN (R 4.2.0)
##  lazyeval            0.2.2     2019-03-15 [1] CRAN (R 4.3.0)
##  libcoin             1.0-9     2021-09-27 [1] CRAN (R 4.3.0)
##  lifecycle           1.0.3     2022-10-07 [1] CRAN (R 4.3.0)
##  lubridate         * 1.9.2     2023-02-10 [1] CRAN (R 4.3.0)
##  magrittr            2.0.3     2022-03-30 [1] CRAN (R 4.3.0)
##  markdown            1.6       2023-04-07 [1] CRAN (R 4.3.0)
##  MASS                7.3-58.2  2023-01-23 [4] CRAN (R 4.2.2)
##  Matrix              1.5-1     2022-09-13 [4] CRAN (R 4.2.1)
##  matrixStats         1.0.0     2023-06-02 [1] CRAN (R 4.3.0)
##  memoise             2.0.1     2021-11-26 [1] CRAN (R 4.3.0)
##  modeltools          0.2-23    2020-03-05 [1] CRAN (R 4.3.0)
##  multcomp            1.4-25    2023-06-20 [1] CRAN (R 4.3.0)
##  munsell             0.5.0     2018-06-12 [1] CRAN (R 4.3.0)
##  mvtnorm             1.1-3     2021-10-08 [1] CRAN (R 4.3.0)
##  nlme                3.1-162   2023-01-31 [4] CRAN (R 4.2.2)
##  patchwork         * 1.1.2     2022-08-19 [1] CRAN (R 4.3.0)
##  pillar              1.9.0     2023-03-22 [1] CRAN (R 4.3.0)
##  pkgconfig           2.0.3     2019-09-22 [1] CRAN (R 4.3.0)
##  plyr                1.8.8     2022-11-11 [1] CRAN (R 4.3.0)
##  png                 0.1-8     2022-11-29 [1] CRAN (R 4.3.0)
##  polyclip            1.10-4    2022-10-20 [1] CRAN (R 4.3.0)
##  ppcor               1.1       2015-12-03 [1] CRAN (R 4.3.0)
##  purrr             * 1.0.1     2023-01-10 [1] CRAN (R 4.3.0)
##  qvalue              2.32.0    2023-04-25 [1] Bioconductor
##  R6                  2.5.1     2021-08-19 [1] CRAN (R 4.3.0)
##  RColorBrewer        1.1-3     2022-04-03 [1] CRAN (R 4.3.0)
##  Rcpp                1.0.10    2023-01-22 [1] CRAN (R 4.3.0)
##  RCurl               1.98-1.12 2023-03-27 [1] CRAN (R 4.3.0)
##  readr             * 2.1.4     2023-02-10 [1] CRAN (R 4.3.0)
##  reshape2            1.4.4     2020-04-09 [1] CRAN (R 4.3.0)
##  rlang               1.1.1     2023-04-28 [1] CRAN (R 4.3.0)
## rmarkdown         2.23     2023-07-01 [1] CRAN (R 4.3.0)
## rprojroot         2.0.3    2022-04-02 [1] CRAN (R 4.3.0)
## RSQLite           2.3.1    2023-04-03 [1] CRAN (R 4.3.0)
## rstatix         * 0.7.2    2023-02-01 [1] CRAN (R 4.3.0)
## rstudioapi        0.14     2022-08-22 [1] CRAN (R 4.3.0)
## S4Vectors         0.38.0   2023-04-25 [1] Bioconductor
## sandwich          3.0-2    2022-06-15 [1] CRAN (R 4.3.0)
## scales            1.2.1    2022-08-20 [1] CRAN (R 4.3.0)
## scatterpie        0.2.1    2023-06-07 [1] CRAN (R 4.3.0)
## sessioninfo       1.2.2    2021-12-06 [1] CRAN (R 4.3.0)
## shadowtext        0.1.2    2022-04-22 [1] CRAN (R 4.3.0)
## stringi           1.7.12   2023-01-11 [1] CRAN (R 4.3.0)
## stringr         * 1.5.0    2022-12-02 [1] CRAN (R 4.3.0)
## survival          3.5-3    2023-02-12 [4] CRAN (R 4.2.2)
## TH.data           1.1-2    2023-04-17 [1] CRAN (R 4.3.0)
## tibble          * 3.2.1    2023-03-20 [1] CRAN (R 4.3.0)
## tidygraph         1.2.3    2023-02-01 [1] CRAN (R 4.3.0)
## tidyr           * 1.3.0    2023-01-24 [1] CRAN (R 4.3.0)
## tidyselect        1.2.0    2022-10-10 [1] CRAN (R 4.3.0)
## tidytree          0.4.2    2022-12-18 [1] CRAN (R 4.3.0)
## tidyverse       * 2.0.0    2023-02-22 [1] CRAN (R 4.3.0)
## timechange        0.2.0    2023-01-11 [1] CRAN (R 4.3.0)
## treeio            1.24.1   2023-05-31 [1] Bioconductor
## tweenr            2.0.2    2022-09-06 [1] CRAN (R 4.3.0)
## tzdb              0.3.0    2022-03-28 [1] CRAN (R 4.3.0)
## utf8              1.2.3    2023-01-31 [1] CRAN (R 4.3.0)
## vctrs             0.6.3    2023-06-14 [1] CRAN (R 4.3.0)
## vipor             0.4.5    2017-03-22 [1] CRAN (R 4.3.0)
## viridis           0.6.2    2021-10-13 [1] CRAN (R 4.3.0)
## viridisLite       0.4.2    2023-05-02 [1] CRAN (R 4.3.0)
## withr             2.5.0    2022-03-03 [1] CRAN (R 4.3.0)
## xfun              0.39     2023-04-20 [1] CRAN (R 4.3.0)
## xml2              1.3.4    2023-04-27 [1] CRAN (R 4.3.0)
## XVector           0.40.0   2023-04-25 [1] Bioconductor
## yaml              2.3.7    2023-01-23 [1] CRAN (R 4.3.0)
## yulab.utils       0.0.6    2022-12-20 [1] CRAN (R 4.3.0)
## zlibbioc          1.46.0   2023-04-25 [1] Bioconductor
## zoo               1.8-12   2023-04-13 [1] CRAN (R 4.3.0)
##
## [1] /home/faalm/R/x86_64-pc-linux-gnu-library/4.3
## [2] /usr/local/lib/R/site-library
## [3] /usr/lib/R/site-library
## [4] /usr/lib/R/library
##
## ------------------------------------------------------------------------------
```

References

Emms, David M, and Steven Kelly. 2015. "OrthoFinder: Solving Fundamental Biases in Whole Genome Comparisons Dramatically Improves Orthogroup Inference Accuracy." *Genome Biology* 16 (1): 1–14.

———. 2019. "OrthoFinder: Phylogenetic Orthology Inference for Comparative Genomics." *Genome Biology* 20 (1): 1–14.

Supplementary Text S5: On overclustering correction

Fabricio Almeida-Silva[1,2] ***and Yves Van de Peer***[1,2,3,4]

[1]VIB-UGent Center for Plant Systems Biology, Ghent, Belgium
[2]Department of Plant Biotechnology and Bioinformatics, Ghent University, Ghent, Belgium
[3]College of Horticulture, Academy for Advanced Interdisciplinary Studies, Nanjing Agricultural University, Nanjing, China
[4]Center for Microbial Ecology and Genomics, Department of Biochemistry, Genetics and Microbiology, University of Pretoria, Pretoria, South Africa

6 October 2023

Contents

1 Overview 2
2 Data acquisition 2
3 Validating the overclustering correction 3
Session info 6
References 9

1 Overview

Here, we will verify whether the *dispersal* term of the orthogroup scores really penalizes overclustering. For that, we ran OrthoFinder (Emms and Kelly 2019) one more time using the previously described Brassicaceae data set, but now with an Markov inflation parameter (*mcl*) of 5. An *mcl* of 5 is usually considered too large, so we would expect orthogroup scores to be lower than, for instance, runs with $mcl = 3$. Our goal here is to verify if our hypothesis is true.

Loading required packages:

```
set.seed(123) # for reproducibility

library(here)
library(cogeqc)
library(ggpubr)
library(rstatix)
library(patchwork)
library(tidyverse)

source(here("code", "utils.R"))
```

2 Data acquisition

We ran OrthoFinder with the following code:

```
# Run OrthoFinder - default DIAMOND mode, mcl = 5
orthofinder -f data -S diamond -I 5 -o products/result_files/default_5 -og
```

Now, we will load our data to the R session as a list of **cogeqc**-friendly orthogroup data frames.

```
# Extract tar.xz file
tarfile <- here("products", "result_files", "Orthogroups.tar.xz")
outdir <- tempdir()

system2("tar", args = c("-xf", tarfile, "--directory", outdir))

# Get path to OrthoFinder output
og_files <- list.files(
    path = outdir,
    pattern = "Orthogroups.*", full.names = TRUE
)

# Remove files for the ultrasensitive DIAMOND mode and add mcl=5
og_files <- c(
    og_files[c(2, 1, 3, 4)],
    here("products", "result_files", "Orthogroups_default_5.tsv.gz")
)
```

```r
# Read and parse files
ogs <- lapply(og_files, function(x) {
    og <- read_orthogroups(x)
    og <- og %>%
        mutate(Species = stringr::str_replace_all(Species, "\\.", "")) %>%
        mutate(Gene = str_replace_all(
            Gene, c(
                "\\.[0-9]$" = "",
                "\\.[0-9]\\.p$" = "",
                "\\.t[0-9]$" = "",
                "\\.g$" = ""
            )
        ))
    return(og)
})
names(ogs) <- c("1", "1.5", "2", "3", "5")
```

Next, we will load InterPro annotation from PLAZA 5.0 (Van Bel et al. 2022).

```r
# Define function to read functional annotation from PLAZA 5.0
read_annotation <- function(url, cols = c(1, 3)) {
    annot <- readr::read_tsv(url, show_col_types = FALSE, skip = 8) %>%
        select(cols)
    names(annot)[1:2] <- c("Gene", "Annotation")
    return(annot)
}

# Get Interpro annotation
base <- "https://ftp.psb.ugent.be/pub/plaza/plaza_public_dicots_05/InterPro/"
interpro <- list(
    Athaliana = read_annotation(paste0(base, "interpro.ath.csv.gz")),
    Aarabicum = read_annotation(paste0(base, "interpro.aar.csv.gz")),
    Alyrata_cvMN47 = read_annotation(paste0(base, "interpro.aly.csv.gz")),
    Bcarinata_cvzd1 = read_annotation(paste0(base, "interpro.bca.csv.gz")),
    Crubella_cvMonteGargano = read_annotation(paste0(base, "interpro.cru.csv.gz")),
    Chirsuta = read_annotation(paste0(base, "interpro.chi.csv.gz")),
    Sparvula = read_annotation(paste0(base, "interpro.spa.csv.gz"))
)

interpro <- lapply(interpro, as.data.frame)
```

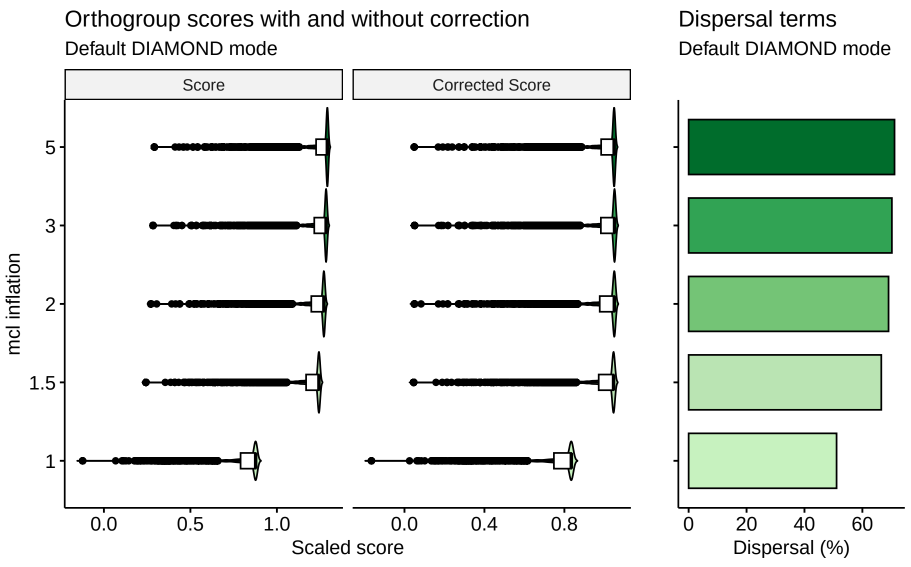
3 Validating the overclustering correction

Now that we have all data we need (orthogroup data frames and domain annotations), let's calculate orthogroup scores. Here, we will use the function `calculate_H_with_terms()` from the file *utils.R*, which contains a slightly modified version of the function `calculate_H()` from **cogeqc**, but instead of updating the uncorrected scores with the corrected scores, it returns the dispersal terms and corrected scores and separate variables.

```r
# Calculate orthogroup scores with and without correction for overclustering
og_homogeneity <- Reduce(rbind, lapply(seq_along(ogs), function(x) {

    mode <- names(ogs)[x]
    annotation <- Reduce(rbind, interpro) |> distinct()

    message("Working on ", mode)

    orthogroup_df <- merge(
        ogs[[x]],
        annotation,
        all.x = TRUE
    )

    scores_df <- calculate_H_with_terms(
        orthogroup_df, correct_overclustering = TRUE, update_score = FALSE
    )
    scores_df$Mode <- mode

    return(scores_df)
}))

og_homogeneity$Mode <- factor(
    og_homogeneity$Mode, levels = unique(og_homogeneity$Mode)
)
```

Next, let's visualize orthogroup scores with and without corrections, as well as look at the dispersal terms for each mode.

```r
# Plot scores
p_scores <- og_homogeneity |>
    mutate(
        Score = (Score - min(Score) / (max(Score) - min(Score))),
        Score_c = (Score_c - min(Score_c) / (max(Score_c) - min(Score_c)))
    ) |>
    dplyr::select(Orthogroup, Score, `Corrected Score` = Score_c, Mode) |>
    pivot_longer(
        !c("Orthogroup", "Mode"),
        names_to = "Measure",
        values_to = "Score"
    ) |>
    mutate(
        Measure = factor(Measure, levels = c("Score", "Corrected Score"))
    ) |>
    ggpubr::ggviolin(
        x = "Mode", y = "Score",
        orientation = "horiz",
        fill = "Mode",
        palette = rev(c("#006D2C", "#31A354", "#74C476", "#BAE4B3", "#c7f2bf")),
        add = "boxplot", add.params = list(fill = "white")
    ) +
    labs(
        x = "mcl inflation", y = "Scaled score",
        title = "Orthogroup scores with and without correction",
        subtitle = "Default DIAMOND mode"
    ) +
    facet_wrap(~Measure, scales = "free_x", nrow = 1) +
    theme(legend.position = "none")

# Plot dispersal terms
p_dispersal <- og_homogeneity |>
    dplyr::select(Mode, Dispersal) |>
    mutate(Dispersal = Dispersal * 100) |>
    dplyr::distinct() |>
    ggpubr::ggbarplot(
        x = "Mode", y = "Dispersal", stat = "identity",
        orientation = "horiz",
        fill = "Mode",
        palette = rev(c("#006D2C", "#31A354", "#74C476", "#BAE4B3", "#c7f2bf")),
    ) +
    labs(
        x = "", y = "Dispersal (%)",
        title = "Dispersal terms",
        subtitle = "Default DIAMOND mode"
    ) +
    theme(
        legend.position = "none",
        axis.text.y = element_blank()
    )

# Combine plots
p_combined <- patchwork::wrap_plots(p_scores, p_dispersal, widths = c(2.5, 1))

p_combined
```



We can see that, without correction (homogeneity only), increasing the value for the *mcl* parameter leads to increasingly larger scores. However, as homogeneity increases, the dispersal also increases. After correcting for dispersal, larger values for the *mcl* parameter do not lead to higher orthogroup scores.

To verify that formally, let's perform a Mann-Whitney U test for differences in orthogroup scores for runs with mcl of 3 and 5 with and without correcting for dispersal.

```
# Without dispersal
compare(og_homogeneity, "Score ~ Mode") |>
    filter(group1 == "3" & group2 == "5")
##   group1 group2    n1    n2 padj_greater padj_less padj_interpretation
## 1      3      5 23849 25595            1         0                less
##     effsize magnitude
## 1 0.3913194  moderate

# With dispersal
compare(og_homogeneity, "Score_c ~ Mode") |>
    filter(group1 == "3" & group2 == "5")
##   group1 group2    n1    n2 padj_greater padj_less padj_interpretation
## 1      3      5 23849 25595            0         1             greater
##     effsize magnitude
## 1 0.3591549  moderate
```

As expected, without correcting for dispersal, using $mcl = 5$ leads to better orthogroup scores than using $mcl = 3$. However, after correction, orthogroup scores for $mcl = 5$ are worse than scores for $mcl = 3$, which is desired.

Session info

This document was created under the following conditions:

```
## - Session info -------------------------------------------------------------
##  setting  value
##  version  R version 4.3.0 (2023-04-21)
##  os       Ubuntu 20.04.5 LTS
##  system   x86_64, linux-gnu
##  ui       X11
##  language (EN)
##  collate  en_US.UTF-8
##  ctype    en_US.UTF-8
##  tz       Europe/Brussels
##  date     2023-10-06
##  pandoc   3.1.1 @ /usr/lib/rstudio/resources/app/bin/quarto/bin/tools/ (via rmarkdown)
##
## - Packages -----------------------------------------------------------------
##  package          * version   date (UTC) lib source
##  abind              1.4-5     2016-07-21 [1] CRAN (R 4.3.0)
##  ape                5.7-1     2023-03-13 [1] CRAN (R 4.3.0)
##  aplot              0.1.10    2023-03-08 [1] CRAN (R 4.3.0)
##  backports          1.4.1     2021-12-13 [1] CRAN (R 4.3.0)
##  beeswarm           0.4.0     2021-06-01 [1] CRAN (R 4.3.0)
##  BiocGenerics       0.46.0    2023-04-25 [1] Bioconductor
##  BiocManager        1.30.21.1 2023-07-18 [1] CRAN (R 4.3.0)
##  BiocStyle        * 2.29.1    2023-08-04 [1] Github (Bioconductor/BiocStyle@7c0e093)
##  Biostrings         2.68.0    2023-04-25 [1] Bioconductor
##  bitops             1.0-7     2021-04-24 [1] CRAN (R 4.3.0)
##  bookdown           0.34      2023-05-09 [1] CRAN (R 4.3.0)
##  broom              1.0.4     2023-03-11 [1] CRAN (R 4.3.0)
##  car                3.1-2     2023-03-30 [1] CRAN (R 4.3.0)
##  carData            3.0-5     2022-01-06 [1] CRAN (R 4.3.0)
##  cli                3.6.1     2023-03-23 [1] CRAN (R 4.3.0)
##  codetools          0.2-19    2023-02-01 [4] CRAN (R 4.2.2)
##  cogeqc           * 1.4.0     2023-04-25 [1] Bioconductor
##  coin               1.4-2     2021-10-08 [1] CRAN (R 4.3.0)
##  colorspace         2.1-0     2023-01-23 [1] CRAN (R 4.3.0)
##  crayon             1.5.2     2022-09-29 [1] CRAN (R 4.3.0)
##  digest             0.6.33    2023-07-07 [1] CRAN (R 4.3.0)
##  dplyr            * 1.1.2     2023-04-20 [1] CRAN (R 4.3.0)
##  evaluate           0.21      2023-05-05 [1] CRAN (R 4.3.0)
##  fansi              1.0.4     2023-01-22 [1] CRAN (R 4.3.0)
##  farver             2.1.1     2022-07-06 [1] CRAN (R 4.3.0)
##  fastmap            1.1.1     2023-02-24 [1] CRAN (R 4.3.0)
##  forcats          * 1.0.0     2023-01-29 [1] CRAN (R 4.3.0)
##  generics           0.1.3     2022-07-05 [1] CRAN (R 4.3.0)
##  GenomeInfoDb       1.36.0    2023-04-25 [1] Bioconductor
##  GenomeInfoDbData   1.2.10    2023-04-28 [1] Bioconductor
##  ggbeeswarm         0.7.2     2023-04-29 [1] CRAN (R 4.3.0)
##  ggfun              0.0.9     2022-11-21 [1] CRAN (R 4.3.0)
##  ggplot2          * 3.4.1     2023-02-10 [1] CRAN (R 4.3.0)
##  ggplotify          0.1.0     2021-09-02 [1] CRAN (R 4.3.0)
##  ggpubr           * 0.6.0     2023-02-10 [1] CRAN (R 4.3.0)
##  ggsignif           0.6.4     2022-10-13 [1] CRAN (R 4.3.0)
##  ggtree             3.8.0     2023-04-25 [1] Bioconductor
##  glue               1.6.2     2022-02-24 [1] CRAN (R 4.3.0)
##  gridGraphics       0.5-1     2020-12-13 [1] CRAN (R 4.3.0)
##  gtable             0.3.3     2023-03-21 [1] CRAN (R 4.3.0)
##  here             * 1.0.1     2020-12-13 [1] CRAN (R 4.3.0)
##  hms                1.1.3     2023-03-21 [1] CRAN (R 4.3.0)
##  htmltools          0.5.5     2023-03-23 [1] CRAN (R 4.3.0)
##  igraph             1.4.2     2023-04-07 [1] CRAN (R 4.3.0)
##  IRanges            2.34.0    2023-04-25 [1] Bioconductor
##  jsonlite           1.8.7     2023-06-29 [1] CRAN (R 4.3.0)
##  knitr              1.43      2023-05-25 [1] CRAN (R 4.3.0)
##  labeling           0.4.2     2020-10-20 [1] CRAN (R 4.3.0)
##  lattice            0.20-45   2021-09-22 [4] CRAN (R 4.2.0)
##  lazyeval           0.2.2     2019-03-15 [1] CRAN (R 4.3.0)
##  libcoin            1.0-9     2021-09-27 [1] CRAN (R 4.3.0)
##  lifecycle          1.0.3     2022-10-07 [1] CRAN (R 4.3.0)
##  lubridate        * 1.9.2     2023-02-10 [1] CRAN (R 4.3.0)
##  magrittr           2.0.3     2022-03-30 [1] CRAN (R 4.3.0)
##  MASS               7.3-58.2  2023-01-23 [4] CRAN (R 4.2.2)
##  Matrix             1.5-1     2022-09-13 [4] CRAN (R 4.2.1)
##  matrixStats        1.0.0     2023-06-02 [1] CRAN (R 4.3.0)
##  modeltools         0.2-23    2020-03-05 [1] CRAN (R 4.3.0)
##  multcomp           1.4-25    2023-06-20 [1] CRAN (R 4.3.0)
##  munsell            0.5.0     2018-06-12 [1] CRAN (R 4.3.0)
##  mvtnorm            1.1-3     2021-10-08 [1] CRAN (R 4.3.0)
##  nlme               3.1-162   2023-01-31 [4] CRAN (R 4.2.2)
##  patchwork        * 1.1.2     2022-08-19 [1] CRAN (R 4.3.0)
##  pillar             1.9.0     2023-03-22 [1] CRAN (R 4.3.0)
##  pkgconfig          2.0.3     2019-09-22 [1] CRAN (R 4.3.0)
##  plyr               1.8.8     2022-11-11 [1] CRAN (R 4.3.0)
##  purrr            * 1.0.1     2023-01-10 [1] CRAN (R 4.3.0)
##  R6                 2.5.1     2021-08-19 [1] CRAN (R 4.3.0)
##  Rcpp               1.0.10    2023-01-22 [1] CRAN (R 4.3.0)
##  RCurl              1.98-1.12 2023-03-27 [1] CRAN (R 4.3.0)
##  readr            * 2.1.4     2023-02-10 [1] CRAN (R 4.3.0)
##  reshape2           1.4.4     2020-04-09 [1] CRAN (R 4.3.0)
##  rlang              1.1.1     2023-04-28 [1] CRAN (R 4.3.0)
##  rmarkdown          2.23      2023-07-01 [1] CRAN (R 4.3.0)
##  rprojroot          2.0.3     2022-04-02 [1] CRAN (R 4.3.0)
##  rstatix          * 0.7.2     2023-02-01 [1] CRAN (R 4.3.0)
##  rstudioapi         0.14      2022-08-22 [1] CRAN (R 4.3.0)
##  S4Vectors          0.38.0    2023-04-25 [1] Bioconductor
##  sandwich           3.0-2     2022-06-15 [1] CRAN (R 4.3.0)
##  scales             1.2.1     2022-08-20 [1] CRAN (R 4.3.0)
##  sessioninfo        1.2.2     2021-12-06 [1] CRAN (R 4.3.0)
##  stringi            1.7.12    2023-01-11 [1] CRAN (R 4.3.0)
##  stringr          * 1.5.0     2022-12-02 [1] CRAN (R 4.3.0)
##  survival           3.5-3     2023-02-12 [4] CRAN (R 4.2.2)
##  TH.data            1.1-2     2023-04-17 [1] CRAN (R 4.3.0)
##  tibble           * 3.2.1     2023-03-20 [1] CRAN (R 4.3.0)
##  tidyr            * 1.3.0     2023-01-24 [1] CRAN (R 4.3.0)
##  tidyselect         1.2.0     2022-10-10 [1] CRAN (R 4.3.0)
##  tidytree           0.4.2     2022-12-18 [1] CRAN (R 4.3.0)
##  tidyverse        * 2.0.0     2023-02-22 [1] CRAN (R 4.3.0)
##  timechange         0.2.0     2023-01-11 [1] CRAN (R 4.3.0)
##  treeio             1.24.1    2023-05-31 [1] Bioconductor
##  tzdb               0.3.0     2022-03-28 [1] CRAN (R 4.3.0)
##  utf8               1.2.3     2023-01-31 [1] CRAN (R 4.3.0)
##  vctrs              0.6.3     2023-06-14 [1] CRAN (R 4.3.0)
##  vipor              0.4.5     2017-03-22 [1] CRAN (R 4.3.0)
##  withr              2.5.0     2022-03-03 [1] CRAN (R 4.3.0)
##  xfun               0.39      2023-04-20 [1] CRAN (R 4.3.0)
##  XVector            0.40.0    2023-04-25 [1] Bioconductor
##  yaml               2.3.7     2023-01-23 [1] CRAN (R 4.3.0)
##  yulab.utils        0.0.6     2022-12-20 [1] CRAN (R 4.3.0)
##  zlibbioc           1.46.0    2023-04-25 [1] Bioconductor
##  zoo                1.8-12    2023-04-13 [1] CRAN (R 4.3.0)
##
##  [1] /home/faalm/R/x86_64-pc-linux-gnu-library/4.3
##  [2] /usr/local/lib/R/site-library
##  [3] /usr/lib/R/site-library
##  [4] /usr/lib/R/library
##
## ------------------------------------------------------------------------------
```

References

Emms, David M, and Steven Kelly. 2019. "OrthoFinder: Phylogenetic Orthology Inference for Comparative Genomics." *Genome Biology* 20 (1): 1–14.

Van Bel, Michiel, Francesca Silvestri, Eric M Weitz, Lukasz Kreft, Alexander Botzki, Frederik Coppens, and Klaas Vandepoele. 2022. "PLAZA 5.0: Extending the Scope and Power of Comparative and Functional Genomics in Plants." *Nucleic Acids Research* 50 (D1): D1468–74.

Supplementary Text S6: Assessing synteny detection in Fabaceae

Fabricio Almeida-Silva[1,2] ***and Yves Van de Peer***[1,2,3,4]

[1]VIB-UGent Center for Plant Systems Biology, Ghent, Belgium
[2]Department of Plant Biotechnology and Bioinformatics, Ghent University, Ghent, Belgium
[3]College of Horticulture, Academy for Advanced Interdisciplinary Studies, Nanjing Agricultural University, Nanjing, China
[4]Center for Microbial Ecology and Genomics, Department of Biochemistry, Genetics and Microbiology, University of Pretoria, Pretoria, South Africa

6 October 2023

Contents

1 Overview 2
2 Data acquisition 2
3 Network-based synteny assessment 3
3.1 Assessing the Fabaceae synteny network 3
3.2 Assessing species-specific synteny networks 5
Session info 7
References 10

```r
library(cogeqc)
library(here)
library(tidyverse)
library(syntenet)
library(tidytext)
```

1 Overview

Here, we will assess synteny detection using a network-based approach. The anchor pairs from synteny identification will be interpreted as edges of an unweighted undirected graph (i.e., a synteny network), and the best synteny detection will be identified based on the graphs' clustering coefficients and node number.

We will demonstrate our network-based synteny assessment using genomic data on Fabaceae species available on PLAZA 5.0 (Van Bel et al. 2022).

2 Data acquisition

In this section, we will download whole-genome protein sequences and gene annotation from PLAZA 5.0, and then we will preprocess the data with `syntenet::process_input()`.

```r
species <- c("mtr", "tpr", "psa", "car", "lja", "gma", "vmu", "lal", "arhy")
```

```r
base_url <- "https://ftp.psb.ugent.be/pub/plaza/plaza_public_dicots_05/"

# Get proteomes
seq_url <- paste0(
    base_url, "Fasta/proteome.selected_transcript.",
    species, ".fasta.gz"
)

## Import files and clean gene IDs
seq <- lapply(seq_url, function(x) {
    s <- Biostrings::readAAStringSet(x)
    names(s) <- gsub(".* | ", "", names(s))
    return(s)
})
names(seq) <- species


# Get gene annotation
annot_url <- paste0(
    base_url, "GFF/", species, "/annotation.selected_transcript.exon_features.",
    species, ".gff3.gz"
)

## Import files and keep only relevant fields
annot <- lapply(annot_url, function(x) {
    a <- rtracklayer::import(x)
    a <- a[, c("type", "gene_id")]
    a <- a[a$type == "gene"]
    return(a)
})
names(annot) <- species

# Process data
pdata <- process_input(seq, annot)

# Remove unprocessed data to clean the working environment
rm(annot)
rm(seq)
```

3 Network-based synteny assessment

We will infer synteny networks using the Bioconductor package *syntenet*. This package detects synteny using the MCScanX algorithm (Wang et al. 2012), which can produce different results based on 2 main parameters:

1. **anchors:** minimum required number of genes to call a syntenic block. Default: 5.
2. **max_gaps:** number of upstream and downstream genes to search for anchors. Default: 25.

We will infer synteny networks with 5 combinations of parameters, similarly to Zhao and Schranz (2019), using two approaches:

1. A single Fabaceae synteny network;
2. Species-specific synteny networks for each Fabaceae species.

To start with, let's define the combinations of parameters we will use.

```
# Define combinations of parameters: anchors (a), max_gaps (m)
synteny_params <- list(
    c(3, 25),
    c(5, 15),
    c(5, 25),
    c(5, 35),
    c(7, 25)
)
```

3.1 Assessing the Fabaceae synteny network

First, we will perform similarity searches with DIAMOND.

```
# Define wrapper function to run DIAMOND with different top_hits
out <- file.path(tempdir(), "diamond_all")
d5 <- run_diamond(seq = seq, top_hits = 5, outdir = out)
```

With the DIAMOND list, we can detect synteny.

```r
# Define helper function to detect synteny with multiple combinations of params
synteny_wrapper <- function(diamond, annotation, params) {

    syn <- lapply(params, function(x) {

        anchors <- x[1]
        max_gaps <- x[2]
        outdir <- file.path(tempdir(), paste0("syn_a", anchors, "_m", max_gaps))

        s <- infer_syntenet(
            blast_list = diamond,
            annotation = pdata$annotation,
            outdir = outdir,
            anchors = anchors,
            max_gaps = max_gaps
        )
        return(s)
    })
    return(syn)
}

# Detect synteny
syn_fabaceae <- synteny_wrapper(d5, pdata$annotation, synteny_params)
names(syn_fabaceae) <- unlist(
    lapply(synteny_params, function(x) paste0("a", x[1], "_m", x[2]))
)
```

Now, let's use the network-based synteny assessment to see which combination of parameters is the best.

```r
# Assess networks
fabaceae_scores <- assess_synnet_list(syn_fabaceae)

# Look at scores, ranked from highest to lowest
fabaceae_scores %>%
    arrange(-Score) |>
    knitr::kable(
        caption = "Scores for each synteny network."
    )
```

Table 1: Scores for each synteny network.

CC	Node_count	Rsquared	Score	Network
0.8253002	237723	0.6227916	122187.2	a3_m25
0.8290880	235290	0.6156847	120105.4	a5_m35
0.8392223	226657	0.6026291	114629.5	a5_m15
0.8412602	224325	0.5972865	112717.3	a7_m25
0.8347725	231820	0.5795957	112161.6	a5_m25

As we can see, the combination of parameters `a = 3; m = 25` is the best for this data set.

Finally, let's visualize scores. To make visualization better, we will scale scores by the maximum value, so that values range from 0 to 1.

```r
# Plot scores
synteny_scores_fabaceae <- fabaceae_scores %>%
    arrange(Score) %>%
    mutate(Score = Score / max(Score)) %>%
    mutate(Parameters = str_replace_all(Network, "_", ", ")) %>%
    mutate(Parameters = factor(Parameters, levels = unique(Parameters))) %>%
    ggplot(., aes(x = Parameters, y = Score)) +
    geom_col(fill = "grey60", color = "black") +
    theme_bw() +
    labs(
        title = "Assessment of the Fabaceae synteny network",
        subtitle = "a = minimum # of anchors; m = maximum # of gaps",
        y = "Scaled score"
    )

synteny_scores_fabaceae
```

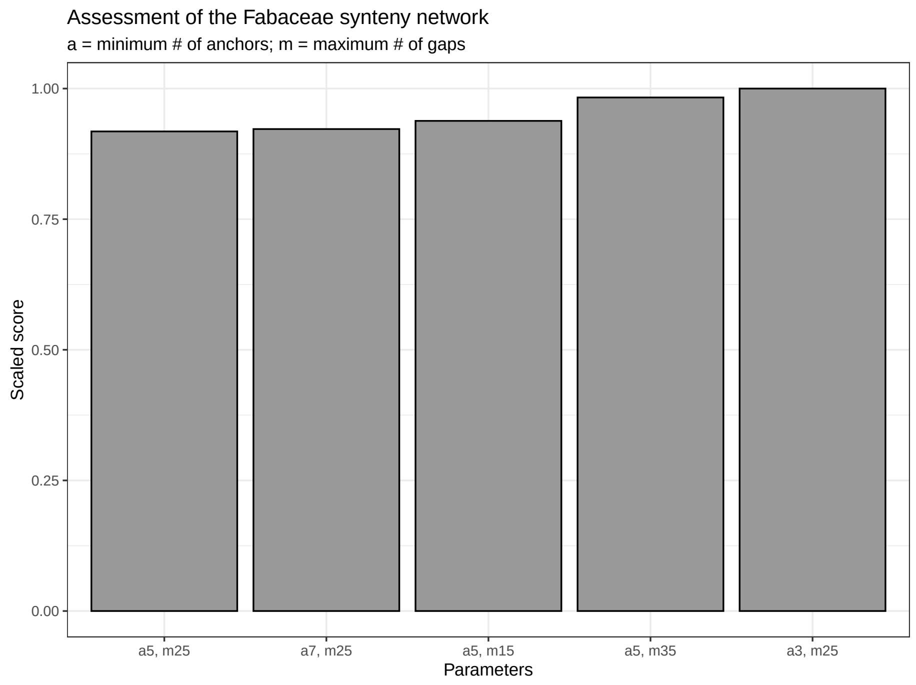


Figure 1: Scaled scores for each synteny network.

3.2 Assessing species-specific synteny networks

In this section, we will infer species-specific synteny networks and assess each of them with our network-based approach.

This time, as we already have synteny networks for the whole Fabaceae family, we don't need to infer them again; we will simply subset edges of the network that contain nodes from the same species.

```r
# Create species-specific networks
species_ids <- substr(species, start = 1, stop = 3)

species_networks <- lapply(species_ids, function(x) {

    nets <- lapply(syn_fabaceae, function(y) {
        edges <- y[startsWith(y$Anchor1, x) & startsWith(y$Anchor2, x), ]
        return(edges)
    })
    return(nets)
})
names(species_networks) <- species_ids

# Exploring data
names(species_networks)
## [1] "mtr" "tpr" "psa" "car" "lja" "gma" "vmu" "lal" "arh"
names(species_networks$mtr)
## [1] "a3_m25" "a5_m15" "a5_m25" "a5_m35" "a7_m25"

# Rename `species_networks` to keep full name
names(species_networks) <- c(
    "M. truncatula", "T. pratense", "P. sativum", "C. arietinum",
    "L. japonicus", "G. max", "V. mungo", "L. albus", "A. hypogaea"
)
```

For each species, we will assess the networks inferred with different combinations of parameters.

```r
# Assess species-specific networks
scores_species_nets <- lapply(seq_along(species_networks), function(x) {

    species <- names(species_networks)[x]
    scores <- assess_synnet_list(species_networks[[species]])
    scores$Score[is.nan(scores$Score)] <- 0
    scores <- scores[order(scores$Score, decreasing = TRUE), ]
    scores$Species <- species
    scores$Score <- scores$Score / max(scores$Score)
    return(scores)
})
scores_species_nets <- Reduce(rbind, scores_species_nets)

# Plot data
synteny_scores_species <- scores_species_nets %>%
    mutate(
        Parameters = as.factor(str_replace_all(Network, "_", ", ")),
        Species = as.factor(Species)
    ) %>%
    mutate(Network = reorder_within(Parameters, Score, Species)) %>%
    ggplot(., aes(x = Network, y = Score, fill = Parameters)) +
    geom_bar(stat = "identity", color = "grey90") +
    facet_wrap(~Species, ncol = 3, scales = "free") +
    scale_x_reordered() +
    ggsci::scale_fill_jama() +
    theme_bw() +
    theme(axis.text.x = element_blank()) +
    labs(
        title = "Assessment of species-specific synteny networks",
        subtitle = "a = minimum # of anchors; m = maximum # of gaps",
        y = "Scaled score (by species)", x = ""
    )

synteny_scores_species
```

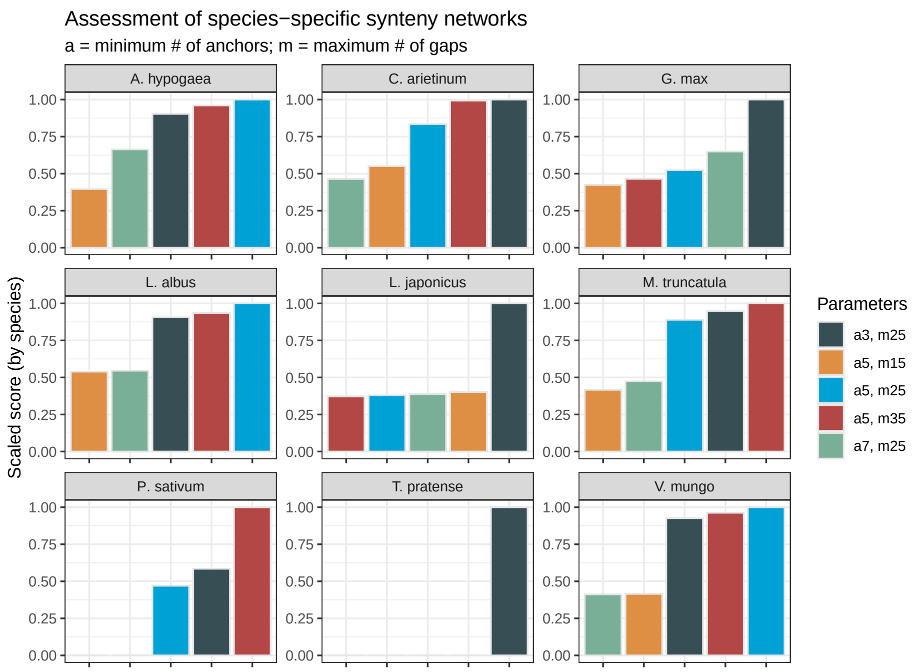


Figure 2: Scores for species-specific synteny networks.

The figure demonstrates that the best combination of parameters depends on the species, so there is no "universally" best combination. However, some patterns emerge. The combinations `a = 7; m= 25` and `a = 5; m = 15` are typically the worst. In some cases, they even lead to zero scores due to clustering coefficients of zero. Thus, if users want to test multiple combinations of parameters for their own data set, they should only test the combinations `a = 3; m = 25`, `a = 5; m = 25`, and `a = 5; m = 35`, which lead to the best score in 45%, 33%, and 22% of the species-specific networks, respectively. Interestingly, the combination that leads to the best score in most networks (`a = 3; m = 25`) is also the best when considering the whole Fabaceae synteny network (see previous section).

Session info

This document was created under the following conditions:

```
## - Session info ---------------------------------------------------------------
##  setting  value
##  version  R version 4.3.0 (2023-04-21)
##  os       Ubuntu 20.04.5 LTS
##  system   x86_64, linux-gnu
##  ui       X11
##  language (EN)
##  collate  en_US.UTF-8
##  ctype    en_US.UTF-8
##  tz       Europe/Brussels
##  date     2023-10-06
##  pandoc   3.1.1 @ /usr/lib/rstudio/resources/app/bin/quarto/bin/tools/ (via rmarkdown)
##
## - Packages -------------------------------------------------------------------
##  package             * version   date (UTC) lib source
##  ape                   5.7-1     2023-03-13 [1] CRAN (R 4.3.0)
##  aplot                 0.1.10    2023-03-08 [1] CRAN (R 4.3.0)
##  beeswarm              0.4.0     2021-06-01 [1] CRAN (R 4.3.0)
##  Biobase               2.60.0    2023-04-25 [1] Bioconductor
##  BiocGenerics          0.46.0    2023-04-25 [1] Bioconductor
##  BiocIO                1.10.0    2023-04-25 [1] Bioconductor
##  BiocManager           1.30.21.1 2023-07-18 [1] CRAN (R 4.3.0)
##  BiocParallel          1.34.0    2023-04-25 [1] Bioconductor
##  BiocStyle           * 2.29.1    2023-08-04 [1] Github (Bioconductor/BiocStyle@7c0e093)
##  Biostrings            2.68.0    2023-04-25 [1] Bioconductor
##  bitops                1.0-7     2021-04-24 [1] CRAN (R 4.3.0)
##  bookdown              0.34      2023-05-09 [1] CRAN (R 4.3.0)
##  cli                   3.6.1     2023-03-23 [1] CRAN (R 4.3.0)
##  coda                  0.19-4    2020-09-30 [1] CRAN (R 4.3.0)
##  codetools             0.2-19    2023-02-01 [4] CRAN (R 4.2.2)
##  cogeqc              * 1.4.0     2023-04-25 [1] Bioconductor
##  colorspace            2.1-0     2023-01-23 [1] CRAN (R 4.3.0)
##  crayon                1.5.2     2022-09-29 [1] CRAN (R 4.3.0)
##  DelayedArray          0.26.1    2023-05-01 [1] Bioconductor
##  digest                0.6.33    2023-07-07 [1] CRAN (R 4.3.0)
##  dplyr               * 1.1.2     2023-04-20 [1] CRAN (R 4.3.0)
##  evaluate              0.21      2023-05-05 [1] CRAN (R 4.3.0)
##  fansi                 1.0.4     2023-01-22 [1] CRAN (R 4.3.0)
##  farver                2.1.1     2022-07-06 [1] CRAN (R 4.3.0)
##  fastmap               1.1.1     2023-02-24 [1] CRAN (R 4.3.0)
##  forcats             * 1.0.0     2023-01-29 [1] CRAN (R 4.3.0)
##  generics              0.1.3     2022-07-05 [1] CRAN (R 4.3.0)
##  GenomeInfoDb          1.36.0    2023-04-25 [1] Bioconductor
##  GenomeInfoDbData      1.2.10    2023-04-28 [1] Bioconductor
##  GenomicAlignments     1.36.0    2023-04-25 [1] Bioconductor
##  GenomicRanges         1.52.0    2023-04-25 [1] Bioconductor
##  ggbeeswarm            0.7.2     2023-04-29 [1] CRAN (R 4.3.0)
##  ggfun                 0.0.9     2022-11-21 [1] CRAN (R 4.3.0)
##  ggnetwork             0.5.12    2023-03-06 [1] CRAN (R 4.3.0)
##  ggplot2             * 3.4.1     2023-02-10 [1] CRAN (R 4.3.0)
##  ggplotify             0.1.0     2021-09-02 [1] CRAN (R 4.3.0)
## ggsci              3.0.0    2023-03-08 [1] CRAN (R 4.3.0)
## ggtree             3.8.0    2023-04-25 [1] Bioconductor
## glue               1.6.2    2022-02-24 [1] CRAN (R 4.3.0)
## gridGraphics       0.5-1    2020-12-13 [1] CRAN (R 4.3.0)
## gtable             0.3.3    2023-03-21 [1] CRAN (R 4.3.0)
## here             * 1.0.1    2020-12-13 [1] CRAN (R 4.3.0)
## hms                1.1.3    2023-03-21 [1] CRAN (R 4.3.0)
## htmltools          0.5.5    2023-03-23 [1] CRAN (R 4.3.0)
## htmlwidgets        1.6.2    2023-03-17 [1] CRAN (R 4.3.0)
## igraph             1.4.2    2023-04-07 [1] CRAN (R 4.3.0)
## intergraph         2.0-2    2016-12-05 [1] CRAN (R 4.3.0)
## IRanges            2.34.0   2023-04-25 [1] Bioconductor
## janeaustenr        1.0.0    2022-08-26 [1] CRAN (R 4.3.0)
## jsonlite           1.8.7    2023-06-29 [1] CRAN (R 4.3.0)
## knitr              1.43     2023-05-25 [1] CRAN (R 4.3.0)
## labeling           0.4.2    2020-10-20 [1] CRAN (R 4.3.0)
## lattice            0.20-45  2021-09-22 [4] CRAN (R 4.2.0)
## lazyeval           0.2.2    2019-03-15 [1] CRAN (R 4.3.0)
## lifecycle          1.0.3    2022-10-07 [1] CRAN (R 4.3.0)
## lubridate        * 1.9.2    2023-02-10 [1] CRAN (R 4.3.0)
## magrittr           2.0.3    2022-03-30 [1] CRAN (R 4.3.0)
## Matrix             1.5-1    2022-09-13 [4] CRAN (R 4.2.1)
## MatrixGenerics     1.12.2   2023-06-09 [1] Bioconductor
## matrixStats        1.0.0    2023-06-02 [1] CRAN (R 4.3.0)
## munsell            0.5.0    2018-06-12 [1] CRAN (R 4.3.0)
## network            1.18.1   2023-01-24 [1] CRAN (R 4.3.0)
## networkD3          0.4      2017-03-18 [1] CRAN (R 4.3.0)
## nlme               3.1-162  2023-01-31 [4] CRAN (R 4.2.2)
## patchwork          1.1.2    2022-08-19 [1] CRAN (R 4.3.0)
## pheatmap           1.0.12   2019-01-04 [1] CRAN (R 4.3.0)
## pillar             1.9.0    2023-03-22 [1] CRAN (R 4.3.0)
## pkgconfig          2.0.3    2019-09-22 [1] CRAN (R 4.3.0)
## plyr               1.8.8    2022-11-11 [1] CRAN (R 4.3.0)
## purrr            * 1.0.1    2023-01-10 [1] CRAN (R 4.3.0)
## R6                 2.5.1    2021-08-19 [1] CRAN (R 4.3.0)
## RColorBrewer       1.1-3    2022-04-03 [1] CRAN (R 4.3.0)
## Rcpp               1.0.10   2023-01-22 [1] CRAN (R 4.3.0)
## RCurl              1.98-1.12 2023-03-27 [1] CRAN (R 4.3.0)
## readr            * 2.1.4    2023-02-10 [1] CRAN (R 4.3.0)
## reshape2           1.4.4    2020-04-09 [1] CRAN (R 4.3.0)
## restfulr           0.0.15   2022-06-16 [1] CRAN (R 4.3.0)
## rjson              0.2.21   2022-01-09 [1] CRAN (R 4.3.0)
## rlang              1.1.1    2023-04-28 [1] CRAN (R 4.3.0)
## rmarkdown          2.23     2023-07-01 [1] CRAN (R 4.3.0)
## rprojroot          2.0.3    2022-04-02 [1] CRAN (R 4.3.0)
## Rsamtools          2.16.0   2023-04-25 [1] Bioconductor
## rstudioapi         0.14     2022-08-22 [1] CRAN (R 4.3.0)
## rtracklayer        1.60.0   2023-04-25 [1] Bioconductor
## S4Arrays           1.0.1    2023-05-01 [1] Bioconductor
## S4Vectors          0.38.0   2023-04-25 [1] Bioconductor
## scales             1.2.1    2022-08-20 [1] CRAN (R 4.3.0)
## sessioninfo           1.2.2     2021-12-06 [1] CRAN (R 4.3.0)
## SnowballC             0.7.1     2023-04-25 [1] CRAN (R 4.3.0)
## statnet.common        4.8.0     2023-01-24 [1] CRAN (R 4.3.0)
## stringi               1.7.12    2023-01-11 [1] CRAN (R 4.3.0)
## stringr             * 1.5.0     2022-12-02 [1] CRAN (R 4.3.0)
## SummarizedExperiment  1.30.1    2023-05-01 [1] Bioconductor
## syntenet            * 1.3.3     2023-06-15 [1] Bioconductor
## tibble              * 3.2.1     2023-03-20 [1] CRAN (R 4.3.0)
## tidyr               * 1.3.0     2023-01-24 [1] CRAN (R 4.3.0)
## tidyselect            1.2.0     2022-10-10 [1] CRAN (R 4.3.0)
## tidytext            * 0.4.1     2023-01-07 [1] CRAN (R 4.3.0)
## tidytree              0.4.2     2022-12-18 [1] CRAN (R 4.3.0)
## tidyverse           * 2.0.0     2023-02-22 [1] CRAN (R 4.3.0)
## timechange            0.2.0     2023-01-11 [1] CRAN (R 4.3.0)
## tokenizers            0.3.0     2022-12-22 [1] CRAN (R 4.3.0)
## treeio                1.24.1    2023-05-31 [1] Bioconductor
## tzdb                  0.3.0     2022-03-28 [1] CRAN (R 4.3.0)
## utf8                  1.2.3     2023-01-31 [1] CRAN (R 4.3.0)
## vctrs                 0.6.3     2023-06-14 [1] CRAN (R 4.3.0)
## vipor                 0.4.5     2017-03-22 [1] CRAN (R 4.3.0)
## withr                 2.5.0     2022-03-03 [1] CRAN (R 4.3.0)
## xfun                  0.39      2023-04-20 [1] CRAN (R 4.3.0)
## XML                   3.99-0.14 2023-03-19 [1] CRAN (R 4.3.0)
## XVector               0.40.0    2023-04-25 [1] Bioconductor
## yaml                  2.3.7     2023-01-23 [1] CRAN (R 4.3.0)
## yulab.utils           0.0.6     2022-12-20 [1] CRAN (R 4.3.0)
## zlibbioc              1.46.0    2023-04-25 [1] Bioconductor
##
## [1] /home/faalm/R/x86_64-pc-linux-gnu-library/4.3
## [2] /usr/local/lib/R/site-library
## [3] /usr/lib/R/site-library
## [4] /usr/lib/R/library
##
## ------------------------------------------------------------------------------
```

References

Van Bel, Michiel, Francesca Silvestri, Eric M Weitz, Lukasz Kreft, Alexander Botzki, Frederik Coppens, and Klaas Vandepoele. 2022. "PLAZA 5.0: Extending the Scope and Power of Comparative and Functional Genomics in Plants." *Nucleic Acids Research* 50 (D1): D1468–74.

Wang, Yupeng, Haibao Tang, Jeremy D DeBarry, Xu Tan, Jingping Li, Xiyin Wang, Tae-ho Lee, et al. 2012. "MCScanX: A Toolkit for Detection and Evolutionary Analysis of Gene Synteny and Collinearity." *Nucleic Acids Research* 40 (7): e49–49.

Zhao, Tao, and M Eric Schranz. 2019. "Network-Based Microsynteny Analysis Identifies Major Differences and Genomic Outliers in Mammalian and Angiosperm Genomes." *Proceedings of the National Academy of Sciences* 116 (6): 2165–74.