Supplementary Text S5: Runtime benchmark

Fabricio Almeida-Silva [1] ***and Yves Van de Peer*** [1]

[1]VIB-UGent Center for Plant Systems Biology, Ghent University, Ghent, Belgium

26 July 2024

Contents

1 Introduction

Here, we will perform a runtime benchmark for functions related to duplicate classification and substitution rates calculation using model organisms.

To start, let's load the required data and packages.

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

# Load required packages
library(doubletrouble)
library(here)
## here() starts at /home/faalm/Dropbox/package_benchmarks/doubletrouble_paper
library(tidyverse)
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
library(patchwork)

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

# Load sample metadata for Ensembl instances
load(here("products", "result_files", "metadata_all.rda"))
```

2 Benchmark 1: `classify_gene_pairs()`

Here, we will benchmark the performance of `classify_gene_pairs()` with model organisms.

First, let's get the genome and annotation data.

```
# Create a data frame with names of model species and their Ensembl instances
model_species <- data.frame(
    species = c(
        "arabidopsis_thaliana", "caenorhabditis_elegans",
        "homo_sapiens", "saccharomyces_cerevisiae",
        "drosophila_melanogaster", "danio_rerio"
    ),
    instance = c(
        "plants", "metazoa", "ensembl", "fungi", "metazoa", "ensembl"
    )
)

# For each organism, download data, and identify and classify duplicates
model_duplicates <- lapply(seq_len(nrow(model_species)), function(x) {
```

Supplementary Text S5: Runtime benchmark

```
        species <- model_species$species[x]
        instance <- model_species$instance[x]

        # Get annotation
        annot <- get_annotation(model_species[x, ], instance)

        # Get proteome and keep only primary transcripts
        seq <- get_proteomes(model_species[x, ], instance)
        seq <- filter_sequences(seq, annot)

        # Process data
        pdata <- syntenet::process_input(seq, annot, filter_annotation = TRUE)

        # Perform DIAMOND search
        outdir <- file.path(tempdir(), paste0(species, "_intra"))
        diamond <- syntenet::run_diamond(
            seq = pdata$seq,
            compare = "intraspecies",
            outdir = outdir,
            threads = 4,
            ... = "--sensitive"
        )

        fs::dir_delete(outdir)

        # Classify duplicates - standard mode
        start <- Sys.time()
        duplicate_pairs <- classify_gene_pairs(
                blast_list = diamond,
                annotation = pdata$annotation,
                scheme = "standard"
            )[[1]]
        end <- Sys.time()
        runtime <- end - start

        return(runtime)
    })
    names(model_duplicates) <- gsub("_", " ", str_to_title(model_species$species))

    # Summarize results in a table
    benchmark_classification <- data.frame(
        species = names(model_duplicates),
        time_seconds = as.numeric(unlist(model_duplicates))
    )

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

3 Benchmark 2: `pairs2kaks()`

Next, we will benchmark the performance of `pairs2kaks()` for duplicate pairs in the *Saccharomyces cerevisiae* genome. We will do it using a single thread, and using parallelization (with 4 and 8 threads).

First of all, let's get the required data for `pairs2kaks()`.

```
# Load duplicate pairs for S. cerevisiae
load(here("products", "result_files", "fungi_duplicates.rda"))
scerevisiae_pairs <- fungi_duplicates["saccharomyces_cerevisiae"]

# Get CDS for S. cerevisiae
scerevisiae_cds <- get_cds_ensembl("saccharomyces_cerevisiae", "fungi")
```

Now, we can do the benchmark.

```
# 1 thread
start <- Sys.time()
kaks <- pairs2kaks(
    scerevisiae_pairs,
    scerevisiae_cds,
    threads = 1
)
end <- Sys.time()
runtime1 <- end - start

# 4 threads
start <- Sys.time()
kaks <- pairs2kaks(
    scerevisiae_pairs,
    scerevisiae_cds,
    threads = 4
)
end <- Sys.time()
runtime4 <- end - start

# 8 threads
start <- Sys.time()
kaks <- pairs2kaks(
    scerevisiae_pairs,
    scerevisiae_cds,
    threads = 8
)
end <- Sys.time()
runtime8 <- end - start

# Summarize results in a table
benchmark_kaks <- data.frame(
    Threads = factor(c(1, 4, 8)),
    Time_minutes = as.numeric(c(runtime1, runtime4, runtime8))
) |>
```

```
        dplyr::mutate(
            Pairs_per_minute = nrow(scerevisiae_pairs[[1]]) / Time_minutes,
            Pairs_per_second = nrow(scerevisiae_pairs[[1]]) / (Time_minutes * 60)
        )
    save(
        benchmark_kaks, compress = "xz",
        file = here("products", "result_files", "benchmark_kaks.rda")
    )
```

4 Benchmark 3: **doubletrouble** vs **DupGen_finder**

Here, we will classify duplicate pairs in the *A. thaliana* genome using **doubletrouble** and DupGen_finder to assess if they produce the same results, and compare their runtimes.

First, let's get all data we need (proteomes, annotation, and DIAMOND tables).

```
    # Get annotation
    smeta <- data.frame(
        species = c("arabidopsis_thaliana", "amborella_trichopoda"),
        instance = "plants"
    )
    annot <- get_annotation(smeta, "plants")

    # Get proteome and keep only primary transcripts
    seq <- get_proteomes(smeta, "plants")
    seq <- filter_sequences(seq, annot)

    # Process data
    pdata <- syntenet::process_input(seq, annot, filter_annotation = TRUE)

    # Run intraspecies DIAMOND search
    outdir <- file.path(tempdir(), "benchmark3_intra")
    diamond_intra <- syntenet::run_diamond(
        seq = pdata$seq,
        compare = "intraspecies",
        outdir = outdir,
        threads = 4,
        ... = "--sensitive"
    )
    fs::dir_delete(outdir)

    # Run interspecies DIAMOND search
    outdir2 <- file.path(tempdir(), "benchmark3_inter")
    compare_df <- data.frame(
        query = "arabidopsis.thaliana", target = "amborella.trichopoda"
    )

    diamond_inter <- syntenet::run_diamond(
        seq = pdata$seq,
        compare = compare_df,
```

```
    outdir = outdir2,
    threads = 4,
    ... = "--sensitive"
)
fs::dir_delete(outdir2)
```

Now, let's classify duplicates with **doubletrouble**.

```
# Classify duplicates with doubletrouble
start1 <- Sys.time()
dups1 <- classify_gene_pairs(
    blast_list = diamond_intra[2],
    annotation = pdata$annotation,
    blast_inter = diamond_inter,
    scheme = "extended",
    collinearity_dir = here("products")
)
end1 <- Sys.time()
runtime1 <- end1 - start1
```

Next, we will classify duplicates with DupGen_finder. For that, we will first export input data in the required format.

```
# Export data
## .blast files
b1 <- diamond_intra$arabidopsis.thaliana_arabidopsis.thaliana |>
    mutate(
        query = str_replace_all(query, "^ara_", ""),
        db = str_replace_all(db, "^ara_", "")
    )

b2 <- diamond_inter$arabidopsis.thaliana_amborella.trichopoda |>
    mutate(
        query = str_replace_all(query, "^ara_", ""),
        db = str_replace_all(db, "^amb_", "")
    )

write_tsv(b1, file = file.path(tempdir(), "Ath.blast"), col_names = FALSE)
write_tsv(b2, file = file.path(tempdir(), "Ath_Atr.blast"), col_names = FALSE)

## .gff files
gff1 <- pdata$annotation$arabidopsis.thaliana |>
    as.data.frame() |>
    mutate(gene = str_replace_all(gene, "^ara_", "")) |>
    mutate(seqnames = str_replace_all(seqnames, "ara_", "ara-")) |>
    dplyr::select(seqnames, gene, start, end)

gff2 <- pdata$annotation$amborella.trichopoda |>
    as.data.frame() |>
    mutate(gene = str_replace_all(gene, "^amb_", "")) |>
    mutate(seqnames = str_replace_all(seqnames, "^amb_", "amb-")) |>
    dplyr::select(seqnames, gene, start, end)
```

```
gff2 <- bind_rows(gff1, gff2)

write_tsv(gff1, file = file.path(tempdir(), "Ath.gff"), col_names = FALSE)
write_tsv(gff2, file = file.path(tempdir(), "Ath_Atr.gff"), col_names = FALSE)

# Classify duplicates with DupGen_finder
args = c(
    "-i", tempdir(), "-t Ath -c Atr",
    "-o", file.path(tempdir(), "results"),
    "-e 1e-10"
)

start2 <- Sys.time()
system2("DupGen_finder.pl", args = args)
end2 <- Sys.time()
runtime2 <- end2 - start2
```

Now, let's compare both algorithms in terms of runtime and results.

```
# Load `DupGen_finder.pl` results
files <- c(
    "Ath.wgd.pairs", "Ath.tandem.pairs", "Ath.proximal.pairs",
    "Ath.transposed.pairs", "Ath.dispersed.pairs"
)
files <- file.path(tempdir(), "results", files)
names(files) <- c("SD", "TD", "PD", "TRD", "DD")
dups2 <- Reduce(rbind, lapply(seq_along(files), function(x) {
    d <- read_tsv(files[x], show_col_types = FALSE) |>
        mutate(type = names(files)[x]) |>
        select(1, 3, type) |>
        as.data.frame()

    names(d)[c(1,2)] <- c("dup1", "dup2")

    return(d)
}))

# Compare runtime
comp_runtime <- data.frame(doubletrouble = runtime1, DupGen_finder = runtime2)

# Compare number of gene pairs per category
comp_results <- inner_join(
    count(dups1$arabidopsis.thaliana, type) |>
        dplyr::rename(n_doubletrouble = n),
    count(dups2, type) |>
        dplyr::rename(n_DupGen_finder = n),
    by = "type"
)
comp_results <- bind_rows(
    comp_results,
    data.frame(
```

```
        type = "Total",
        n_doubletrouble = sum(comp_results$n_doubletrouble),
        n_DupGen_finder = sum(comp_results$n_DupGen_finder)
    )
)
```

```
list(runtime = comp_runtime, results = comp_results)
## $runtime
##   doubletrouble DupGen_finder
## 1  3.71474 secs 3.373682 secs
##
## $results
##    type n_doubletrouble n_DupGen_finder
## 1    SD            4329            4355
## 2    TD            2075            2131
## 3    PD            2789             896
## 4   TRD            6703            5941
## 5    DD           31589           17834
## 6 Total           47485           31157
```

Overall, results are similar, but there are important differences. In terms of runtime, there is no significant difference (this is the runtime of a single run, so there's some stochasticity). In terms of results, the numbers of SD, TD, and TRD pairs are similar, but there are more pronounced differences for PD and DD pairs. In particular, the total number of paralogous pairs differs between algorithms. When we remove rows from the DIAMOND output based on the E-value threshold and remove self (e.g., "gene1-gene1") and redundant hits (e.g., "gene1-gene2" and "gene2-gene1"), we get the following total number of pairs:

```
# Get total number of paralog pairs from DIAMOND output
evalue <- 1e-10
dmd <- diamond_intra$arabidopsis.thaliana_arabidopsis.thaliana

all_paralogs <- lapply(list(dmd), function(x) {
    fpair <- x[x$evalue <= evalue, c(1, 2)]
    fpair <- fpair[fpair[, 1] != fpair[, 2], ]
    fpair <- fpair[!duplicated(t(apply(fpair, 1, sort))), ]
    names(fpair) <- c("dup1", "dup2")
    return(fpair)
})[[1]]
```

```
nrow(all_paralogs)
## [1] 47485
```

The total number of paralogous pairs in the DIAMOND output is the same as sum of classes for **doubletrouble**, but greater than the sum of classes for **DupGen_finder**, indicating that the latter probably does some additional (undocumented) filtering before classifying gene pairs, or it could be a bug.

Finally, since the number of PD pairs identified by **doubletrouble** is much greater than the number of PD pairs identified by **DupGen_finder**, we will explore a few of these PD pairs so check whether **doubletrouble** misclassified them.

Supplementary Text S5: Runtime benchmark

```
pd1 <- dups1$arabidopsis.thaliana |>
    filter(type == "PD")

pd2 <- dups2 |>
    filter(type == "PD")

# Get all pairs in `pd1` and `pd2`
p1 <- t(apply(pd1[, 1:2], 1, sort)) |>
    as.data.frame() |>
    mutate(V1 = str_replace_all(V1, "^ara_", "")) |>
    mutate(V2 = str_replace_all(V2, "^ara_", "")) |>
    mutate(pair_string = str_c(V1, V2, sep = "-")) |>
    pull(pair_string)

p2 <- t(apply(pd2[, 1:2], 1, sort)) |>
    as.data.frame() |>
    mutate(pair_string = str_c(V1, V2, sep = "-")) |>
    pull(pair_string)

# Sample PD pairs from doubletrouble that are not PD pairs in DupGen_finder
examples <- p1[!p1 %in% p2] |> head(n = 5)

# Check whether they are PD pairs or not
ath_annot <- pdata$annotation$arabidopsis.thaliana
range_examples <- lapply(examples, function(x) {
    g1 <- paste0("ara_", strsplit(x, "-")[[1]][1])
    g2 <- paste0("ara_", strsplit(x, "-")[[1]][2])
    ranges <- ath_annot[ath_annot$gene %in% c(g1, g2)]
    return(ranges)
})
```

```
range_examples
## [[1]]
## GRanges object with 2 ranges and 1 metadata column:
##       seqnames        ranges strand |          gene
##          <Rle>     <IRanges>  <Rle> |   <character>
##   128    ara_1 428650-430720      - | ara_AT1G02220
##   130    ara_1 437860-439559      - | ara_AT1G02250
##   -------
##   seqinfo: 7 sequences from an unspecified genome; no seqlengths
##
## [[2]]
## GRanges object with 2 ranges and 1 metadata column:
##       seqnames        ranges strand |          gene
##          <Rle>     <IRanges>  <Rle> |   <character>
##   127    ara_1 427548-427811      - | ara_AT1G02210
##   130    ara_1 437860-439559      - | ara_AT1G02250
##   -------
##   seqinfo: 7 sequences from an unspecified genome; no seqlengths
##
## [[3]]
```

```
## GRanges object with 2 ranges and 1 metadata column:
##         seqnames        ranges strand |          gene
##            <Rle>     <IRanges>  <Rle> |   <character>
##   17117    ara_4 656407-659178      - | ara_AT4G01520
##   17120    ara_4 673862-676445      - | ara_AT4G01550
##   -------
##   seqinfo: 7 sequences from an unspecified genome; no seqlengths
##
## [[4]]
## GRanges object with 2 ranges and 1 metadata column:
##         seqnames            ranges strand |          gene
##            <Rle>         <IRanges>  <Rle> |   <character>
##   15329    ara_3 17882465-17884515      + | ara_AT3G48290
##   15332    ara_3 17887996-17889942      + | ara_AT3G48310
##   -------
##   seqinfo: 7 sequences from an unspecified genome; no seqlengths
##
## [[5]]
## GRanges object with 2 ranges and 1 metadata column:
##         seqnames            ranges strand |          gene
##            <Rle>         <IRanges>  <Rle> |   <character>
##   25845    ara_5 21378702-21379744      + | ara_AT5G52720
##   25847    ara_5 21382482-21383432      + | ara_AT5G52740
##   -------
##   seqinfo: 7 sequences from an unspecified genome; no seqlengths
```

In these first five examples of pairs that are classified as PD pairs by **doubletrouble**, but not by **DupGen_finder**, we can see based on the numbers in row names that they are indeed very close, separated by only a few genes (<10). Thus, they are true PD pairs that **DupGen_finder** failed to classify as PD pairs, or removed in their undocumented filtering (described above).

Session info

This document was created under the following conditions:

```
## - Session info ---------------------------------------------------------------
##  setting  value
##  version  R version 4.4.1 (2024-06-14)
##  os       Ubuntu 22.04.4 LTS
##  system   x86_64, linux-gnu
##  ui       X11
##  language (EN)
##  collate  en_US.UTF-8
##  ctype    en_US.UTF-8
##  tz       Europe/Brussels
##  date     2024-07-26
##  pandoc   3.1.1 @ /usr/lib/rstudio/resources/app/bin/quarto/bin/tools/ (via rmarkdown)
##
## - Packages -------------------------------------------------------------------
```

Supplementary Text S5: Runtime benchmark

```
##  package              * version   date (UTC) lib source
##  abind                  1.4-5     2016-07-21 [1] CRAN (R 4.4.1)
##  ade4                   1.7-22    2023-02-06 [1] CRAN (R 4.4.1)
##  AnnotationDbi          1.66.0    2024-05-01 [1] Bioconductor 3.19 (R 4.4.1)
##  ape                    5.8       2024-04-11 [1] CRAN (R 4.4.1)
##  Biobase                2.64.0    2024-04-30 [1] Bioconductor 3.19 (R 4.4.1)
##  BiocGenerics           0.50.0    2024-04-30 [1] Bioconductor 3.19 (R 4.4.1)
##  BiocIO                 1.14.0    2024-04-30 [1] Bioconductor 3.19 (R 4.4.1)
##  BiocManager            1.30.23   2024-05-04 [1] CRAN (R 4.4.1)
##  BiocParallel           1.38.0    2024-04-30 [1] Bioconductor 3.19 (R 4.4.1)
##  BiocStyle            * 2.32.1    2024-06-16 [1] Bioconductor 3.19 (R 4.4.1)
##  Biostrings             2.72.1    2024-06-02 [1] Bioconductor 3.19 (R 4.4.1)
##  bit                    4.0.5     2022-11-15 [1] CRAN (R 4.4.1)
##  bit64                  4.0.5     2020-08-30 [1] CRAN (R 4.4.1)
##  bitops                 1.0-7     2021-04-24 [1] CRAN (R 4.4.1)
##  blob                   1.2.4     2023-03-17 [1] CRAN (R 4.4.1)
##  bookdown               0.40      2024-07-02 [1] CRAN (R 4.4.1)
##  cachem                 1.1.0     2024-05-16 [1] CRAN (R 4.4.1)
##  cli                    3.6.3     2024-06-21 [1] CRAN (R 4.4.1)
##  coda                   0.19-4.1  2024-01-31 [1] CRAN (R 4.4.1)
##  codetools              0.2-20    2024-03-31 [1] CRAN (R 4.4.1)
##  colorspace             2.1-0     2023-01-23 [1] CRAN (R 4.4.1)
##  crayon                 1.5.3     2024-06-20 [1] CRAN (R 4.4.1)
##  curl                   5.2.1     2024-03-01 [1] CRAN (R 4.4.1)
##  DBI                    1.2.3     2024-06-02 [1] CRAN (R 4.4.1)
##  DelayedArray           0.30.1    2024-05-07 [1] Bioconductor 3.19 (R 4.4.1)
##  digest                 0.6.36    2024-06-23 [1] CRAN (R 4.4.1)
##  doParallel             1.0.17    2022-02-07 [1] CRAN (R 4.4.1)
##  doubletrouble        * 1.4.1     2024-07-25 [1] Bioconductor
##  dplyr                * 1.1.4     2023-11-17 [1] CRAN (R 4.4.1)
##  evaluate               0.24.0    2024-06-10 [1] CRAN (R 4.4.1)
##  fansi                  1.0.6     2023-12-08 [1] CRAN (R 4.4.1)
##  fastmap                1.2.0     2024-05-15 [1] CRAN (R 4.4.1)
##  forcats              * 1.0.0     2023-01-29 [1] CRAN (R 4.4.1)
##  foreach                1.5.2     2022-02-02 [1] CRAN (R 4.4.1)
##  generics               0.1.3     2022-07-05 [1] CRAN (R 4.4.1)
##  GenomeInfoDb           1.40.1    2024-05-24 [1] Bioconductor 3.19 (R 4.4.1)
##  GenomeInfoDbData       1.2.12    2024-07-24 [1] Bioconductor
##  GenomicAlignments      1.40.0    2024-04-30 [1] Bioconductor 3.19 (R 4.4.1)
##  GenomicFeatures        1.56.0    2024-04-30 [1] Bioconductor 3.19 (R 4.4.1)
##  GenomicRanges          1.56.1    2024-06-12 [1] Bioconductor 3.19 (R 4.4.1)
##  ggnetwork              0.5.13    2024-02-14 [1] CRAN (R 4.4.1)
##  ggplot2              * 3.5.1     2024-04-23 [1] CRAN (R 4.4.1)
##  glue                   1.7.0     2024-01-09 [1] CRAN (R 4.4.1)
##  gtable                 0.3.5     2024-04-22 [1] CRAN (R 4.4.1)
##  here                 * 1.0.1     2020-12-13 [1] CRAN (R 4.4.1)
##  hms                    1.1.3     2023-03-21 [1] CRAN (R 4.4.1)
##  htmltools              0.5.8.1   2024-04-04 [1] CRAN (R 4.4.1)
##  htmlwidgets            1.6.4     2023-12-06 [1] CRAN (R 4.4.1)
##  httr                   1.4.7     2023-08-15 [1] CRAN (R 4.4.1)
##  igraph                 2.0.3     2024-03-13 [1] CRAN (R 4.4.1)
```

Supplementary Text S5: Runtime benchmark

```
##  intergraph            2.0-4     2024-02-01 [1] CRAN (R 4.4.1)
##  IRanges               2.38.1    2024-07-03 [1] Bioconductor 3.19 (R 4.4.1)
##  iterators             1.0.14    2022-02-05 [1] CRAN (R 4.4.1)
##  jsonlite              1.8.8     2023-12-04 [1] CRAN (R 4.4.1)
##  KEGGREST              1.44.1    2024-06-19 [1] Bioconductor 3.19 (R 4.4.1)
##  knitr                 1.48      2024-07-07 [1] CRAN (R 4.4.1)
##  lattice               0.22-6    2024-03-20 [1] CRAN (R 4.4.1)
##  lifecycle             1.0.4     2023-11-07 [1] CRAN (R 4.4.1)
##  lubridate           * 1.9.3     2023-09-27 [1] CRAN (R 4.4.1)
##  magrittr              2.0.3     2022-03-30 [1] CRAN (R 4.4.1)
##  MASS                  7.3-61    2024-06-13 [1] CRAN (R 4.4.1)
##  Matrix                1.7-0     2024-04-26 [1] CRAN (R 4.4.1)
##  MatrixGenerics        1.16.0    2024-04-30 [1] Bioconductor 3.19 (R 4.4.1)
##  matrixStats           1.3.0     2024-04-11 [1] CRAN (R 4.4.1)
##  mclust                6.1.1     2024-04-29 [1] CRAN (R 4.4.1)
##  memoise               2.0.1     2021-11-26 [1] CRAN (R 4.4.1)
##  MSA2dist              1.8.0     2024-04-30 [1] Bioconductor 3.19 (R 4.4.1)
##  munsell               0.5.1     2024-04-01 [1] CRAN (R 4.4.1)
##  network               1.18.2    2023-12-05 [1] CRAN (R 4.4.1)
##  networkD3             0.4       2017-03-18 [1] CRAN (R 4.4.1)
##  nlme                  3.1-165   2024-06-06 [1] CRAN (R 4.4.1)
##  patchwork           * 1.2.0     2024-01-08 [1] CRAN (R 4.4.1)
##  pheatmap              1.0.12    2019-01-04 [1] CRAN (R 4.4.1)
##  pillar                1.9.0     2023-03-22 [1] CRAN (R 4.4.1)
##  pkgconfig             2.0.3     2019-09-22 [1] CRAN (R 4.4.1)
##  png                   0.1-8     2022-11-29 [1] CRAN (R 4.4.1)
##  purrr               * 1.0.2     2023-08-10 [1] CRAN (R 4.4.1)
##  pwalign               1.0.0     2024-04-30 [1] Bioconductor 3.19 (R 4.4.1)
##  R6                    2.5.1     2021-08-19 [1] CRAN (R 4.4.1)
##  RColorBrewer          1.1-3     2022-04-03 [1] CRAN (R 4.4.1)
##  Rcpp                  1.0.13    2024-07-17 [1] CRAN (R 4.4.1)
##  RCurl                 1.98-1.16 2024-07-11 [1] CRAN (R 4.4.1)
##  readr               * 2.1.5     2024-01-10 [1] CRAN (R 4.4.1)
##  restfulr              0.0.15    2022-06-16 [1] CRAN (R 4.4.1)
##  rjson                 0.2.21    2022-01-09 [1] CRAN (R 4.4.1)
##  rlang                 1.1.4     2024-06-04 [1] CRAN (R 4.4.1)
##  rmarkdown             2.27      2024-05-17 [1] CRAN (R 4.4.1)
##  rprojroot             2.0.4     2023-11-05 [1] CRAN (R 4.4.1)
##  Rsamtools             2.20.0    2024-04-30 [1] Bioconductor 3.19 (R 4.4.1)
##  RSQLite               2.3.7     2024-05-27 [1] CRAN (R 4.4.1)
##  rstudioapi            0.16.0    2024-03-24 [1] CRAN (R 4.4.1)
##  rtracklayer           1.64.0    2024-04-30 [1] Bioconductor 3.19 (R 4.4.1)
##  S4Arrays              1.4.1     2024-05-20 [1] Bioconductor 3.19 (R 4.4.1)
##  S4Vectors             0.42.1    2024-07-03 [1] Bioconductor 3.19 (R 4.4.1)
##  scales                1.3.0     2023-11-28 [1] CRAN (R 4.4.1)
##  seqinr                4.2-36    2023-12-08 [1] CRAN (R 4.4.1)
##  sessioninfo           1.2.2     2021-12-06 [1] CRAN (R 4.4.1)
##  SparseArray           1.4.8     2024-05-24 [1] Bioconductor 3.19 (R 4.4.1)
##  statnet.common        4.9.0     2023-05-24 [1] CRAN (R 4.4.1)
##  stringi               1.8.4     2024-05-06 [1] CRAN (R 4.4.1)
##  stringr             * 1.5.1     2023-11-14 [1] CRAN (R 4.4.1)
```

Supplementary Text S5: Runtime benchmark

```
##  SummarizedExperiment   1.34.0    2024-05-01 [1] Bioconductor 3.19 (R 4.4.1)
##  syntenet               1.6.0     2024-05-01 [1] Bioconductor 3.19 (R 4.4.1)
##  tibble               * 3.2.1     2023-03-20 [1] CRAN (R 4.4.1)
##  tidyr                * 1.3.1     2024-01-24 [1] CRAN (R 4.4.1)
##  tidyselect             1.2.1     2024-03-11 [1] CRAN (R 4.4.1)
##  tidyverse            * 2.0.0     2023-02-22 [1] CRAN (R 4.4.1)
##  timechange             0.3.0     2024-01-18 [1] CRAN (R 4.4.1)
##  tzdb                   0.4.0     2023-05-12 [1] CRAN (R 4.4.1)
##  UCSC.utils             1.0.0     2024-04-30 [1] Bioconductor 3.19 (R 4.4.1)
##  utf8                   1.2.4     2023-10-22 [1] CRAN (R 4.4.1)
##  vctrs                  0.6.5     2023-12-01 [1] CRAN (R 4.4.1)
##  withr                  3.0.0     2024-01-16 [1] CRAN (R 4.4.1)
##  xfun                   0.46      2024-07-18 [1] CRAN (R 4.4.1)
##  XML                    3.99-0.17 2024-06-25 [1] CRAN (R 4.4.1)
##  XVector                0.44.0    2024-04-30 [1] Bioconductor 3.19 (R 4.4.1)
##  yaml                   2.3.9     2024-07-05 [1] CRAN (R 4.4.1)
##  zlibbioc               1.50.0    2024-04-30 [1] Bioconductor 3.19 (R 4.4.1)
##
##  [1] /home/faalm/R/x86_64-pc-linux-gnu-library/4.4
##  [2] /usr/local/lib/R/site-library
##  [3] /usr/lib/R/site-library
##  [4] /usr/lib/R/library
##
## ------------------------------------------------------------------------------
```