Creating the Soybean Expression Atlas v2

Fabricio Almeida-Silva [1]

[1]VIB-UGent Center for Plant Systems Biology

16 March 2023

Contents

1 Overview

Here, I will describe the code I used to create the Soybean Expression Atlas using the R package *bears*.

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

# Load required packages
library(here)
library(bears)
library(tidyverse)
```

2 Downloading data

We will download samples that have not been downloaded yet. The data frames with sample metadata are stored in the `data/` directory.

```
options(timeout = 1e+10)

#----Get sample metadata------------------------------------------------------
term <- "Glycine max[ORGN] AND RNA-seq[STRA]"
metadata_all <- create_sample_info(term, retmax = 10000)

# Remove unsupported technologies
metadata_all <- metadata_all[!grepl(
    "454|SOLiD|PacBio|Torrent|AB 310|Complete Genomics|ION",
    metadata_all$Instrument
), ]

## Save metadata object
save(
    metadata_all,
    file = here("data", "metadata_all.rda"),
    compress = "xz"
)

## Create directory structure
rootdir <- here("results")
ds <- create_dir_structure(rootdir = rootdir)
save(ds, file = here("data", "ds.rda"), compress = "xz")

#----Downloading samples------------------------------------------------------
urls <- get_url_ena(metadata_all)
download <- download_from_ena(urls = urls, fastqdir = ds$fastqdir)
```

Now, let's check the file integrity of downloaded files. Here, I will do it for the whole atlas, including previous versions, just as a sanity check.

```
#----Check file integrity with md5sum check----------------------------------

# Remove runs that were not downloaded
```

```
downloaded <- fastq_exists(metadata_all, fastqdir = ds$fastqdir)
downloaded <- downloaded$Run[!is.na(downloaded$Status)]
metadata_atlas_v2 <- metadata_all[metadata_all$Run %in% downloaded, ]

## Check md5sum
integrity <- check_md5(
    run_accessions = metadata_all$Run,
    fastqdir = ds$fastqdir
)

failed_corrupt <- integrity[integrity$Status == FALSE, "Run"]
failed_corrupt <- unique(as.character(failed_corrupt))
```

The `failed_corrupt` object is a character vector containing run accessions that failed the integrity check and, thus, must be re-downloaded. Now, let's also check for files that were not downloaded at all. Here, we will only consider for re-download runs that are part of BioProjects with effectively downloaded runs. In other words, if all runs of a BioProject were not downloaded, I will ignore them (they are probably not available on EBI).

```
# Get download status
dstatus <- fastq_exists(metadata_all, fastqdir = ds$fastqdir)

# Get BioProject info of failed runs
failed_bioproject <- dstatus %>%
    full_join(., metadata_all)

# Get BioProjects with missing percentage (m) = 0 < m < 100
bioprojects_download <- failed_bioproject %>%
    group_by(BioProject) %>%
    summarise(perc = sum(is.na(Status)) / length(BioProject)) %>%
    filter(perc != 0 & perc != 1)


# Get vector of runs from these BioProjects to re-download
failed_nd <- failed_bioproject %>%
    filter(is.na(Status) & BioProject %in% bioprojects_download$BioProject) %>%
    select(Run)

failed_nd <- unique(as.character(failed_nd$Run))
```

In the end, 33 runs failed the integrity check (i.e., they were downloaded, but md5sums were different from the reference), and 33 runs failed to download (i.e., some runs from the BioProject were downloaded, but others were not). Re-downloading failed runs:

```
# Metadata data frame of failed runs only
todownload <- c(failed_corrupt, failed_nd)
metadata_failed <- metadata_all[metadata_all$Run %in% todownload, ]

# Download again
urls_missing <- get_url_ena(metadata_failed)
```

```
options(timeout = 1e10)
download_missing <- download_from_ena(
    metadata_failed,
    urls = urls_missing,
    fastqdir = ds$fastqdir
)
```

Done! Finally, let's create a final metadata data frame with all samples we have:

```
dstatus <- fastq_exists(metadata_all, fastqdir = ds$fastqdir)
ok_runs <- unique(dstatus$Run[dstatus$Status == "OK"])

metadata_atlas_v2 <- metadata_all[metadata_all$Run %in% ok_runs, ]

## Save a data frame containing metadata for all samples of SEA v2
save(
    metadata_atlas_v2_downloaded,
    file = here::here("data", "metadata_atlas_v2_downloaded.rda"),
    compress = "xz"
)
```

3 Sequence QC and filtering

Here, we will remove sequence adapters and low-quality bases with **fastp**.

```
metadata_atlas_v2 <- metadata_atlas_v2_downloaded

runs_fastp <- fastq_exists(metadata_atlas_v2, ds$fastqdir) %>%
    dplyr::filter(!is.na(Status)) %>%
    dplyr::pull(Run)

metadata_fastp <- metadata_atlas_v2[metadata_atlas_v2$Run %in% runs_fastp, ]

# Run fastp
fastp_status <- trim_reads(
    metadata_fastp,
    fastqdir = ds$fastqdir,
    filtdir = ds$filtdir,
    qcdir = ds$qcdir,
    threads = 16,
    delete_raw = TRUE
)

# Get a metadata data frame with only reads that have undergone filtering
filtered_reads <- unique(
    gsub(
        "(\\.fastq.*)|(_.*)", "",
        basename(list.files(ds$filtdir, pattern = "fastq.gz"))
    )
)
```

```
metadata_atlas_v2_filtered <- metadata_atlas_v2[
    metadata_atlas_v2$Run %in% filtered_reads,
]

save(
    metadata_atlas_v2_filtered, compress = "xz",
    file = here("data", "metadata_atlas_v2_filtered.rda")
)
```

Now, let's import and save read filtering stats.

```
# Get a data frame of summary stats from fastp
fastp_stats <- summary_stats_fastp(ds$qcdir)

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

Finally, we will remove low-quality files based on the following criteria:

1. Mean length after filtering < 40
2. Q20 rate $<80\%$ after filtering.

```
# Remove files whose mean length after filtering is <40 and Q20 <80%
keep <- fastp_stats %>%
    filter(after_meanlength >= 40) %>%
    filter(after_q20rate >= 0.8) %>%
    pull(Sample)

filtered_metadata <- metadata_atlas_v2_filtered[
    metadata_atlas_v2_filtered$Run %in% keep,
]
rownames(filtered_metadata) <- 1:nrow(filtered_metadata)
```

4 Quantifying transcript abundance

Now, we will quantify transcript abundance with **salmon**. To do that, we first need to index the reference transcriptome.

```
# Index transcriptome
transcriptome_path <- here("data", "gmax_transcriptome.fa.gz")

idx_salmon <- salmon_index(
    salmonindex = ds$salmonindex,
    transcriptome_path = transcriptome_path
)

idx_salmon
```

Then, we can quantify transcript abundance.

```
# Quantify transcript abundance
quant_salmon <- salmon_quantify(
    filtered_metadata,
    filtdir = ds$filtdir,
    salmonindex = ds$salmonindex,
    salmondir = ds$salmondir,
    threads = 40
)

# Checking percentage of samples that ran sucessfully
n_ok <- nrow(quant_salmon[!is.na(quant_salmon$status), ])
n_ok / nrow(quant_salmon)
```

salmon was run sucessfully for 100% of the samples. Great! Now, let's obtain mapping rates for each BioSample to see whether or not we need to discard samples. Here, we will remove samples with mapping rate <50% (i.e., less than 50% of the reads failed to "map").

```
# Get a data frame of mapping rate per BioSample
biosamples <- unique(filtered_metadata$BioSample)
mapping_rate <- summary_stats_salmon(ds$salmondir, biosamples)

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

# Removing BioSamples with mapping rate <50%
biosamples_to_keep <- mapping_rate %>%
    filter(Mapping_rate >= 50) %>%
    pull(BioSample)

# Update metadata data frame to keep only samples that passed the filtering step
final_metadata_atlas_v2 <- filtered_metadata %>%
    filter(BioSample %in% biosamples_to_keep)
```

82% of the samples (5481/6644) passed the filtering step (i.e., had mapping rates >=50%). Thus, this is the final number of samples in the Soybean Expression Atlas v2.

To conclude, let's just fix and standardize tissue names in the metadata data frame and save it.

```
final_metadata_classified_tissues <- final_metadata_atlas_v2 %>%
    mutate(Tissue_PO = str_to_lower(Tissue)) %>%
    mutate(Tissue_PO = str_replace_all(
        Tissue_PO,
        c(".*nodule.*" = "nodule",
          ".*leaves and roots.*" = "whole plant",
          ".*leaves.*" = "leaf",
          ".*leaf.*" = "leaf",
          ".*trifoliate.*" = "leaf",
          ".*seed coat.*" = "seed coat",
          ".*crown.*" = "root",
```

```
                ".*shoot.*" = "shoot",
                ".*stem.*" = "shoot",
                ".*hypocotyl.*" = "hypocotyl",
                ".*pod.*" = "pod",
                ".*root.*" = "",
                ".*embryo.*" = "embryo",
                ".*seedling.*" = "seedling",
                ".*cytoledon.*" = "cotyledon",
                ".*cotyledon.*" = "cotyledon",
                ".*seed.*" = "seed",
                ".*flower.*" = "flower",
                ".*petiole.*" = "petiole",
                ".*meristem.*" = "meristem",
                ".*radicle*" = "radicle",
                ".*epicotyl.*" = "epicotyl",
                ".*whole.*" = "whole plant",
                ".*roots + leaves.*" = "whole plant",
                ".*sprout*" = "seedling",
                ".*ovule.*" = "flower",
                ".*suspensor*" = "suspensor",
                ".*floral.*" = "flower",
                ".*leaf bud.*" = "leaf",
                ".*anther*" = "flower",
                ".*stem and leaf*" = "whole plant",
                ".*ovary.*" = "flower",
                ".*embryo.*" = "embryo",
                ".*embryo.*" = "embryo",
                ".*embryo.*" = "embryo"
            )
        )
        )

    write_tsv(
        final_metadata_classified_tissues,
        file = here("products", "tables", "final_metadata_classified_tissues.tsv")
    )
```

The file *final_metadata_classified_tissues.tsv* was manually edited to include classifications for missing tissues, and then it was renamed to *final_metadata_classified_atlas_v2.tsv*. This file only contains BioSample-level information (runs were not considered).

Session information

```
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     2023-03-16
##  pandoc   2.19.2 @ /usr/lib/rstudio/resources/app/bin/quarto/bin/tools/ (via rmarkdown)
##
## - Packages ---------------------------------------------------------------
##  package     * version date (UTC) lib source
##  BiocManager   1.30.20 2023-02-24 [1] CRAN (R 4.2.2)
##  BiocStyle   * 2.25.0  2022-06-15 [1] Github (Bioconductor/BiocStyle@7150c28)
##  bookdown      0.33    2023-03-06 [1] CRAN (R 4.2.2)
##  cli           3.6.0   2023-01-09 [1] CRAN (R 4.2.2)
##  digest        0.6.31  2022-12-11 [1] CRAN (R 4.2.2)
##  evaluate      0.20    2023-01-17 [1] CRAN (R 4.2.2)
##  fastmap       1.1.1   2023-02-24 [1] CRAN (R 4.2.2)
##  htmltools     0.5.4   2022-12-07 [1] CRAN (R 4.2.2)
##  knitr         1.42    2023-01-25 [1] CRAN (R 4.2.2)
##  rlang         1.0.6   2022-09-24 [1] CRAN (R 4.2.1)
##  rmarkdown     2.20    2023-01-19 [1] CRAN (R 4.2.2)
##  rstudioapi    0.14    2022-08-22 [1] CRAN (R 4.2.1)
##  sessioninfo   1.2.2   2021-12-06 [1] CRAN (R 4.2.0)
##  xfun          0.37    2023-01-31 [1] CRAN (R 4.2.2)
##  yaml          2.3.7   2023-01-23 [1] CRAN (R 4.2.2)
##
##  [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
##
## --------------------------------------------------------------------------
```