Motif analyses of GRN and PPI networks

Fabricio Almeida-Silva

2023-05-03

```
library(here)
library(magrene)
library(BiocParallel)
library(tidyverse)
library(ComplexHeatmap)
library(RColorBrewer)
library(ggstatsplot)
library(ggpubr)

set.seed(123)
dup_palette <- c(SSD = "#1984c5", WGD = "#ffb400")
```

Overview

Here, we will describe the code to:

1. Calculate motif frequencies in the networks
2. Compare observed frequencies to null distributions
3. Evaluate interaction similarity between paralogous gene pairs.
4. Perform a functional enrichment analysis on genes in motifs.

Calculating motif frequencies

Here, we will calculate motif frequencies for each species, but we will do it separately for each Ks peak we identified previously.

To start, let's load the required data.

```
# Load paralogs
load(here("products", "result_files", "duplicate_pairs_with_boundaries.rda"))

## Simplifying object name
paralogs <- duplicate_pairs_with_boundaries
rm(duplicate_pairs_with_boundaries)


# Load GRN
load(here("products", "result_files", "grns.rda"))


# Load PPI network
load(here("products", "result_files", "ppi.rda"))
```

```r
## Keep only edges from PPI network and remove all other columns
ppi <- lapply(ppi, function(x) return(x[, 1:2]))
```

Now, let's get motifs for each species and peak.

```r
# Define function to count all motifs at once.
count_all_motifs <- function(species = NULL, count = FALSE) {

    pairs <- paralogs[[species]][, c(1, 2, 4, 5)]
    names(pairs) <- c("dup1", "dup2", "mode", "peak")

    # Split by peak
    pairs_peak <- split(pairs, pairs$peak)

    # Count motifs for WGD and SSD-derived pairs
    motifs <- lapply(pairs_peak, function(y) {
        ssd <- y[y$mode == "SSD", 1:2]
        wgd <- y[y$mode == "WGD", 1:2]

        ## PPI V
        ssd_ppiv <- find_ppi_v(ppi[[species]], ssd, count_only = count)
        wgd_ppiv <- find_ppi_v(ppi[[species]], wgd, count_only = count)
        message("Finished identifying PPI V motifs.")

        ## V
        ssd_v <- find_v(grns[[species]], ssd, count_only = count)
        wgd_v <- find_v(grns[[species]], wgd, count_only = count)
        message("Finished identifying V motifs.")

        ## Lambda
        ssd_lambda <- find_lambda(grns[[species]], ssd, count_only = FALSE)
        wgd_lambda <- find_lambda(grns[[species]], wgd, count_only = FALSE)
        message("Finished identifying lambda motifs.")

        ## Delta
        ssd_delta <- find_delta(
            edgelist_ppi = ppi[[species]], lambda_vec = ssd_lambda,
            count_only = count
        )
        wgd_delta <- find_delta(
            edgelist_ppi = ppi[[species]], lambda_vec = wgd_lambda,
            count_only = count
        )
        message("Finished identifying delta motifs.")

        ## Bifan
        ssd_bifan <- find_bifan(
            paralogs = ssd, lambda_vec = ssd_lambda, count_only = count
        )
        wgd_bifan <- find_bifan(
            paralogs = wgd, lambda_vec = wgd_lambda, count_only = count
        )
        message("Finished identifying bifan motifs.")
```

```
        ## Result list
        res_ssd <- list(
            V = ssd_v,
            V_PPI = ssd_ppiv,
            lambda = length(ssd_lambda),
            delta = ssd_delta,
            bifan = ssd_bifan
        )
        res_wgd <- list(
            V = wgd_v,
            V_PPI = wgd_ppiv,
            lambda = length(wgd_lambda),
            delta = wgd_delta,
            bifan = wgd_bifan
        )
        res_list <- list(
            SSD = res_ssd,
            WGD = res_wgd
        )
        return(res_list)
    })
    names(motifs) <- paste0("peak", seq_along(pairs_peak))
    return(motifs)
}


dir.create(here("products", "result_files", "motifs"))

# Get motifs for each species separately
motifs_gmax <- count_all_motifs("Gmax")
save(
    motifs_gmax,
    file = here("products", "result_files", "motifs", "motifs_gmax.rda"),
    compress = "xz"
)

motifs_athaliana <- count_all_motifs("Athaliana")
save(
    motifs_athaliana,
    file = here("products", "result_files", "motifs", "motifs_athaliana.rda"),
    compress = "xz"
)

motifs_ptrichocarpa <- count_all_motifs("Ptrichocarpa")
save(
    motifs_ptrichocarpa,
    file = here("products", "result_files", "motifs", "motifs_ptrichocarpa.rda"),
    compress = "xz"
)

motifs_slycopersicum <- count_all_motifs("Slycopersicum")
save(
    motifs_slycopersicum,
```

```
    file = here("products", "result_files", "motifs", "motifs_slycopersicum.rda"),
    compress = "xz"
)

motifs_vvinifera <- count_all_motifs("Vvinifera")
save(
    motifs_vvinifera,
    file = here("products", "result_files", "motifs", "motifs_vvinifera.rda"),
    compress = "xz"
)

motifs_osativa <- count_all_motifs("Osativa")
save(
    motifs_osativa,
    file = here("products", "result_files", "motifs", "motifs_osativa.rda"),
    compress = "xz"
)

motifs_zmays <- count_all_motifs("Zmays")
save(
    motifs_zmays,
    file = here("products", "result_files", "motifs", "motifs_zmays.rda"),
    compress = "xz"
)
```

And finally, only motif counts, not character representations.

```
species <- names(grns)

motif_counts <- lapply(species, function(x) {
    message("Working on species ", x)
    return(count_all_motifs(x, count = TRUE))
})
names(motif_counts) <- species

# Save observed motif frequencies
save(
    motif_counts,
    file = here("products", "result_files", "motifs", "motif_counts.rda"),
    compress = "xz"
)
```

Evaluating significance of motif frequencies

To evaluate the significance of the motif counts we observed, we will generate a null distribution by simulating 1000 degree-preserving simulated networks. Then, we calculate a Z-score for each motif, by species, and by Ks peaks.

```
# Define function to create null distribution for each species
get_null_counts <- function(species, n = 1000,
                            bp_param = BiocParallel::SerialParam(RNGseed = 123)) {

    pairs <- paralogs[[species]][, c(1, 2, 4, 5)]
    names(pairs) <- c("dup1", "dup2", "mode", "peak")
```

```
    # Split by peak
    pairs_peak <- split(pairs, pairs$peak)

    motifs <- lapply(pairs_peak, function(y) {
        ssd <- y[y$mode == "SSD", 1:2]
        wgd <- y[y$mode == "WGD", 1:2]

        ssd_nulls <- generate_nulls(
            grns[[species]], ssd, ppi[[species]], n = n, bp_param = bp_param
        )
        wgd_nulls <- generate_nulls(
            grns[[species]], wgd, ppi[[species]], n = n, bp_param = bp_param
        )

        ## Result list
        res_list <- list(
            SSD = ssd_nulls,
            WGD = wgd_nulls
        )
        return(res_list)
    })
    names(motifs) <- paste0("peak", seq_along(pairs_peak))
    return(motifs)
}

# Get null distros for each species
bp <- BiocParallel::MulticoreParam(workers = 10, RNGseed = 123)

nulls_vvinifera <- get_null_counts("Vvinifera", bp_param = bp)
save(
    nulls_vvinifera,
    file = here("products", "result_files", "motifs", "nulls_vvinifera.rda"),
    compress = "xz"
)

nulls_slycopersicum <- get_null_counts("Slycopersicum", bp_param = bp)
save(
    nulls_slycopersicum,
    file = here("products", "result_files", "motifs", "nulls_slycopersicum.rda"),
    compress = "xz"
)

nulls_osativa <- get_null_counts("Osativa", bp_param = bp)
save(
    nulls_osativa,
    file = here("products", "result_files", "motifs", "nulls_osativa.rda"),
    compress = "xz"
)

nulls_ptrichocarpa <- get_null_counts("Ptrichocarpa", bp_param = bp)
save(
    nulls_ptrichocarpa,
    file = here("products", "result_files", "motifs", "nulls_ptrichocarpa.rda"),
```

```
    compress = "xz"
)

nulls_athaliana <- get_null_counts("Athaliana", bp_param = bp)
save(
    nulls_athaliana,
    file = here("products", "result_files", "motifs", "nulls_athaliana.rda"),
    compress = "xz"
)

nulls_zmays <- get_null_counts("Zmays", bp_param = bp)
save(
    nulls_zmays,
    file = here("products", "result_files", "motifs", "nulls_zmays.rda"),
    compress = "xz"
)

nulls_gmax <- get_null_counts("Gmax", bp_param = bp)
save(
    nulls_gmax,
    file = here("products", "result_files", "motifs", "nulls_gmax.rda"),
    compress = "xz"
)
```

Now that we have null distros for each species and peak, let's calculate Z-scores for observed frequencies.

```
# Load observed frequencies
load(
    here("products", "result_files", "motifs", "motif_counts.rda")
)
observed <- motif_counts

# Create list of nulls that match the structure of list of observed counts
## Load null objects
null_files <- list.files(
    here("products", "result_files", "motifs"), pattern = "nulls*",
    full.names = TRUE
)
for(file in null_files) {
    load(file)
}

## Combine them all in a list, in the same order of `observed`
names(observed)
```

```
## [1] "Osativa"       "Zmays"         "Vvinifera"     "Gmax"
## [5] "Slycopersicum" "Ptrichocarpa"  "Athaliana"
```

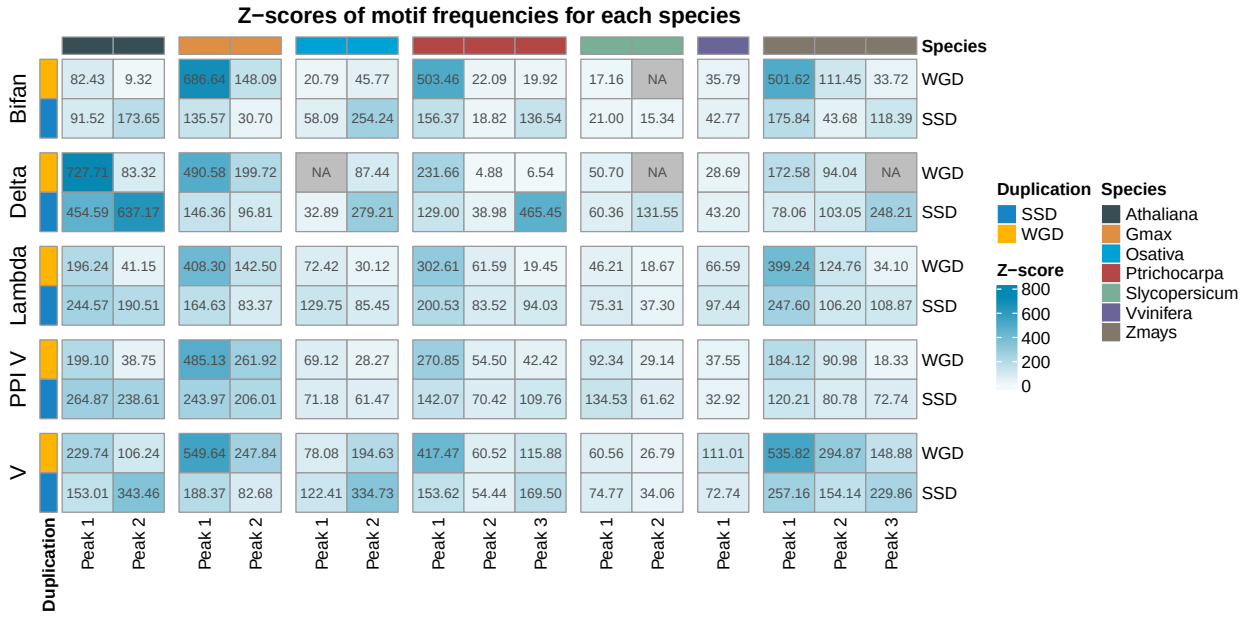
```
nulls <- list(
    Osativa = nulls_osativa,
    Zmays = nulls_zmays,
    Vvinifera = nulls_vvinifera,
    Gmax = nulls_gmax,
    Slycopersicum = nulls_slycopersicum,
    Ptrichocarpa = nulls_ptrichocarpa,
```

```
    Athaliana = nulls_athaliana
)

# Calculate Z-scores for each species and summarize results in a data frame
zscores_df <- Reduce(rbind, lapply(seq_along(observed), function(x) {

    fobs <- observed[[x]]
    fnulls <- nulls[[x]]
    species <- names(observed)[x]
    message("Working on species ", species)

    z <- Reduce(rbind, lapply(seq_along(fobs), function(y) {
        peak <- names(fobs)[y]

        # Fix name mismatch - 'V_PPI' in obs, 'PPI_V' in nulls
        names(fobs[[y]]$WGD)[2] <- "PPI_V"
        names(fobs[[y]]$SSD)[2] <- "PPI_V"

        wgd_z <- calculate_Z(fobs[[y]]$WGD, fnulls[[y]]$WGD)
        ssd_z <- calculate_Z(fobs[[y]]$SSD, fnulls[[y]]$SSD)

        summary_df <- data.frame(
            Motif = rep(names(wgd_z), 2),
            Z = c(wgd_z, ssd_z),
            Mode = c(rep("WGD", 5), rep("SSD", 5)),
            Species = species,
            Peak = peak
        )
        return(summary_df)
    }))
    return(z)
}))

# Reshape data from long to wide format
z_metadata <- zscores_df %>%
    mutate(species_peak = str_c(Species, Peak, sep = "_")) %>%
    mutate(motif_mode = str_c(Motif, Mode, sep = "_"))

z_pdata <- z_metadata %>%
    tidyr::pivot_wider(
        names_from = species_peak, # cols
        id_cols = motif_mode, # rows
        values_from = Z
    ) %>%
    column_to_rownames(., var = "motif_mode") %>%
    as.matrix()
row_order <- c("PPI_V_WGD", "PPI_V_SSD", "V_WGD", "V_SSD",
               "delta_WGD", "delta_SSD", "lambda_WGD", "lambda_SSD",
               "bifan_WGD", "bifan_SSD")
z_pdata <- z_pdata[row_order, ]
```

Now, let's plot the results as a heatmap. Here, we will consider significant Z-scores $> |2|$ (absolute value). Z-scores below this threshold will be set to NA.

```r
# Set non-significant Z-scores to NA
z_pdata[z_pdata < 2 & z_pdata > -2] <- NA
z_pdata[is.nan(z_pdata)] <- NA

# Create row annotation, column annotation, and annotation colors
## Column annotation: species
annotation_col <- data.frame(
    row.names = colnames(z_pdata),
    Species = gsub("_.*", "", colnames(z_pdata))
)

## Row annotation: duplication mode
annotation_row <- data.frame(
    row.names = rownames(z_pdata),
    Duplication = gsub(".*_", "", rownames(z_pdata)),
    Motif = c(
        rep("PPI V", 2), rep("V", 2), rep("Delta", 2), rep("Lambda", 2),
        rep("Bifan", 2)
    )
)

## Annotation colors
annotation_colors <- list(
    Duplication = dup_palette,
    Species = c(
        Athaliana = "#374E55FF", Gmax = "#DF8F44FF", Osativa = "#00A1D5FF",
        Ptrichocarpa = "#B24745FF", Slycopersicum = "#79AF97FF",
        Vvinifera = "#6A6599FF", Zmays = "#80796BFF"
    ),
    Motif = c(
        Bifan = "#E64B35FF", Delta = "#4DBBD5FF", Lambda = "#00A087FF",
        `PPI V` = "#3C5488FF", V = "#F39B7FFF"
    )
)

p_heatmap <- ComplexHeatmap::pheatmap(
    z_pdata, name = "Z-score",
    main = "Z-scores of motif frequencies for each species",
    cluster_rows = FALSE, cluster_cols = FALSE,
    annotation_col = annotation_col,
    annotation_row = annotation_row[, "Duplication", drop = FALSE],
    labels_col = c(
        "Peak 1", "Peak 2", # Osativa
        "Peak 1", "Peak 2", "Peak 3", # Zmays
        "Peak 1", # Vvinifera
        "Peak 1", "Peak 2", # Gmax
        "Peak 1", "Peak 2", # Slycopersicum
        "Peak 1", "Peak 2", "Peak 3", # Ptrichocarpa
        "Peak 1", "Peak 2"
    ),
    labels_row = c(
        rep(c("WGD", "SSD"), 5)
    ),
```

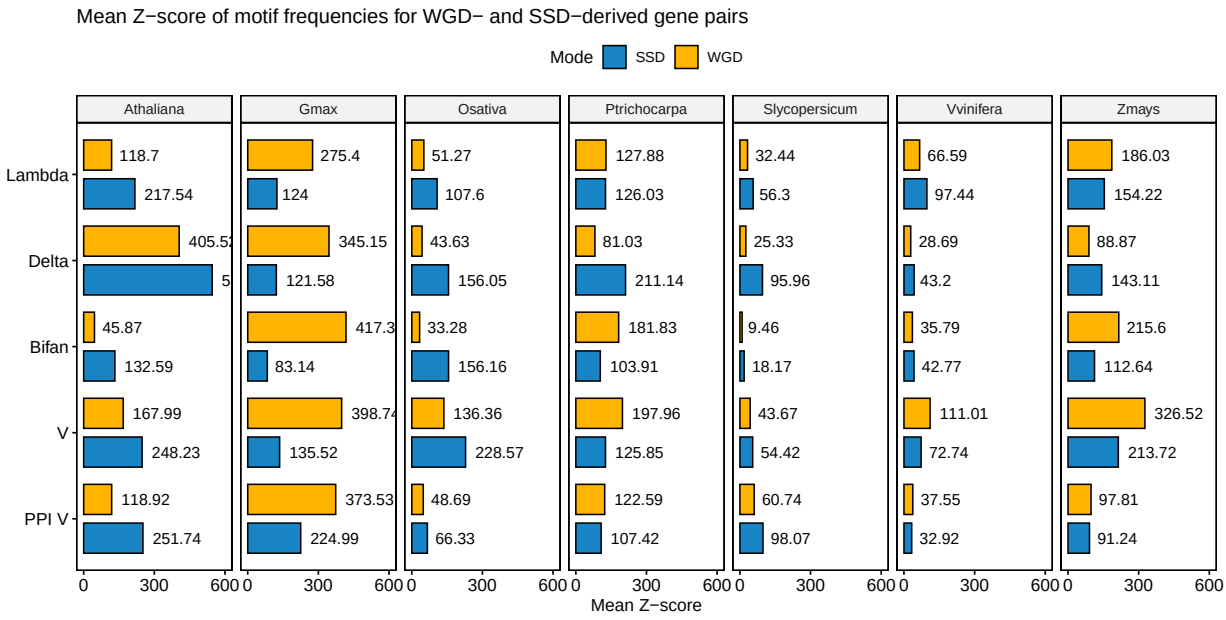
```
    annotation_colors = annotation_colors,
    col = colorRampPalette(c("#F1F8FA", "#0788B2"))(50),
    column_split = annotation_col$Species,
    row_split = annotation_row$Motif,
    row_gap = unit(3, "mm"),
    column_gap = unit(3, "mm"),
    display_numbers = TRUE
)
p_heatmap
```



Now, let's explore differences in motif counts for WGD- and SSD-derived gene pairs, ignoring peaks. For that, we will get the mean of all peaks.

```
# Get mean of all peaks for each species
zscores_df$Z[is.nan(zscores_df$Z)] <- 0
mean_z <- zscores_df %>%
    group_by(Species, Mode, Motif) %>%
    summarise(average_Z = round(mean(Z), 2)) %>%
    mutate(Motif = str_replace_all(
        Motif,
        c("bifan" = "Bifan",
          "delta" = "Delta",
          "lambda" = "Lambda",
          "PPI_V" = "PPI V")
    ))

library(ggpubr)
mean_z_plot <- ggbarplot(
    mean_z, x = "Motif", y = "average_Z",
    fill = "Mode", color = "black", palette = dup_palette,
    label = TRUE, position = position_dodge(0.9),
    facet.by = "Species", ncol = 7,
    lab.vjust = 0.5, lab.hjust = -0.2
) +
```

```
    coord_flip() +
    labs(
        title = "Mean Z-score of motif frequencies for WGD- and SSD-derived gene pairs",
        y = "Mean Z-score", x = ""
    ) +
    scale_y_continuous(limits = c(0, 600), breaks = seq(0, 600, 300))

mean_z_plot
```




```
# Save plots
p_heatmap_z_scores <- p_heatmap
save(
    p_heatmap_z_scores,
    file = here("products", "plots", "p_heatmap_z_scores.rda"),
    compress = "xz"
)


p_mean_z_scores <- mean_z_plot
save(
    p_mean_z_scores,
    file = here("products", "plots", "p_mean_z_scores.rda"),
    compress = "xz"
)
```

The figures shows some interesting patterns:

- For all motifs, genes derived from recent whole-genome duplications form more motifs than genes derived from more ancient whole-genome duplications. This suggests that motifs tend to be lost over time, probably due to fractionation.
- Overall, species with recent whole-genome duplications have a higher frequency of motifs than species with more ancient WGDs. This pattern is emphasized when comparing Z-scores for motif frequencies in *Z. mays* and *O. sativa*. The rice genome has signatures of a WGD that happened in the common ancestor of all Poaceae species, while the maize genome has undergone an additional round of WGD

more recently (~25 MYA). The frequencies of all motif types is dramatically higher in maize as compared to rice, demonstrating the significance of recent whole-genome duplications to the rewiring of GRNs.

- WGD-derived gene pairs have a greater contribution to motif formation in species with recent WGD events. When WGD events are more ancient, SSD has a greater contribution to the formation of motifs.

Calculating interaction similarity between paralogs

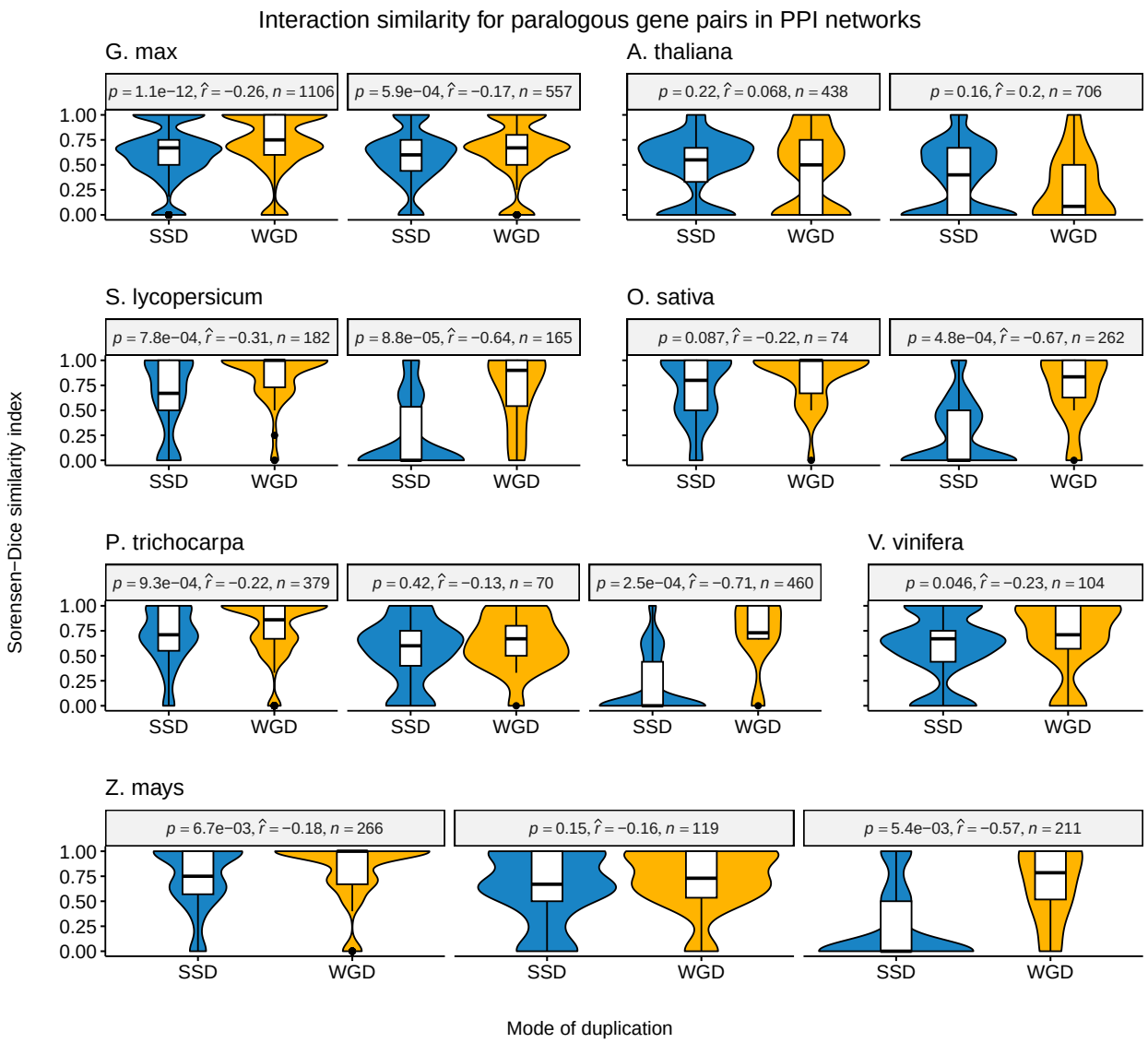
Now, for each species and peak, we will compare the interaction similarity between duplicated targets and duplicated TFs derived from WGD and SSD. Here, interaction similarity will be measured using the Sorensen-Dice similarity index, as implemented in `BiocStyle::Biocpkg("magrene")`.

First, we will calculate interaction similarity for the PPI network.

```
species <- names(motif_counts)

# Interaction similarity in PPI network
ppi_sorensendice <- lapply(species, function(x) {

    message("Working on species ", x)
    pairs <- paralogs[[x]][, c(1, 2, 4, 5)]
    names(pairs) <- c("dup1", "dup2", "mode", "peak")

    pairs_by_peak <- split(pairs, pairs$peak)
    int_by_peak <- Reduce(rbind, lapply(pairs_by_peak, function(y) {
        wgd <- y[y$mode == "WGD", ]
        ssd <- y[y$mode == "SSD", ]

        # Calculate interaction similarity
        sim_wgd <- sd_similarity(ppi[[x]], wgd[, 1:2])
        sim_wgd$mode <- "WGD"

        sim_ssd <- sd_similarity(ppi[[x]], ssd[, 1:2])
        sim_ssd$mode <- "SSD"

        sim <- rbind(sim_ssd, sim_wgd)

        # Add peak info
        sim$peak <- wgd$peak[1]
        return(sim)
    }))
    return(int_by_peak)
})
names(ppi_sorensendice) <- species

# Define function to plot distribution of Sorensen-Dice scores with stats
plot_sd <- function(sd_table = NULL) {

    # Get label with statistical results (test name, P, effect size, N)
    p_statistics <- grouped_ggbetweenstats(
        data = sd_table,
        x = mode,
        y = sorensen_dice,
        grouping.var = peak,
```

```
        type = "nonparametric",
        p.adjust.method = "BH"
    )
    statistics <- lapply(seq_along(unique(sd_table$peak)), function(x) {
        stats_df <- extract_stats(p_statistics[[x]])

        pval <- signif(stats_df$subtitle_data$p.value, 2)
        if(pval < 0.0099) {
            pval <- formatC(
                pval, format = "e", digits = 1
            )
        }
        effsize <- signif(stats_df$subtitle_data$estimate, 2)
        n <- stats_df$subtitle_data$n.obs

        stats <- glue::glue(
            "list(
                italic(p) == '{pval}',
                widehat(italic('r')) == '{effsize}',
                italic(n) == {n}
            )"
        )

        return(stats)
    })

    # Add a column with peak ID and statistics to add in facet
    sd_table$label <- factor(
        sd_table$peak,
        levels = seq_along(unique(sd_table$peak)),
        labels = statistics
    )

    p <- ggpubr::ggviolin(
        data = sd_table, x = "mode", y = "sorensen_dice",
        trim = TRUE,
        fill = "mode", palette = dup_palette,
        add = "boxplot", add.params = list(fill = "white")
    ) +
        facet_wrap(
            vars(label),
            nrow = 1, labeller = ggplot2::label_parsed
        ) +
        labs(y = "", x = "") +
        theme(legend.position = "none")

    return(p)
}

# Create plots for each species
sd_ppi_gma <- plot_sd(ppi_sorensendice$Gmax) + ggtitle("G. max")
sd_ppi_ath <- plot_sd(ppi_sorensendice$Ath) + ggtitle("A. thaliana")
sd_ppi_ptr <- plot_sd(ppi_sorensendice$Ptrichocarpa) + ggtitle("P. trichocarpa")
```

```r
sd_ppi_vvi <- plot_sd(ppi_sorensendice$Vvinifera) + ggtitle("V. vinifera")
sd_ppi_sly <- plot_sd(ppi_sorensendice$Slycopersicum) + ggtitle("S. lycopersicum")
sd_ppi_osa <- plot_sd(ppi_sorensendice$Osativa) + ggtitle("O. sativa")
sd_ppi_zma <- plot_sd(ppi_sorensendice$Zmays) + ggtitle("Z. mays")


# Combine plots into one
sd_ppi_allplots <- ggarrange(
    ggarrange(
        sd_ppi_gma, sd_ppi_ath + theme(axis.text.y = element_blank()),
        nrow = 1
    ),
    ggarrange(
        sd_ppi_sly, sd_ppi_osa + theme(axis.text.y = element_blank()),
        nrow = 1
    ),
    ggarrange(
        sd_ppi_ptr, sd_ppi_vvi + theme(axis.text.y = element_blank()),
        nrow = 1, widths = c(3, 1.2)
    ),
    sd_ppi_zma,
    ncol = 1
)

sd_ppi_allplots_final <- annotate_figure(
    sd_ppi_allplots,
    top = text_grob("Interaction similarity for paralogous gene pairs in PPI networks",
                    size = 15),
    bottom = text_grob("Mode of duplication"),
    left = text_grob("Sorensen-Dice similarity index", rot = 90)
)

sd_ppi_allplots_final
```




```
# Saving plot
p_sorensendice_ppi <- sd_ppi_allplots_final

save(
    p_sorensendice_ppi,
    file = here("products", "plots", "p_sorensendice_ppi.rda"),
    compress = "xz"
)
```

Overall, the genes WGD-derived gene pairs have a higher interaction similarity (i.e., they interact with the same targets) as compared to SSD-derived gene pairs. The difference is even more pronounced for older gene pairs, with gene pairs derived from ancient WGD displaying a much higher interaction similarity than gene pairs derived from ancient SSD (e.g., *S. lycopersicum*, *O. sativa*, *P. trichocarpa*, and *Z. mays*). This pattern suggests that strong selection pressures constrain WGD-derived pairs to maintain their shared interactions, while relaxed selection pressures for SSD-derived gene pairs allow them to diverge in interaction partners.

Now, let's do the same for the GRN. The difference is that, for the GRN, we will calculate Sorensen-Dice similarity indices separately for duplicated TFs and duplicated targets.

```r
species <- names(motif_counts)

# Interaction similarity in PPI network
grn_sorensendice <- lapply(species, function(x) {

    message("Working on species ", x)
    pairs <- paralogs[[x]][, c(1, 2, 4, 5)]
    names(pairs) <- c("dup1", "dup2", "mode", "peak")

    pairs_by_peak <- split(pairs, pairs$peak)
    int_by_peak <- Reduce(rbind, lapply(pairs_by_peak, function(y) {
        wgd <- y[y$mode == "WGD", ]
        ssd <- y[y$mode == "SSD", ]

        # Separate TF paralogs from target paralogs
        tfs <- unique(grns[[x]]$Node1)
        targets <- unique(grns[[x]]$Node2)

        ## WGD
        wgd_tf <- wgd[wgd$dup1 %in% tfs & wgd$dup2 %in% tfs, 1:2]
        wgd_target <- wgd[wgd$dup1 %in% targets & wgd$dup2 %in% targets, 1:2]

        ## SSD
        ssd_tf <- ssd[ssd$dup1 %in% tfs & ssd$dup2 %in% tfs, 1:2]
        ssd_target <- ssd[ssd$dup1 %in% targets & ssd$dup2 %in% targets, 1:2]

        # Calculate interaction similarity
        ## WGD
        sim_wgd_tf <- sd_similarity(grns[[x]], wgd_tf) %>%
            dplyr::mutate(node = "TF", mode = "WGD")
        sim_wgd_target <- sd_similarity(grns[[x]], wgd_target) %>%
            dplyr::mutate(node = "target", mode = "WGD")

        ## SSD
        sim_ssd_tf <- sd_similarity(grns[[x]], ssd_tf) %>%
            dplyr::mutate(node = "TF", mode = "SSD")
        sim_ssd_target <- sd_similarity(grns[[x]], ssd_target) %>%
            dplyr::mutate(node = "target", mode = "SSD")

        # Combine data frames into one
        sim <- rbind(sim_ssd_tf, sim_ssd_target, sim_wgd_tf, sim_wgd_target)

        # Add peak info
        sim$peak <- wgd$peak[1]
        return(sim)
    }))
    return(int_by_peak)
})
names(grn_sorensendice) <- species
```

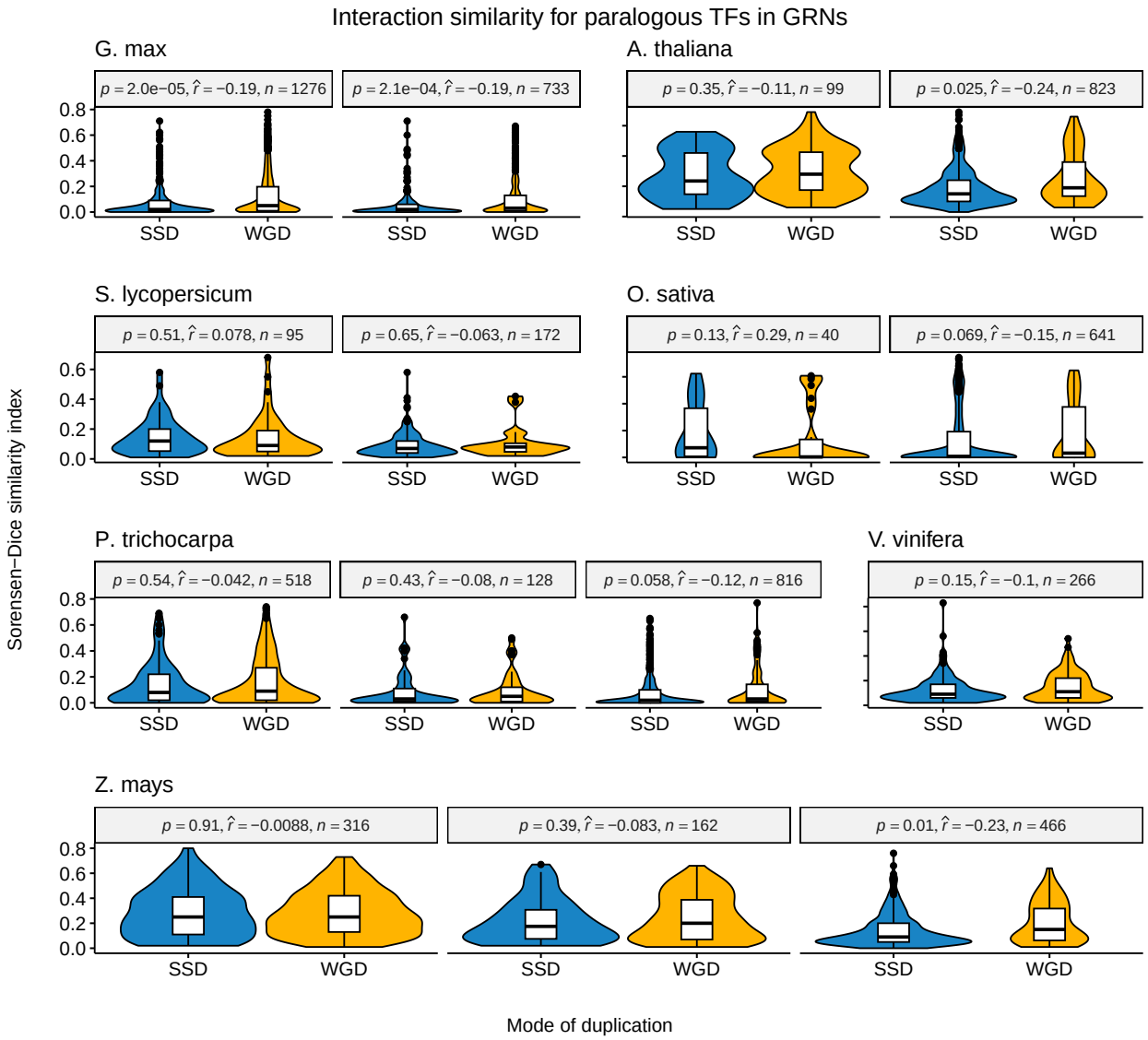
```r
# Save results in an object
save(
    grn_sorensendice,
    file = here("products", "result_files", "grn_sorensendice.rda"),
```

```
    compress = "xz"
)
```

Let's visualize the distributions of Sorensen-Dice indices for duplicated TFs.

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

# Plot Sorensen-Dice indices for duplicated TFs
sd_grn_gma_tf <- grn_sorensendice$Gmax %>% filter(node == "TF") %>%
    plot_sd() + ggtitle("G. max")

sd_grn_ath_tf <- grn_sorensendice$Athaliana %>% filter(node == "TF") %>%
    plot_sd() + ggtitle("A. thaliana")

sd_grn_ptr_tf <- grn_sorensendice$Ptrichocarpa %>% filter(node == "TF") %>%
    plot_sd() + ggtitle("P. trichocarpa")

sd_grn_vvi_tf <- grn_sorensendice$Vvinifera %>% filter(node == "TF") %>%
    plot_sd() + ggtitle("V. vinifera")

sd_grn_sly_tf <- grn_sorensendice$Slycopersicum %>% filter(node == "TF") %>%
    plot_sd() + ggtitle("S. lycopersicum")

sd_grn_osa_tf <- grn_sorensendice$Osativa %>% filter(node == "TF") %>%
    plot_sd() + ggtitle("O. sativa")

sd_grn_zma_tf <- grn_sorensendice$Zmays %>% filter(node == "TF") %>%
    plot_sd() + ggtitle("Z. mays")

## Combine plots into one
sd_grn_tfs <- ggarrange(
    ggarrange(
        sd_grn_gma_tf, sd_grn_ath_tf + theme(axis.text.y = element_blank()),
        nrow = 1
    ),
    ggarrange(
        sd_grn_sly_tf, sd_grn_osa_tf + theme(axis.text.y = element_blank()),
        nrow = 1
    ),
    ggarrange(
        sd_grn_ptr_tf, sd_grn_vvi_tf + theme(axis.text.y = element_blank()),
        nrow = 1, widths = c(3, 1.2)
    ),
    ggarrange(sd_grn_zma_tf, nrow = 1),
    ncol = 1
)
sd_grns_tfs_final <- annotate_figure(
    sd_grn_tfs,
    top = text_grob("Interaction similarity for paralogous TFs in GRNs",
                    size = 15),
    bottom = text_grob("Mode of duplication"),
    left = text_grob("Sorensen-Dice similarity index", rot = 90)
)
```

```
sd_grns_tfs_final
```

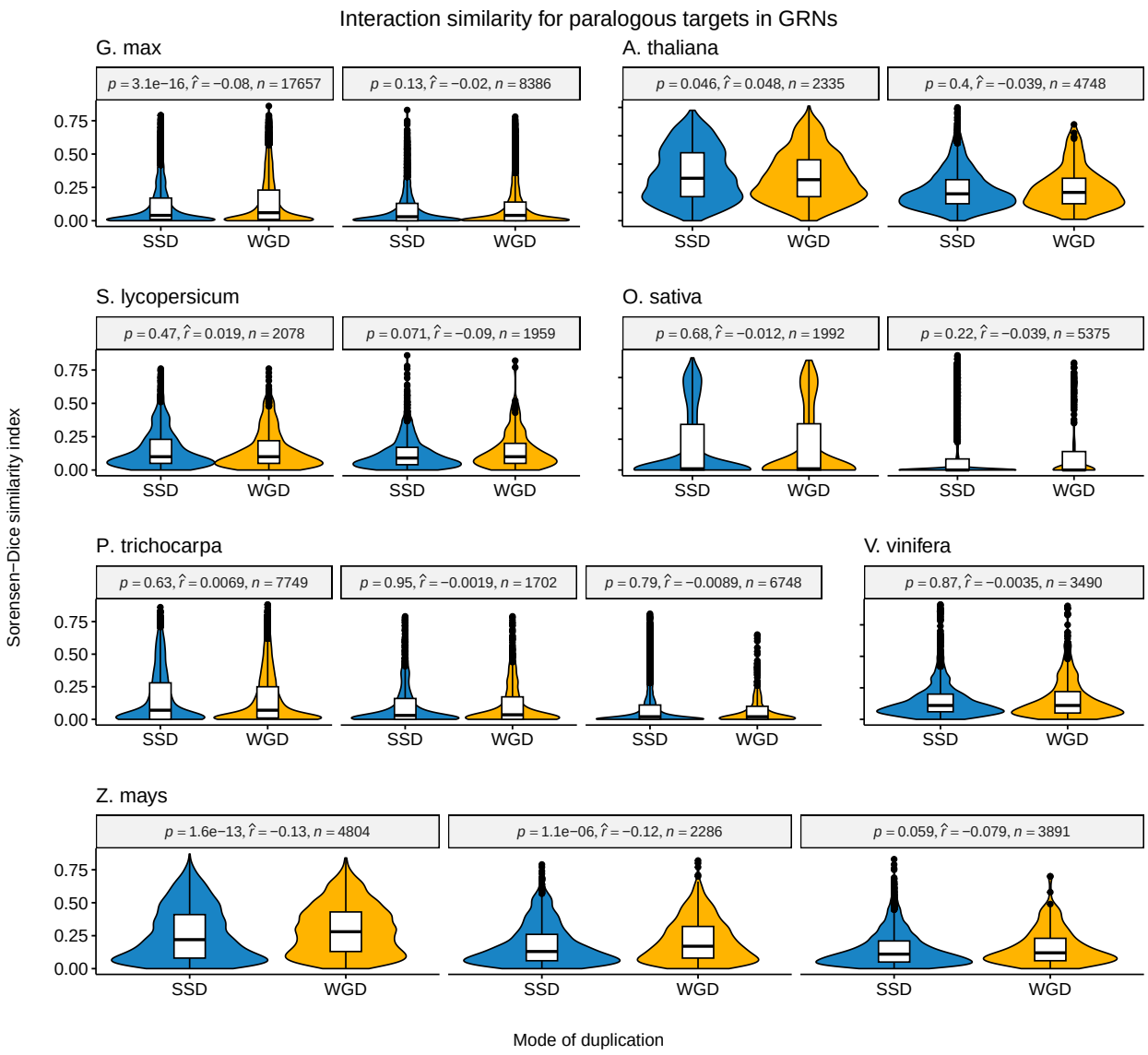


Overall, there is no difference in interaction similarity between TFs duplicated by WGD and SSD in the GRNs. Although there are significant differences for some comparisons (e.g., *G. max*, 2nd peak for *A. thaliana*, and 3rd peak for *Z. mays*), the effect sizes are small, so we can't conclude whether the differences are due to a biological signal or some kind of artifact.

Now, let's explore if the pattern is the same for duplicated targets.

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

# Plot Sorensen-Dice indices for duplicated TFs
sd_grn_gma_target <- grn_sorensendice$Gmax %>% filter(node == "target") %>%
    plot_sd() + ggtitle("G. max")

sd_grn_ath_target <- grn_sorensendice$Athaliana %>% filter(node == "target") %>%
    plot_sd() + ggtitle("A. thaliana")

sd_grn_ptr_target <- grn_sorensendice$Ptrichocarpa %>% filter(node == "target") %>%
```

```
    plot_sd() + ggtitle("P. trichocarpa")

sd_grn_vvi_target <- grn_sorensendice$Vvinifera %>% filter(node == "target") %>%
    plot_sd() + ggtitle("V. vinifera")

sd_grn_sly_target <- grn_sorensendice$Slycopersicum %>% filter(node == "target") %>%
    plot_sd() + ggtitle("S. lycopersicum")

sd_grn_osa_target <- grn_sorensendice$Osativa %>% filter(node == "target") %>%
    plot_sd() + ggtitle("O. sativa")

sd_grn_zma_target <- grn_sorensendice$Zmays %>% filter(node == "target") %>%
    plot_sd() + ggtitle("Z. mays")

## Combine plots into one
sd_grn_targets <- ggarrange(
    ggarrange(
        sd_grn_gma_target, sd_grn_ath_target + theme(axis.text.y = element_blank()),
        nrow = 1
    ),
    ggarrange(
        sd_grn_sly_target, sd_grn_osa_target + theme(axis.text.y = element_blank()),
        nrow = 1
    ),
    ggarrange(
        sd_grn_ptr_target, sd_grn_vvi_target + theme(axis.text.y = element_blank()),
        nrow = 1, widths = c(3, 1.2)
    ),
    ggarrange(sd_grn_zma_target, nrow = 1),
    ncol = 1
)

sd_grns_targets_final <- annotate_figure(
    sd_grn_targets,
    top = text_grob("Interaction similarity for paralogous targets in GRNs",
                    size = 15),
    bottom = text_grob("Mode of duplication"),
    left = text_grob("Sorensen-Dice similarity index", rot = 90)
)

sd_grns_targets_final
```



Indeed, for duplicated targets, we observe the same pattern. Although some differences are significant ($P < 0.05$), the effect sizes are small, so we cannot draw any conclusion from it.

```
# Saving the plots
p_sorensendice_grn_tfs <- sd_grns_tfs_final
save(
    p_sorensendice_grn_tfs,
    file = here("products", "plots", "p_sorensendice_grn_tfs.rda"),
    compress = "xz"
)

p_sorensendice_grn_targets <- sd_grns_targets_final
save(
    p_sorensendice_grn_targets,
    file = here("products", "plots", "p_sorensendice_grn_targets.rda"),
    compress = "xz"
)
```

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     2023-05-03
##  pandoc   2.19.2 @ /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)
##  backports          1.4.1   2021-12-13 [1] CRAN (R 4.3.0)
##  bayestestR         0.13.1  2023-04-07 [1] CRAN (R 4.3.0)
##  BiocGenerics       0.46.0  2023-04-25 [1] Bioconductor
##  BiocParallel     * 1.34.0  2023-04-25 [1] Bioconductor
##  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)
##  circlize           0.4.15  2022-05-10 [1] CRAN (R 4.3.0)
##  cli                3.6.1   2023-03-23 [1] CRAN (R 4.3.0)
##  clue               0.3-64  2023-01-31 [1] CRAN (R 4.3.0)
##  cluster            2.1.4   2022-08-22 [4] CRAN (R 4.2.1)
##  codetools          0.2-19  2023-02-01 [4] CRAN (R 4.2.2)
##  colorspace         2.1-0   2023-01-23 [1] CRAN (R 4.3.0)
##  ComplexHeatmap   * 2.16.0  2023-04-25 [1] Bioconductor
##  correlation        0.8.4   2023-04-06 [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)
##  datawizard         0.7.1   2023-04-03 [1] CRAN (R 4.3.0)
##  digest             0.6.31  2022-12-11 [1] CRAN (R 4.3.0)
##  doParallel         1.0.17  2022-02-07 [1] CRAN (R 4.3.0)
##  dplyr            * 1.1.2   2023-04-20 [1] CRAN (R 4.3.0)
##  effectsize         0.8.3   2023-01-28 [1] CRAN (R 4.3.0)
##  evaluate           0.20    2023-01-17 [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)
##  foreach            1.5.2   2022-02-02 [1] CRAN (R 4.3.0)
##  generics           0.1.3   2022-07-05 [1] CRAN (R 4.3.0)
##  GetoptLong         1.0.5   2020-12-15 [1] CRAN (R 4.3.0)
##  ggplot2          * 3.4.2   2023-04-03 [1] CRAN (R 4.3.0)
##  ggpubr           * 0.6.0   2023-02-10 [1] CRAN (R 4.3.0)
##  ggrepel            0.9.3   2023-02-03 [1] CRAN (R 4.3.0)
```

```
##  ggsignif           0.6.4   2022-10-13 [1] CRAN (R 4.3.0)
##  ggstatsplot      * 0.11.1  2023-04-14 [1] CRAN (R 4.3.0)
##  GlobalOptions      0.1.2   2020-06-10 [1] CRAN (R 4.3.0)
##  glue               1.6.2   2022-02-24 [1] CRAN (R 4.3.0)
##  gridExtra          2.3     2017-09-09 [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)
##  highr              0.10    2022-12-22 [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)
##  insight            0.19.1  2023-03-18 [1] CRAN (R 4.3.0)
##  IRanges            2.34.0  2023-04-25 [1] Bioconductor
##  iterators          1.0.14  2022-02-05 [1] CRAN (R 4.3.0)
##  knitr              1.42    2023-01-25 [1] CRAN (R 4.3.0)
##  labeling           0.4.2   2020-10-20 [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)
##  magrene          * 1.2.0   2023-04-25 [1] Bioconductor
##  magrittr           2.0.3   2022-03-30 [1] CRAN (R 4.3.0)
##  matrixStats        0.63.0  2022-11-18 [1] CRAN (R 4.3.0)
##  munsell            0.5.0   2018-06-12 [1] CRAN (R 4.3.0)
##  paletteer          1.5.0   2022-10-19 [1] CRAN (R 4.3.0)
##  parameters         0.21.0  2023-04-19 [1] CRAN (R 4.3.0)
##  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)
##  png                0.1-8   2022-11-29 [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)
##  readr            * 2.1.4   2023-02-10 [1] CRAN (R 4.3.0)
##  rematch2           2.1.2   2020-05-01 [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          2.21    2023-03-26 [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
##  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)
##  shape              1.4.6   2021-05-19 [1] CRAN (R 4.3.0)
##  statsExpressions   1.5.0   2023-02-19 [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)
##  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)
##  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              0.6.2   2023-04-19 [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)
##  yaml               2.3.7   2023-01-23 [1] CRAN (R 4.3.0)
##  zeallot            0.1.0   2018-01-28 [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
##
## ------------------------------------------------------------------------------
```