AGO binding sites definition

Mirko Brüggemann, Melina Klostermann

21 April, 2023

Contents

1 Libraries and settings

```
# ----------------------------------------
# libraries
# ----------------------------------------

library(rtracklayer)
library(GenomicRanges)
library(ggplot2)
library(AnnotationDbi)
library(dplyr)
library(reshape2)
library(UpSetR)
library(GenomicFeatures)
library(kableExtra)
library(knitr)
library(ggrepel)
library(gridExtra)
library(grid)
library(viridis)
library(BindingSiteFinder)
library(ComplexHeatmap)
library(forcats)
library(ggtext)
library(patchwork)
library(tibble)
library(tidyr)
library(dplyr)
library(ggpointdensity)
library(ggsci)
library(ggrepel)

source("/Users/melinaklostermann/Documents/projects/AgoCLIP_miR181/mirko_files/mirECLIP/DifferentialBin
source("/Users/melinaklostermann/Documents/projects/R_general_functions/theme_paper.R")

# ----------------------------------------
# settings
# ----------------------------------------

out <- "/Users/melinaklostermann/Documents/projects/AgoCLIP_miR181/R_github/miR181_paper/Figure1/AGO_bi

# farben
farbeneg <- "#B4B4B4"
farbe1 <- "#0073C2FF" #WT farbe
farbe2 <- "#EFC000FF"
farbe3 <- "#CD534CFF" #miR181KO farbe
farbe4 <- "#7AA6DCFF"
farbe5 <- "#868686FF"
farbe6 <- "#003C67FF"
farbe7 <- "#8F7700FF"
farbe8 <- "#3B3B3BFF"
farbe9 <- "#A73030FF"
farbe10 <- "#4A6990FF"
farbe11 <- "#FF6F00FF"
farbe12 <- "#C71000FF"
farbe13 <- "#008EA0FF"
farbe14 <- "#8A4198FF"
farbe15 <- "#5A9599FF"
farbe16 <- "#FF6348FF"
```

2 What was done?

- Here we define AGO binding sites on basis of the non-chimeric non-enriched miR-eCLIP data.
- Peaks are called with pureclip on the .bam merge of all three replicates.
- Then binding sites are defined with BindingSiteFinder.
- Binding sites are filtered for their reproducibility in at least 2 of 3 samples.
- Then bindingsites are matched to the bound gene and the bound gene region.

NOTE: Large parts of code and text are from the BindingSiteFinder Vignette. For a detailed explanation see https://www.bioconductor.org/packages/release/bioc/vignettes/BindingSiteFinder/inst/doc/vignette.html

3 Files

3.1 Merge bam files run pureclip

We use the peakcaller pureclip to detect crosslink peaks. pureclip is run on the merge of the non-chimeric crosslinks of all three samples.

```
# ------------------------------------
# Run pureclip
# ----------------------------------------

pureclip \
-i crosslinks_all_samples.sort.bam\
-bai crosslinks_all_samples.sort.bam.bai \
-g GRCm38.p6.genome.strict.IUPAC.fa \
-nt 10 \
-o IP_WT_pureclip_sites.bed \
```

```
# ----------------------------------------
# ----------------------------------------
# Files
# ----------------------------------------
# ----------------------------------------

# ----------------------------------------
# annotation
# ----------------------------------------

annoDb <- loadDb("/Users/melinaklostermann/Documents/projects/AgoCLIP_miR181/R_github/miR181_paper/Meth
gns <- readRDS("/Users/melinaklostermann/Documents/projects/AgoCLIP_miR181/R_github/miR181_paper/Method

# ----------------------------------------
# pureclip sites (from peak calling)
# ----------------------------------------
pureclip_sites <- "/Users/melinaklostermann/Documents/projects/AgoCLIP_miR181/pureclip/IP_WT_pureclip_s
pureclip_sites  = import(con = pureclip_sites , format = "BED", extraCols=c("additionalScores" = "chara
pureclip_sites  = keepStandardChromosomes(pureclip_sites , pruning.mode = "coarse")

# ----------------------------------------
# Load clip data
# ----------------------------------------

clipFiles = "/Users/melinaklostermann/Documents/projects/AgoCLIP_miR181/pipe_output_22_02_14/non-chimer
clipFiles = list.files(clipFiles, pattern = ".bw$", full.names = TRUE)
clipFiles = clipFiles[!grepl("Inp", clipFiles)]
clipFiles = clipFiles[!grepl("miR181_", clipFiles)]
clipFiles = clipFiles[grepl("WT", clipFiles)]
clipFilesP = clipFiles[grep(clipFiles, pattern = "plus")]
clipFilesM = clipFiles[grep(clipFiles, pattern = "minus")]
```

4 Functions

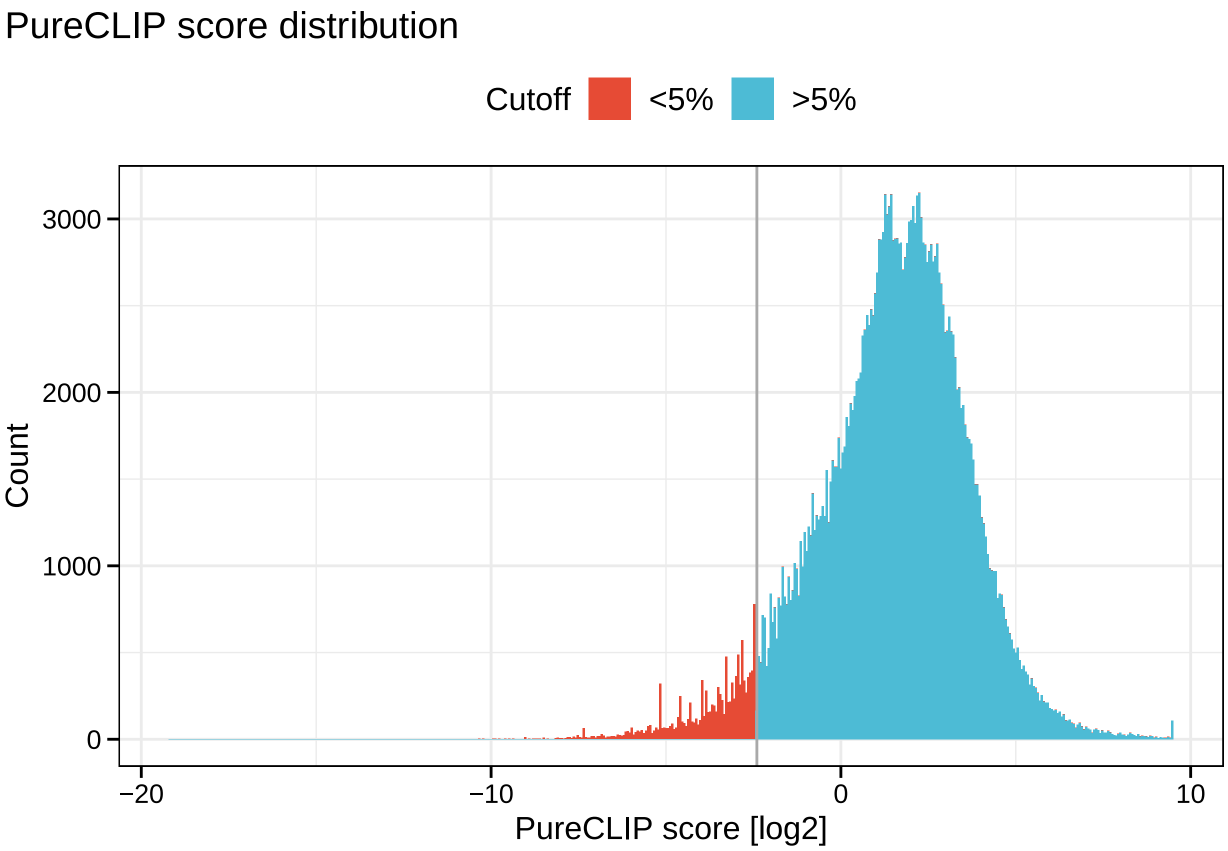
```
# ----------------------------------------
# utility functions
# ----------------------------------------

basicVectorToNiceDf <- function(x){
  # NOTE MK you could do all starting from the 3. line in one mutate command, might be faster and easie
  df = data.frame(Type = names(table(x$olType)), Freq = as.vector(table(x$olType)))
  df = df[order(df$Freq, decreasing = F),]
  df$Type = factor(df$Type, levels = df$Type)
  df$Frac = df$Freq / sum(df$Freq)
  df$ymax = cumsum(df$Frac)
  df$ymin = c(0, head(df$ymax, n=-1))
  df$labPos = (df$ymax + df$ymin) / 2
  df$NFrac = round(df$Frac * 100)
  df$NFrac2 = round(df$Freq / sum(df$Freq), digits = 4)
  df$NFracNice = df$NFrac2 * 100
  df$labelNice = paste0(format(df$Freq, big.mark = ",", decimal.mark = "."), " (", df$NFracNice, "%)")
  df$labelNice2 = paste0(df$Type, ": ", format(df$Freq, big.mark = ",", decimal.mark = "."), " (", df$N
  return(df)
}
```

5 Preprocess pureCLIP output

Initial peak calling was performed with PureCLIP (see Melinas reports), which results in single nucleotide wide crosslink sites. These sites are pre-filtered before merging them into wider binding sites.

5.1 Global filter

```
# ----------------------------------------
# Filter peaks global by pureCLIP score
# ----------------------------------------

# Global 5% cutoff

df = data.frame(score = pureclip_sites $score)
quantileCutoffpureClipScore = quantile(df$score, probs = seq(0,1, by = 0.05))
peaksFiltered = pureclip_sites [pureclip_sites $score >= quantileCutoffpureClipScore[2]]


df$group = ifelse(df$score > quantileCutoffpureClipScore[2], ">5%", "<5%")
ggplot(df, aes(x = log2(score), fill = group)) +
    geom_histogram(bins = 500) +
    geom_vline(xintercept = log2(quantileCutoffpureClipScore[2]), color = "darkgrey") +
    labs(
        title = "PureCLIP score distribution",
        x = "PureCLIP score [log2]",
        y = "Count",
        fill = "Cutoff"
    ) +
    theme_nice() +
    theme(legend.position = "top") +
    scale_fill_npg()
```



PureCLIP score global filter. Distribution of the pureCLIP score. The red line indicates the 95% threshold used to keep only strong signal sites.

Here, PureCLIP called crosslink sites are filtered on a global level, removing sites with the lowest 5% PureCLIP score. This essentially removes crosslink sites that are only barely enriched above the local background. These sites rather contribute more noise to the data rather then enhancing the spectrum of detected sites to lowly abundant transcripts, since they are present uniformly across almost all transcripts.

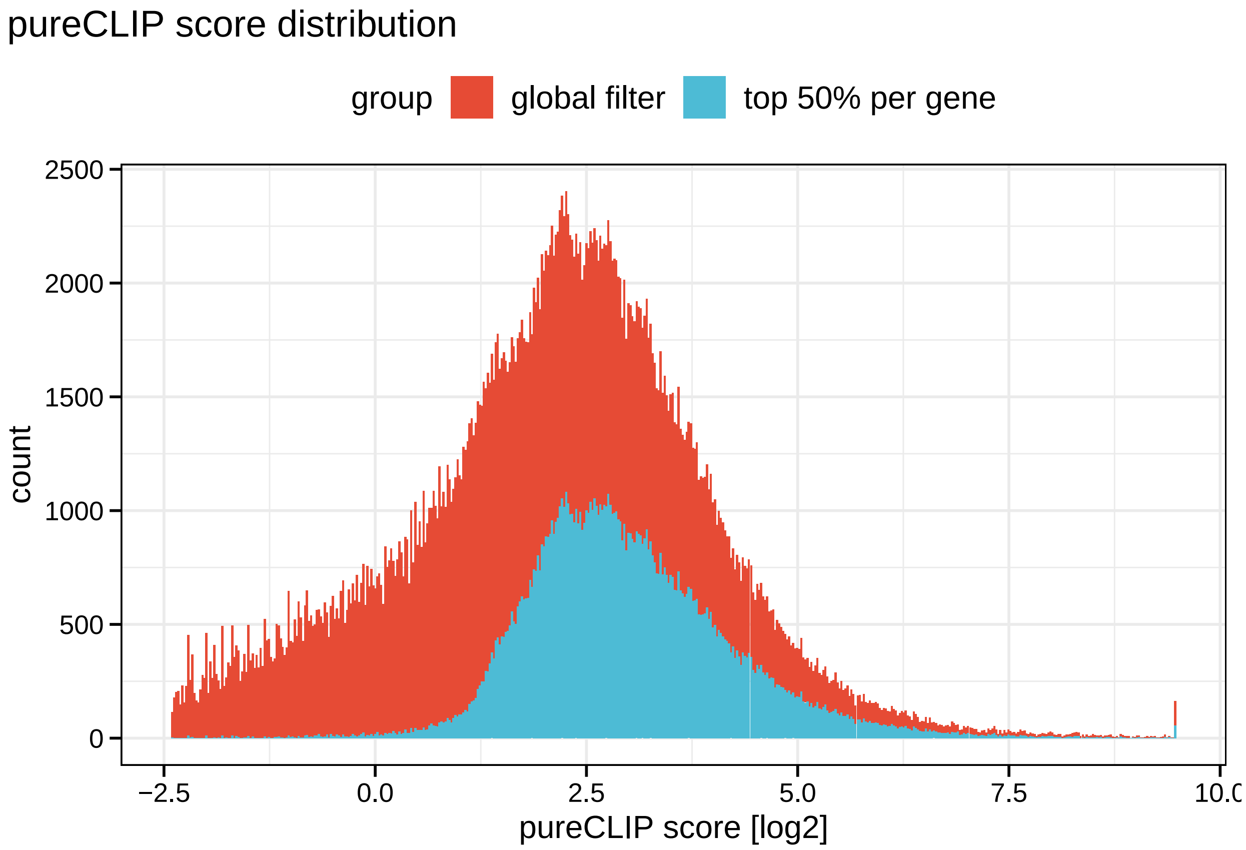
5.2 Gene level filter

To enrich for the most prominent crosslink pattern one can restrict the binding site definition to the top 50% of PureCLIP sites per gene. This type of filter can be very restrictive, potentially removing a large proportion of the data. It is beneficial for detecting the strongest binding pattern presented in the data. It should be removed for the detection of finer more subtle binding pattern, or to allow for a high vs. low comparison scheme.

5.2.1 Final gene level settings

Here we select for the top 50% highest PureCLIP site for each gene.

```
# ----------------------------------------
# Filter peaks genewise by pureCLIP score
# ----------------------------------------

filterByRegion <- function(region, peaks, keepAbove) {
  # remove peaks not located on selected regions
  ols = findOverlaps(region, peaks)
  queries = unique(queryHits(ols))
  # apply selected cutoff to every region
  filteredPerRegion = sapply(queries, function(x){
    currentHits = subjectHits(ols)[queryHits(ols) == x]
    currentPeaks = peaks[currentHits]
    currentQuantiles = quantile(currentPeaks$score, probs = seq(0,1, by = 0.01))
    # NOTE MK actually you could use probs = keepAbove, and then score >= currentQuantile, you do not u
    names(currentQuantiles) = seq(0,1, by = 0.01)
    filteredPeaks = currentPeaks[currentPeaks$score >= currentQuantiles[names(currentQuantiles) == keepA
    filteredPeaks
  })


  filteredPerRegion = unlist(GRangesList(filteredPerRegion))
  # Remove peaks on multiple regions
  filteredPerRegion = filteredPerRegion[countOverlaps(filteredPerRegion) == 1]
  # ----
  # Explanation:
  # It can happen that a single peak overlaps multiple different genes. If that peaks
  # survives the filter for both genes the range of the peak would be duplicated in the output.
  # To prevent this from happening all duplicated peaks were removed to one unique representative
  # after the filtering.

  return(filteredPerRegion)
}

peaksFilteredPerGene = filterByRegion(gns, peaksFiltered, keepAbove = 0.5)


df1 = data.frame(score = peaksFiltered$score, group = "global filter")
df2 = data.frame(score = peaksFilteredPerGene$score, group = "top 50% per gene")
df = rbind(df1,df2)

ggplot(df, aes(x = log2(score), fill = group)) +
    geom_histogram(bins = 500) +
    xlab("pureCLIP score [log2]") +
    ylab("count") +
    ggtitle("pureCLIP score distribution") +
    theme_nice() +
    theme(legend.position = "top") +
    scale_fill_npg()
```



–> PureCLIP score gene level filter. Distribution of the pureCLIP score. Only the top 50% binding sites per gene are retained.

6 Compute binding sites

6.1 Merge and extend binding sites

```
# -------------------------------
# set BS final options
# -------------------------------
bsSize_Final = 7
minWidth_Final = 2
minCrosslinks_Final = 2
minClSites_Final = 2
```

Next binding sites are merged into binding sites of a fixed width. Choosing this width parameter is explained in the following plots.

```
# ---------------------------------------------------------
# Organize clip data in dataframe for binding site finder
# ---------------------------------------------------------

colData = data.frame(
  id = c(1:3),
  condition = factor(c("WT", "WT", "WT"),
                     levels = c("WT")),
                     clPlus = clipFilesP,
                     clMinus = clipFilesM)

# Make BindingSiteFinder object
bds = BSFDataSetFromBigWig(ranges = peaksFilteredPerGene, meta = colData)
bds
```

```
## Object of class BSFDataSet
## Contained ranges:  112.357
## ----> Number of chromosomes:  22
## ----> Ranges width:  1
## Contained Signal: 21,250,289
## Contained conditions:  WT
```

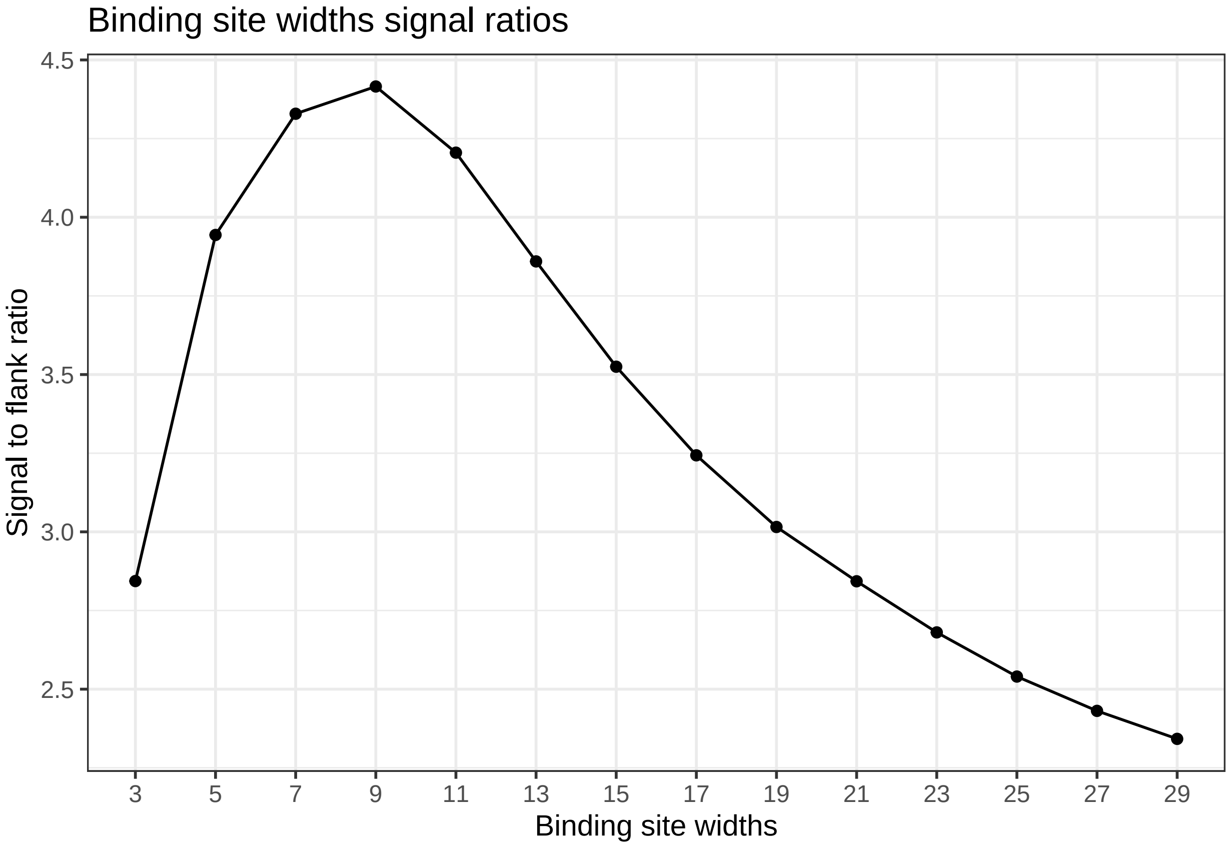
In total 3 samples from 1 condition(s) are used.

6.2 Controls for binding site width setting

We use the following plots to define the optimal binding site width for this experiment.

The optimal binding site width is determined by the binding site signal to noise ratio. For each binding site, the ratio of crosslink events within the binding site and of crosslink events flanking the binding site to both sides is determined.

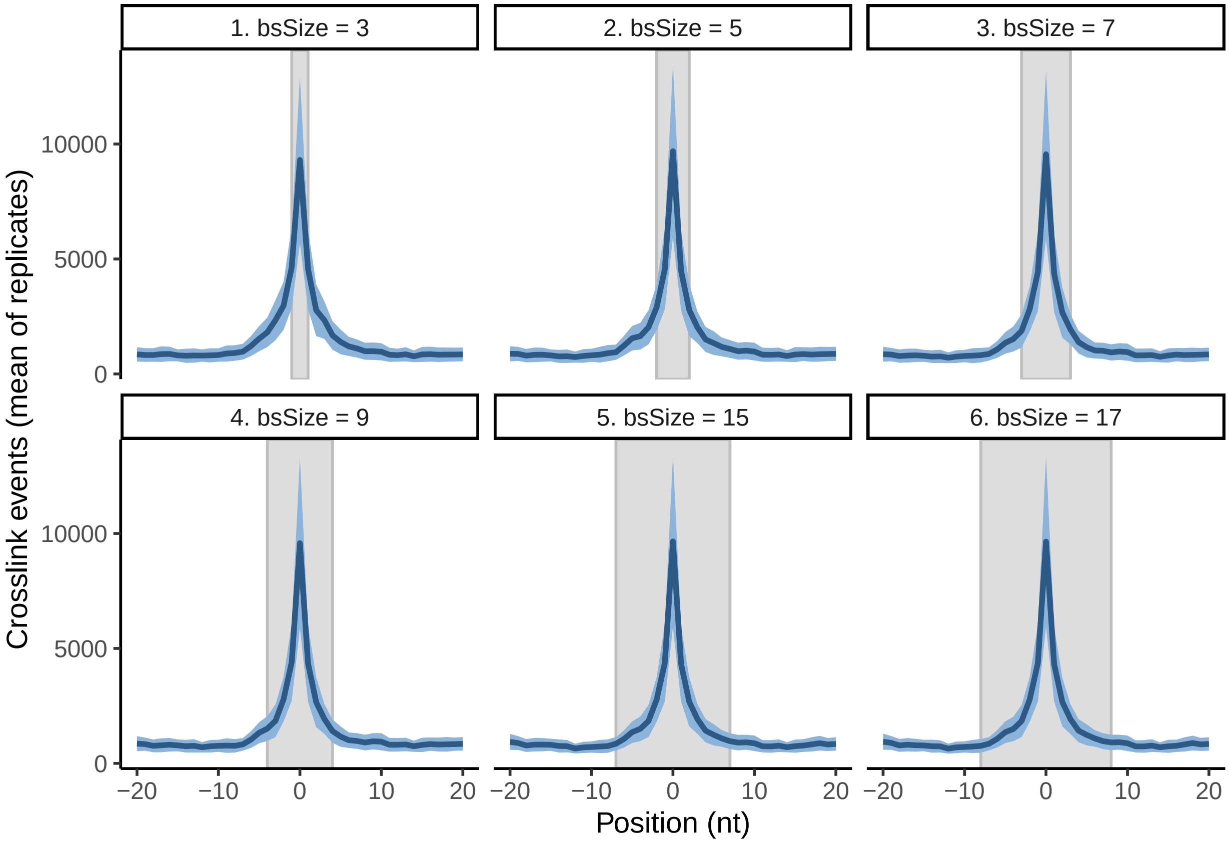
```
# ------------------------------------------------------
# SupportRatio plot
# ------------------------------------------------------
supportRatioPlot(bds, bsWidths = seq(from = 3, to = 29, by = 2), minWidth = 2, minClSites = 2, minCrossl
```



This plot shows that the optimal resolution of signal in binding sites would be for 9nt or 11nt binding sites.

For further visual inspection selected binding site width are plotted as count profiles centered around the binding sites central position.

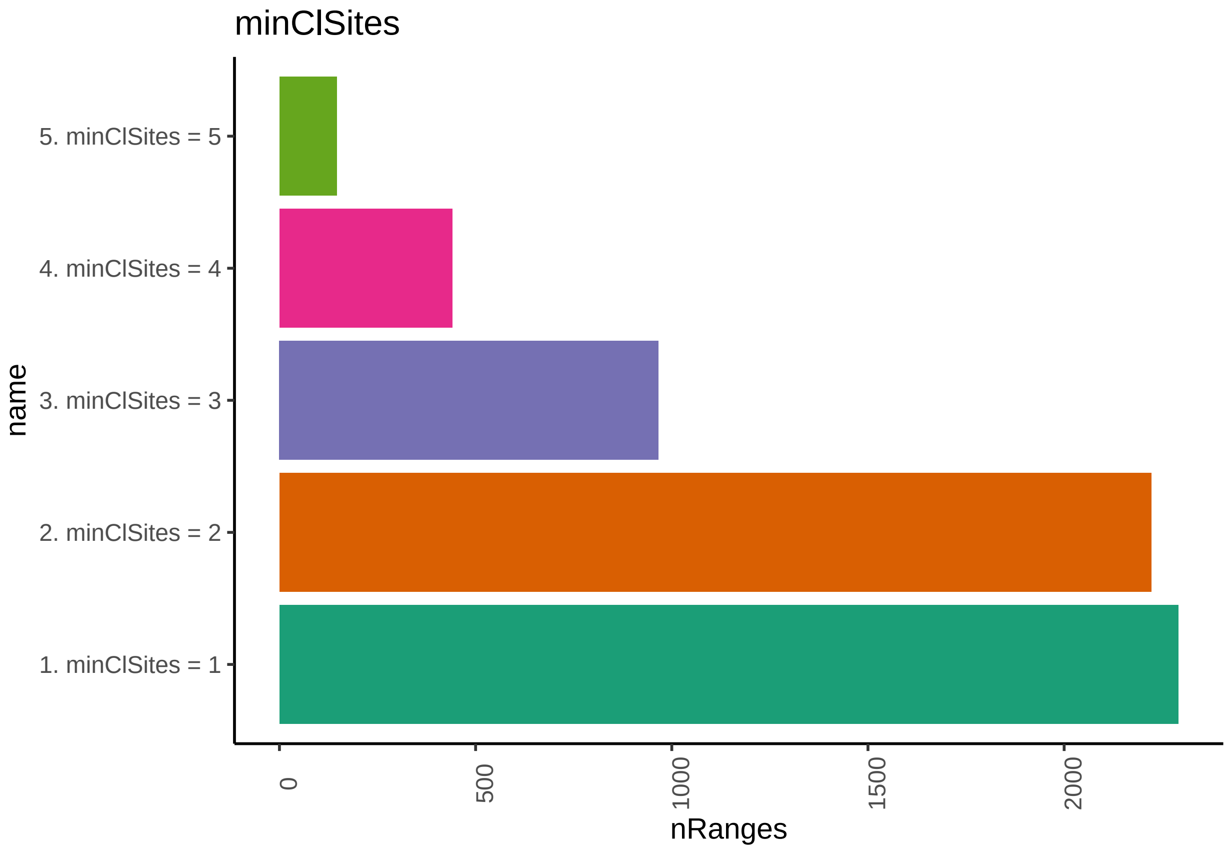
```r
# -------------------------------------------------------
# RangeCoverage plot - binding site width
# -------------------------------------------------------
bds1 <- makeBindingSites(object = bds, bsSize = 3, minWidth = 2,
                              minCrosslinks = 2, minClSites = 2, sub.chr = "chr1")

bds2 <- makeBindingSites(object = bds, bsSize = 5, minWidth = 2,
                              minCrosslinks = 2, minClSites = 2, sub.chr = "chr1")

bds3 <- makeBindingSites(object = bds, bsSize = 7, minWidth = 2,
                              minCrosslinks = 2, minClSites = 2, sub.chr = "chr1")

bds4 <- makeBindingSites(object = bds, bsSize = 9, minWidth = 2,
                              minCrosslinks = 2, minClSites = 2, sub.chr = "chr1")

bds5 <- makeBindingSites(object = bds, bsSize = 15, minWidth = 2,
                              minCrosslinks = 2, minClSites = 2, sub.chr = "chr1")

bds6 <- makeBindingSites(object = bds, bsSize = 17, minWidth = 2,
                              minCrosslinks = 2, minClSites = 2, sub.chr = "chr1")

bds7 <- makeBindingSites(object = bds, bsSize = 11, minWidth = 2,
                              minCrosslinks = 2, minClSites = 2, sub.chr = "chr1")

l = list(`1. bsSize = 3` = bds1, `2. bsSize = 5` = bds2, `3. bsSize = 7` = bds3,
         `4. bsSize = 9` = bds4, `5. bsSize = 15` = bds5, `6. bsSize = 17` = bds6)

rangeCoveragePlot(l, width = 20)
```



In these plot binding sites with width 7 look even better. We therefore used 7nt in the further steps.

Next, we test the effect of different filter stringecies within the binding site merging routine.

```
# ---------------------------------------------------------
# Check how the number of crosslinkSites in a binding site influences binding site definition
# ---------------------------------------------------------
bds1 <- makeBindingSites(object = bds, bsSize = 7, minWidth = 2,
                              minCrosslinks = 2, minClSites = 1, sub.chr = "chr1")

bds2 <- makeBindingSites(object = bds, bsSize = 7, minWidth = 2,
                              minCrosslinks = 2, minClSites = 2, sub.chr = "chr1")

bds3 <- makeBindingSites(object = bds, bsSize = 7, minWidth = 2,
                              minCrosslinks = 2, minClSites = 3, sub.chr = "chr1")

bds4 <- makeBindingSites(object = bds, bsSize = 7, minWidth = 2,
                              minCrosslinks = 2, minClSites = 4, sub.chr = "chr1")

bds5 <- makeBindingSites(object = bds, bsSize = 7, minWidth = 2,
                              minCrosslinks = 2, minClSites = 5, sub.chr = "chr1")

l = list(`1. minClSites = 1` = bds1, `2. minClSites = 2` = bds2, `3. minClSites = 3` = bds3,
         `4. minClSites = 4` = bds4, `5. minClSites = 5` = bds5)

mergeSummaryPlot(l, select = "minClSites") +
  coord_flip()
```



–> Profile of peaks of size 7, with different numbers of forced pureCLIP sites (minClSites). Binding site and signal are subsetted to chr1.

Lastly the final set of binding sites is computed with the following settings:

```
    object = bds
    bsSize_Final = 7
    minWidth_Final = 2
    minCrosslinks_Final = 2
    minClSites_Final = 2
```

- bsSize = 7
- minWidth = 2
- minCrosslinks = 2
- minClSites = 2

7 Make binding sites

```
# ---------------------------------------------------------
# Make  binding sites
# ---------------------------------------------------------
bds_sites <- makeBindingSites(object = bds, bsSize = bsSize_Final, minWidth = minWidth_Final,
                              minCrosslinks = minCrosslinks_Final, minClSites = minClSites_Final)

bds_sites_gr = getRanges(bds_sites)


df = getSummary(bds_sites) # NOTE MK nice function
```

Table 1: Merge and combine

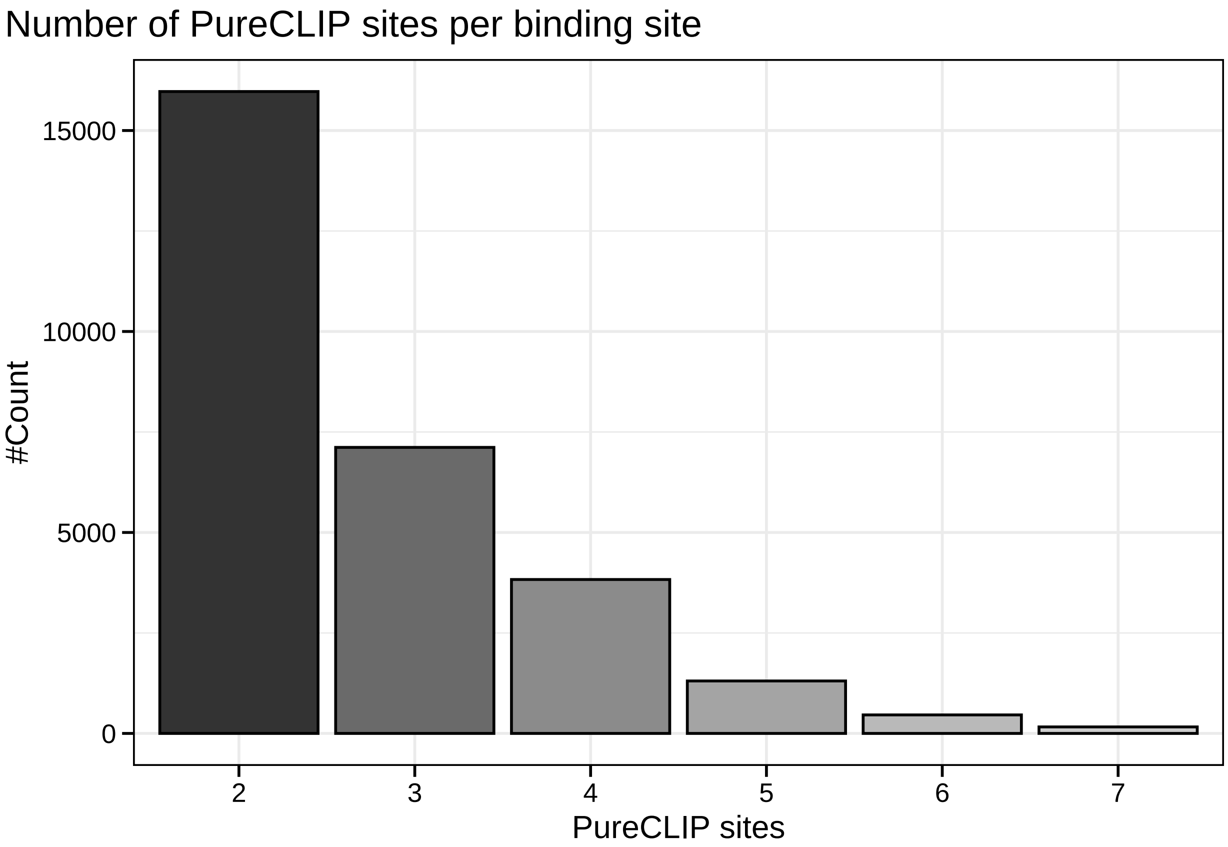
Option	nRanges
inputRanges	112,357
mergeCrosslinkSites	30,304
minCrosslinks	29,975
minClSites	29,133
centerIsClSite	29,112
centerIsSummit	28,843

```
kable(myNumberFormat(df), caption = "Merge and combine")
```

7.1 Binding site composition

Details about the binding site computation can be seen when counting the number of PureCLIP sites that make up a specific binding site.

```
# -------------------------------------------------------
# Check pureclip sites per binding site
# -------------------------------------------------------

df = countOverlaps(getRanges(bds_sites), peaksFilteredPerGene) %>%
    table() %>%
    as.data.frame() %>%
    rename("n" = ".")

ggplot(df, aes(x = n, y = Freq, fill = n)) +
    geom_col(color = "black") +
    scale_fill_grey() +
    theme_nice() +
    theme(legend.position = "none") +
    labs(
        title = "Number of PureCLIP sites per binding site",
        x = "PureCLIP sites",
        y = "#Count"
    )
```

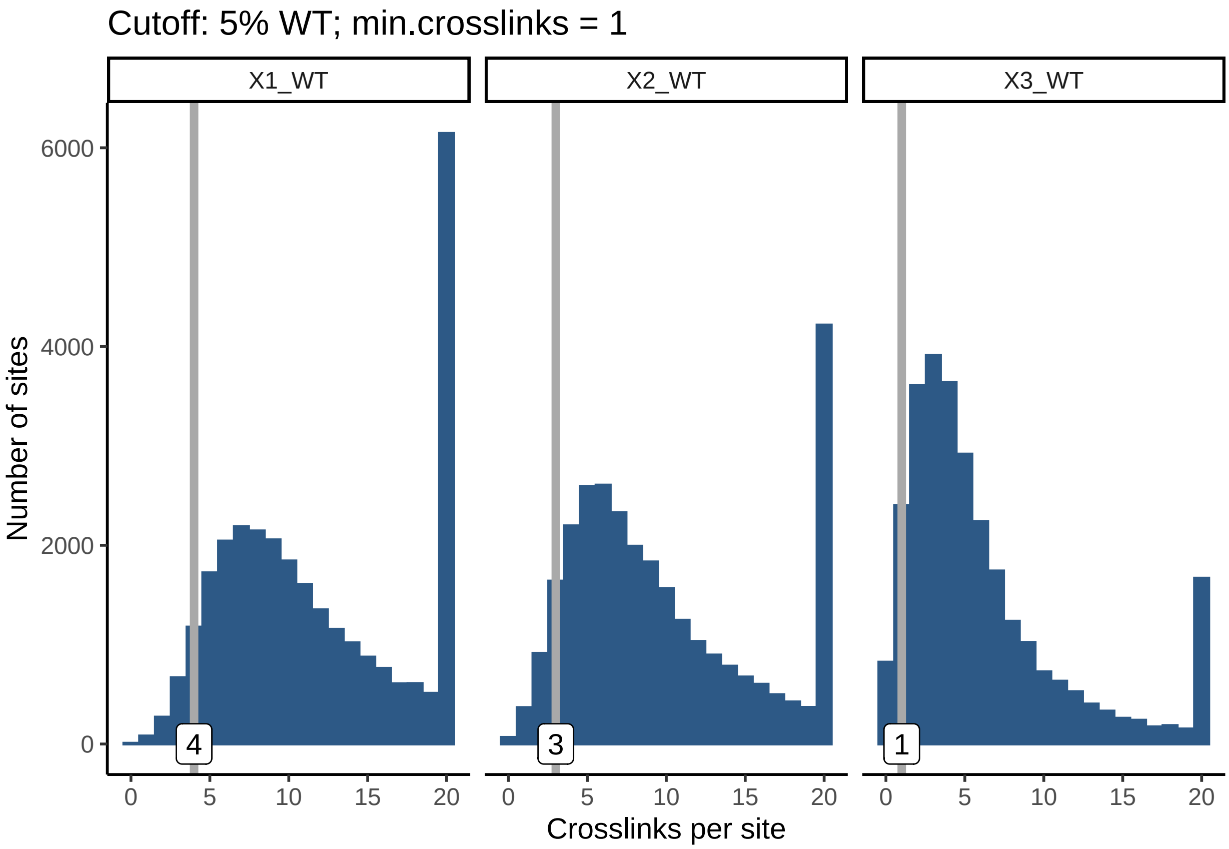


Number of PureCLIP sites per binding sites in the final binding site set.

8 Reproducibility Filter

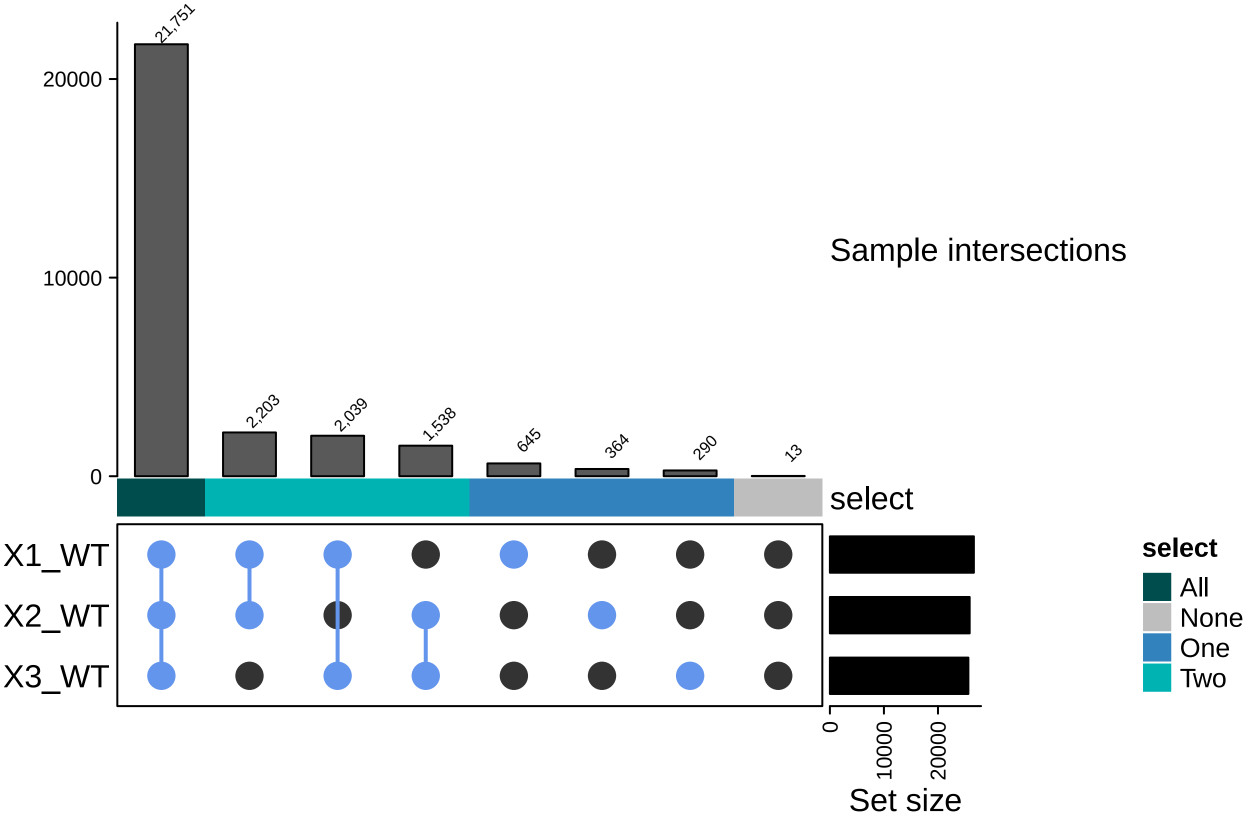
Since peak calling is based on the merge of all replicates, we filter processed binding site for their support by the individual replicates. First, crosslinks are summed up per replicate creating a crosslink distribution. Then, a threshold is set for each replicate to the 5% quantile of that distribution. To account for low crosslink replicates, a lower boundary of 1 crosslink events per binding site is enforced.

```r
# -------------------------------------------------------
# Check reproducibility cutoff
# -------------------------------------------------------

p <- reproducibiliyCutoffPlot(bds_sites, max.range = 20, cutoff = 0.05)

p
```



Finally, a binding sites is deemed reproducible if the thresholds are met for all three replicates. This is the most stringent filter possible.

```
# ------------------------------------------------------------
# Check reproducibility of binding sites between the three non-chimeric and the three chimeric samples
# ------------------------------------------------------------
s1 = reproducibilityFilter(bds_sites, cutoff = 0.05, n.reps = 2, returnType = "data.frame")
m = make_comb_mat(s1, mode = "distinct")

df = strsplit(names(comb_degree(m)), "")
df = do.call(rbind,df)
df = as.matrix(df)
df = apply(df, 2, as.numeric)
df = as.data.frame(df)

df$support = rowSums(df)
rownames(df) = names(comb_degree(m))


df$status = ifelse(df$support == 3, "All",
                   ifelse(df$support == 2, "Two",
                          ifelse(df$support == 1, "One", "None")))

# plot Upset with Complex Heatmap package
ha = HeatmapAnnotation(
  col = list(select = c("All" = "#004d4d", "Two" = "#00b3b3", "One" = "#3182BD", "None" = "grey")),
  "Sample intersections" = anno_barplot(comb_size(m), border = FALSE, gp = gpar(fill = "#595959"), heig
    select = df$status
)

ht = UpSet(m,
           set_order = colnames(s1),
           comb_order = order(comb_size(m), decreasing = T),
           top_annotation = ha,
           comb_col = "cornflowerblue", bg_col = "white", pt_size = unit(.5, "cm") ,
           border = T, lwd = 2, bg_pt_col = "#333333"
)

ss = set_size(m)
cs = comb_size(m)
ht = draw(ht,  padding = unit(c(0, 0, 10, 0), "mm"))
od = column_order(ht)
decorate_annotation("Sample intersections", {
    grid.text(format(cs[od], big.mark = ",", decimal.mark = "."), x = seq_along(cs), y = unit(cs[od], "
        default.units = "native", just = c("left", "bottom"),
        gp = gpar(fontsize = 6, col = "black"), rot = 45)
})
```



Reproducibility summary. Overview of binding sites that are shared between replicates. A binding site is reproducible if supported by all 3 replicates.

8.1 Final binding sites

This leaves us with a final set of binding sites:

seqnames	start	end	width	strand	scoreSum	scoreMean	scoreMax
chr1	4529031	4529037	7	+	20.37027	10.18514	13.34730
chr1	4529066	4529072	7	+	33.00360	16.50180	21.85370
chr1	6238329	6238335	7	+	36.87593	18.43796	32.46330
chr1	6244182	6244188	7	+	9.45328	4.72664	5.37285
chr1	6245324	6245330	7	+	123.40239	24.68048	38.21990
chr1	6245604	6245610	7	+	28.79086	14.39543	23.35330

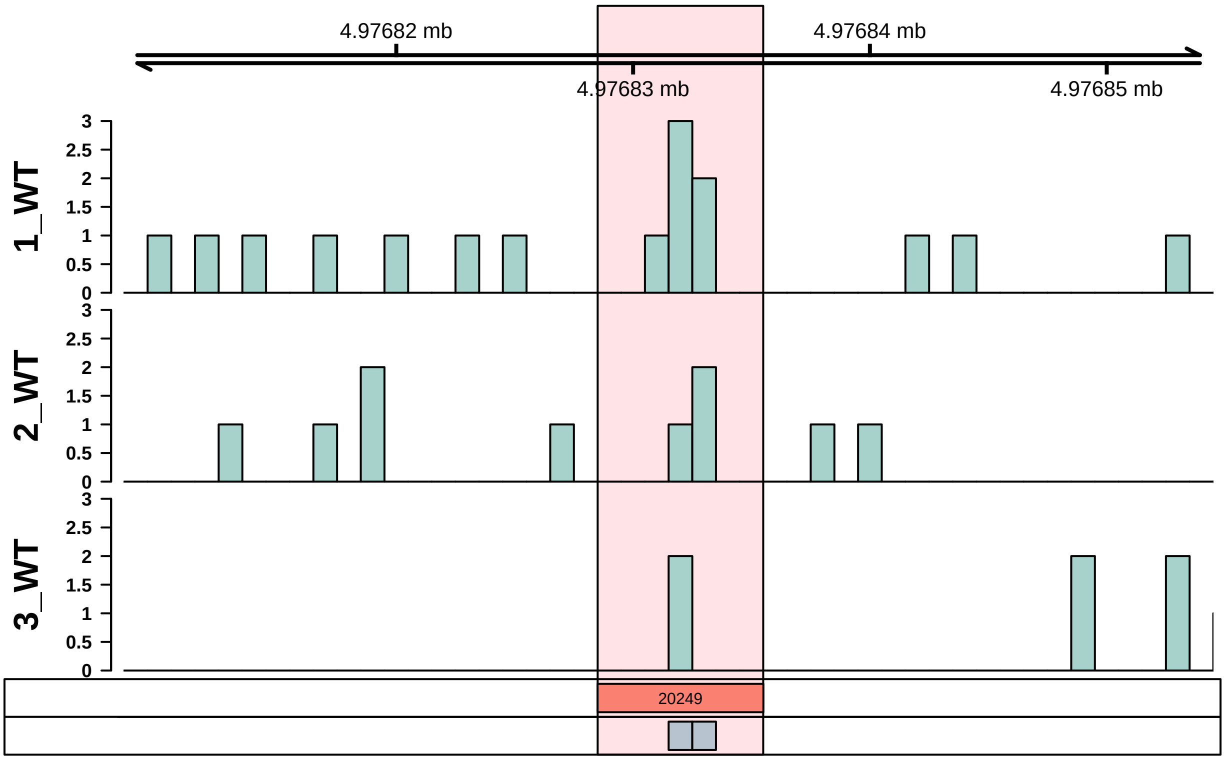
```
# ------------------------------------------------------------
# Get reproducible binding sites
# ------------------------------------------------------------
bdsFinal = reproducibilityFilter(bds_sites, cutoff = 0.05, n.reps = 2)
bdsFinal = annotateWithScore(bdsFinal, getRanges(bds))
bdsFinal_gr = getRanges(bdsFinal)

kable(head(bdsFinal_gr))  %>%
  kable_material(c("striped", "hover"))
```

We get 27531 reproducible binding sites.

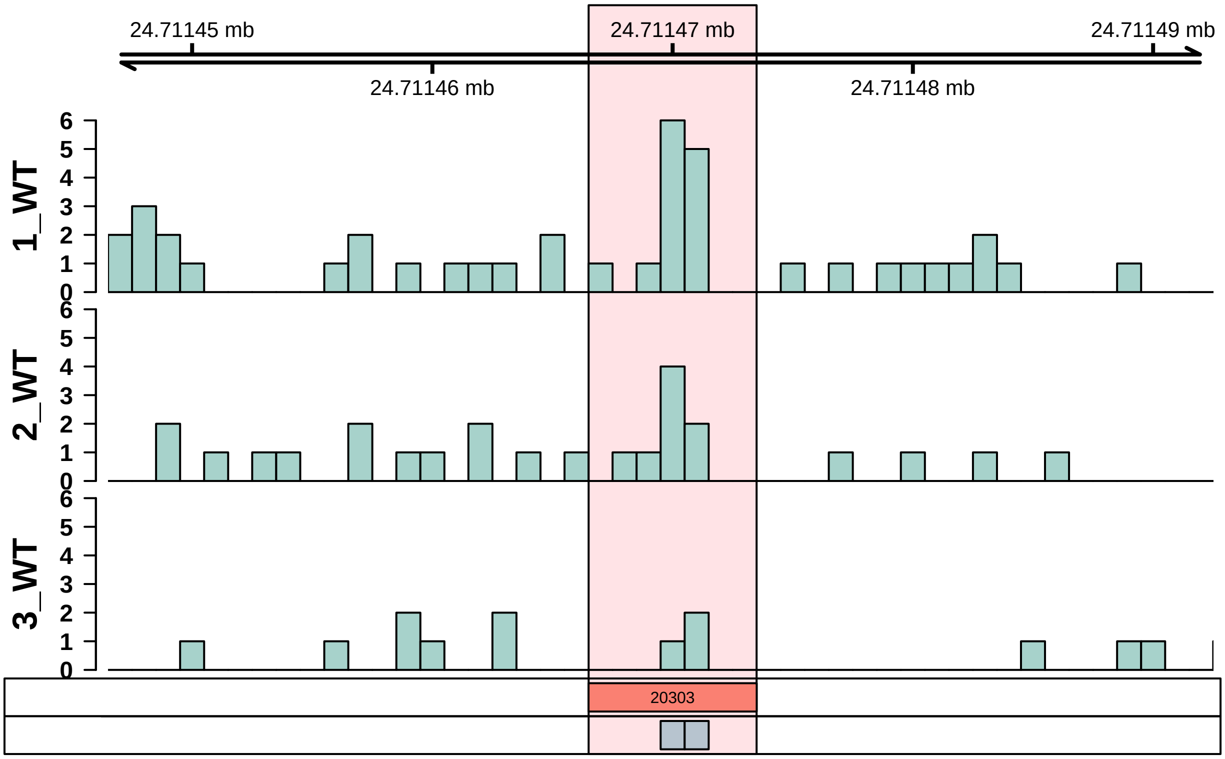
8.2 Examples

```
set.seed(123)

s <- sample(1:NROW(bdsFinal_gr), 8) %>% as.list(.)

lapply(s, function(x) bindingSiteCoveragePlot(bdsFinal,
                                              plotIdx = x,
                                              flankPos = 20,
                                              autoscale = TRUE,
                                              customRange = pureclip_sites,
                                              customRange.name = "pSites"))
```

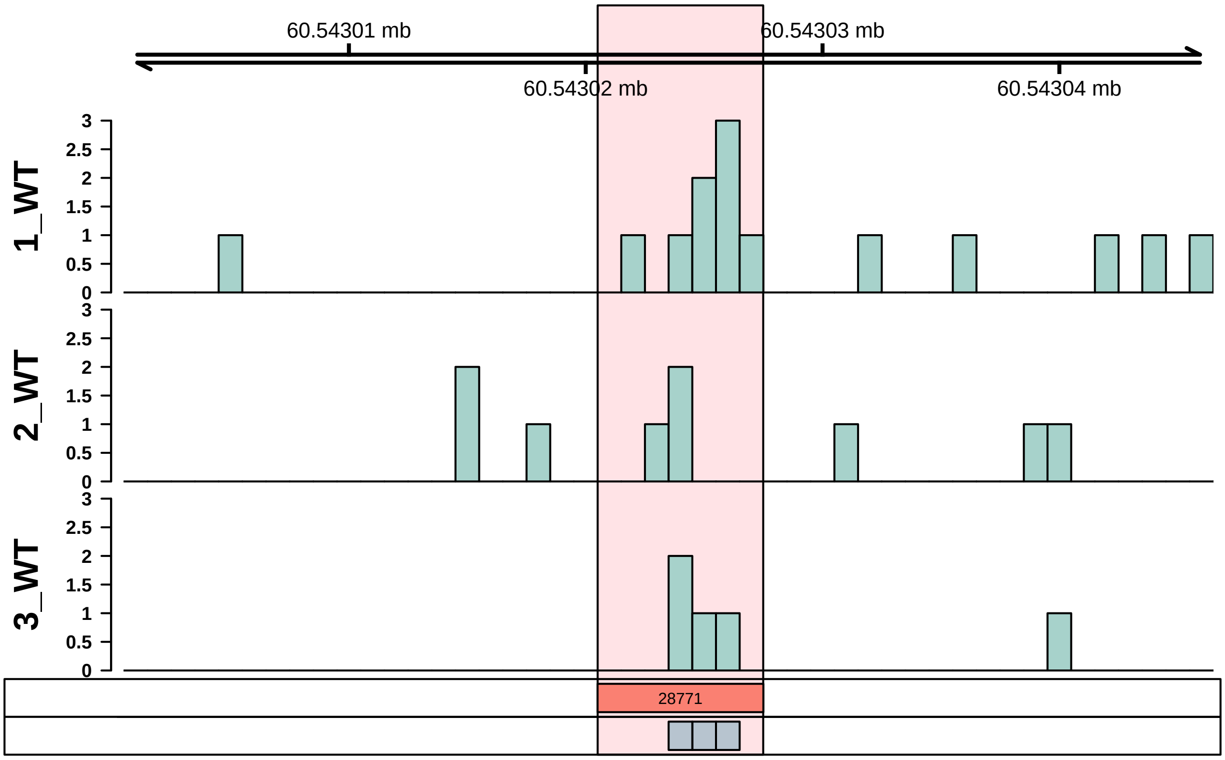
BS:20249/ chr12: 4976809−4976855 +



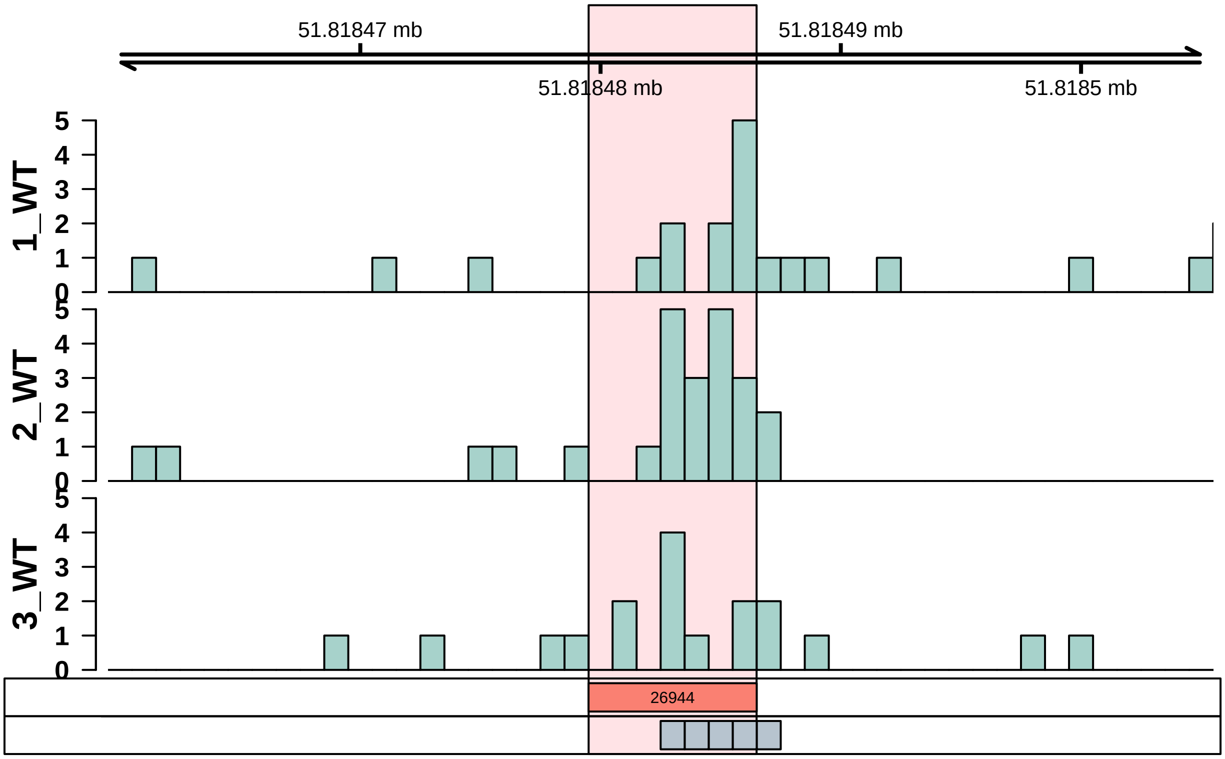
BS:20303/ chr12: 24711447−24711493 +



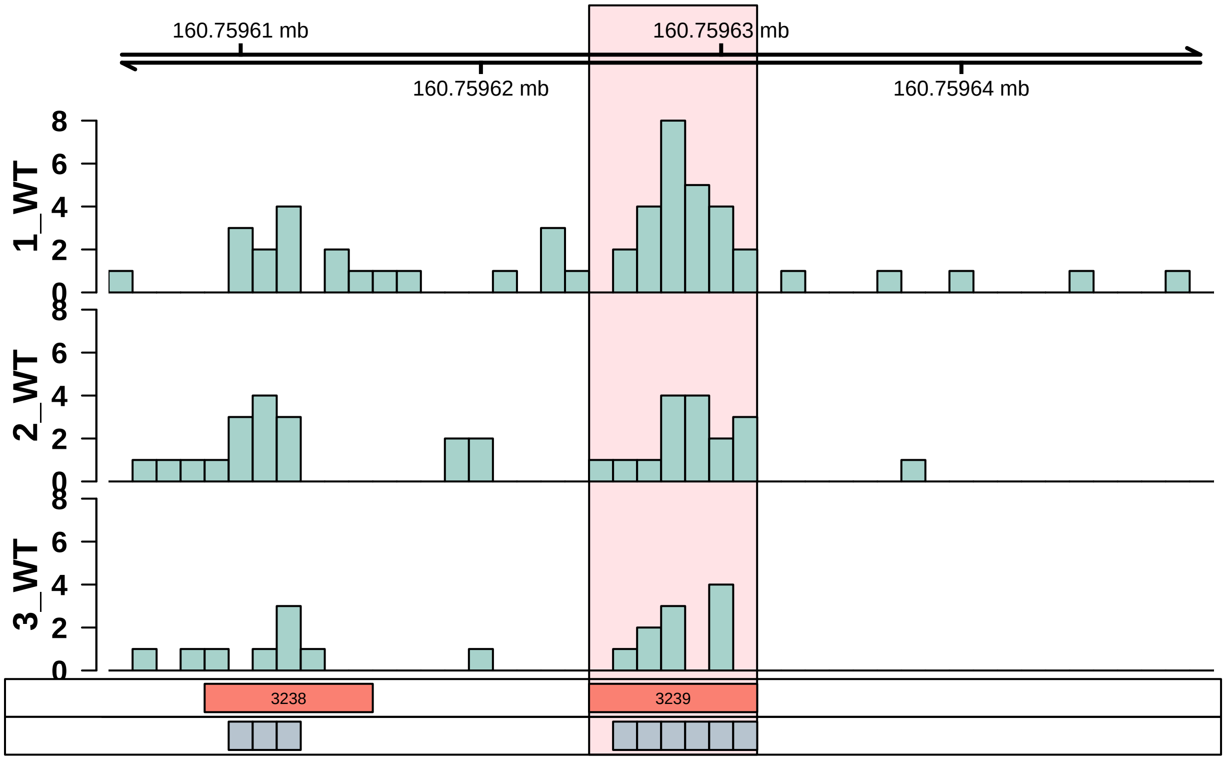
BS:28771/ chr19: 60543001−60543047 −



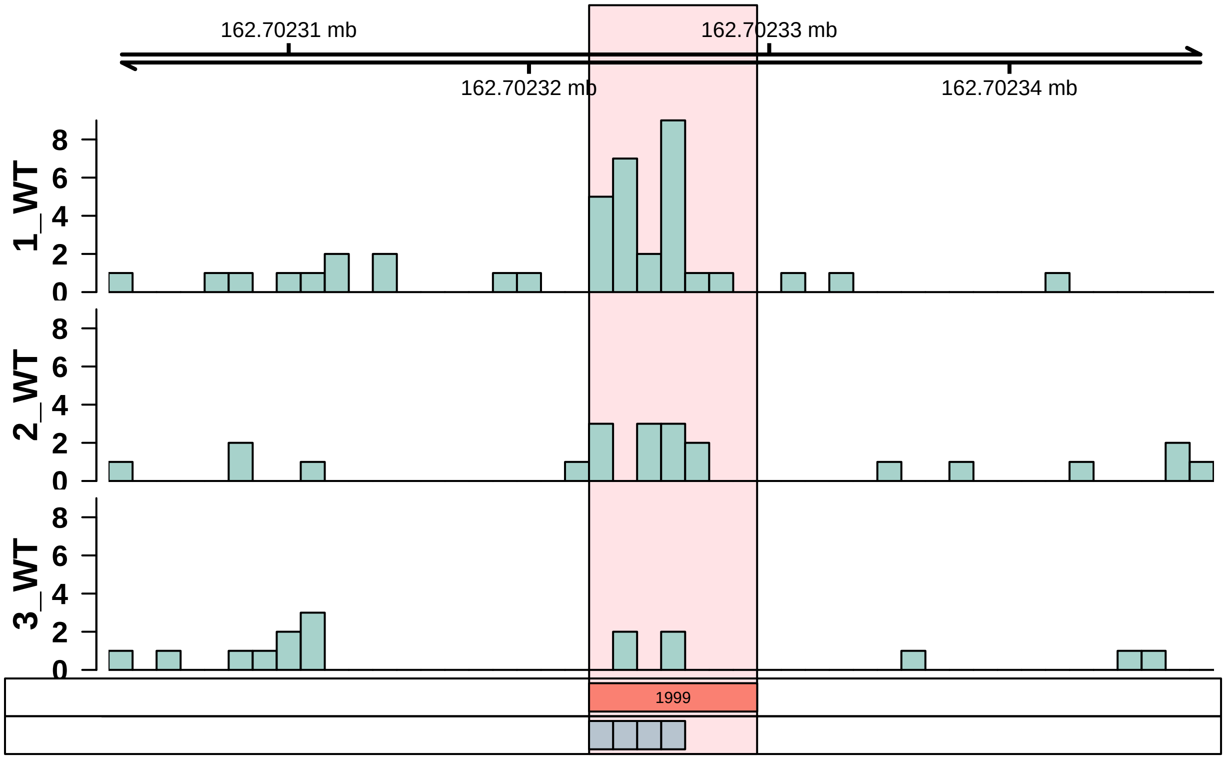
BS:26944/ chr17: 51818460−51818506 −



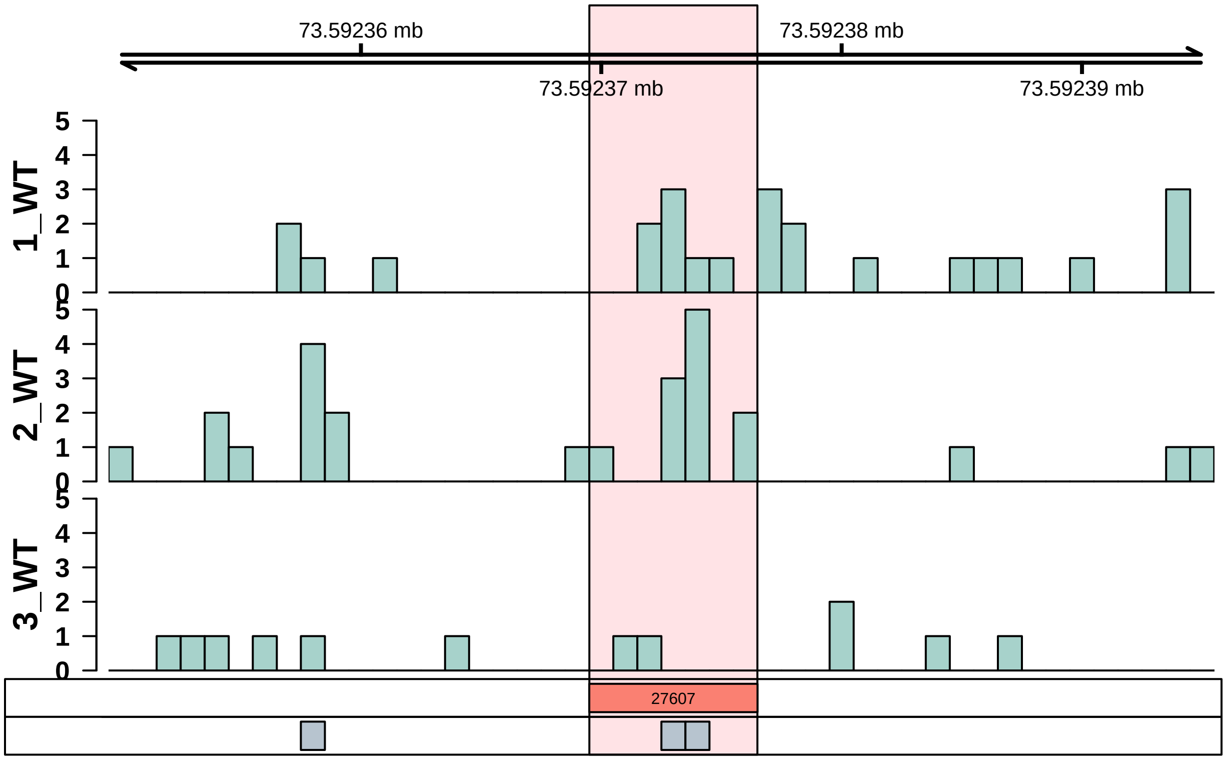
BS:3239/ chr2: 160759605−160759651 +



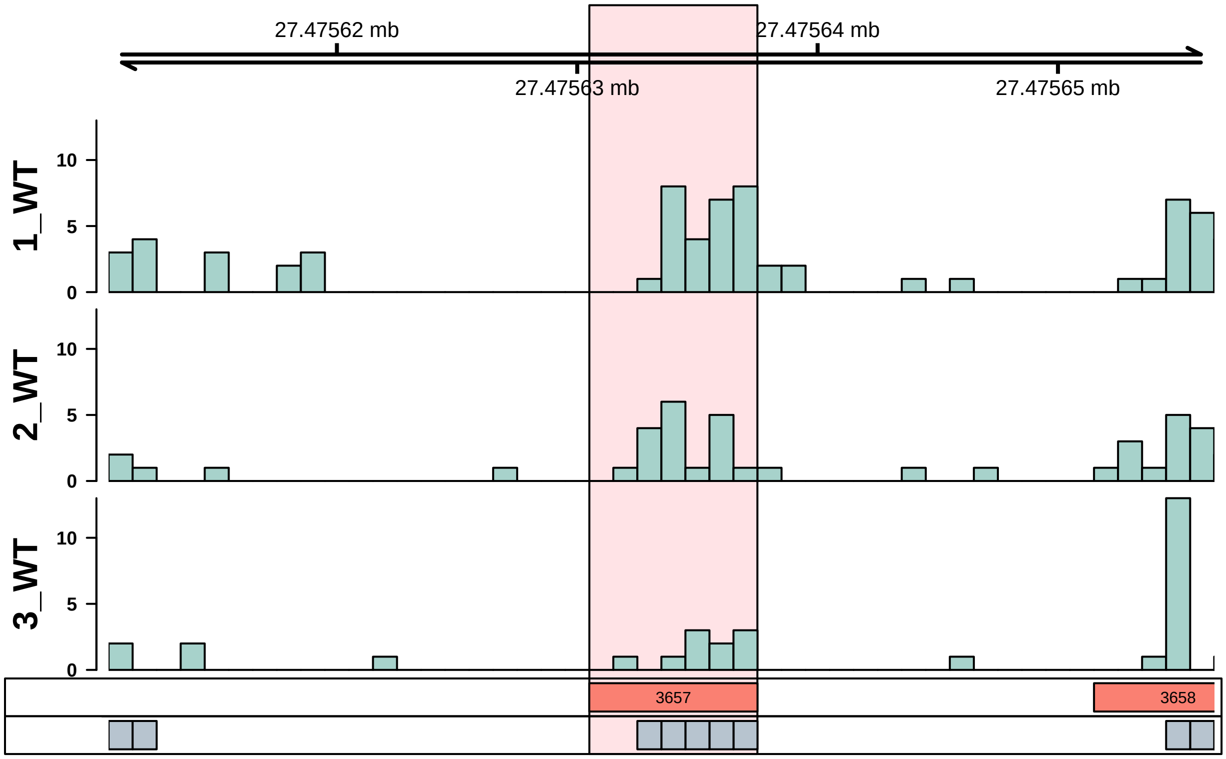
BS:1999/ chr1: 162702303−162702349 −



BS:27607/ chr18: 73592350−73592396 +



BS:3657/ chr2: 27475611−27475657 −



9 Downstream characterization

Next, we assign each binding site to the hosting gene and transcript part, using the initially loaded gene annotation from GENCODE.

9.1 Assignment of binding sites to genes

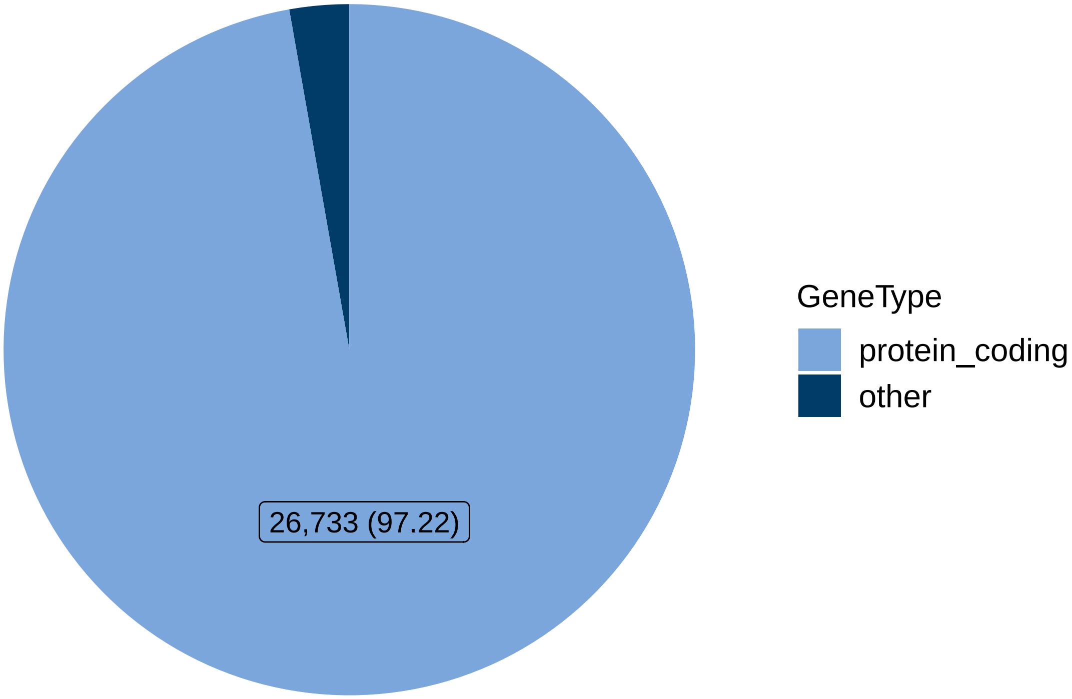
Assigning each binding site to its hosting gene is done by computing the overlap of all binding sites with the gene annotation. This typically results in some binding sites overlapping multiple different genes.

```
#-------------------------------------------------------------------------------
#   gene type priority rule
#-------------------------------------------------------------------------------
selectTerms = c("miRNA", "protein_coding", "tRNA", "lincRNA", "snRNA")
rule = unique(gns$gene_type)
rule = rule[!rule %in% selectTerms]
rule = c(selectTerms, rule)

# filter out pseudo genes
gns <- gns[!grepl(gns$gene_type, pattern = "pseudo")]
```

To resolve these overlaps genes are assigned based on the following hierarchical order: “protein_coding” >“tRNA” > “lincRNA” > “miRNA” > “snRNA”

```
#-------------------------------------------------------------------------------
#  Assign to genes
#-------------------------------------------------------------------------------

# get genes with binding signal
target_genes = subsetByOverlaps(gns, bdsFinal_gr)
df = findOverlaps(target_genes, bdsFinal_gr) %>% as.data.frame()

# split into easy and complex cases
idxDouble = df[duplicated(df$subjectHits),]
idxSingle = df[!duplicated(df$subjectHits),]

# handle single overlap cases
peaksRepoSingle = bdsFinal_gr[idxSingle$subjectHits]
mcols(peaksRepoSingle)$geneType = target_genes$gene_type[idxSingle$queryHits]
mcols(peaksRepoSingle)$geneName = target_genes$gene_name[idxSingle$queryHits]
mcols(peaksRepoSingle)$geneID = target_genes$gene_id[idxSingle$queryHits]%>% sub("\\..*", "", .)

# handle multi overlap cases
peaksRepoDouble = bdsFinal_gr[idxDouble$subjectHits]

peaksRepoDoubleCleaned = as(lapply(seq_along(peaksRepoDouble), function(x){
  currPeak = peaksRepoDouble[x]
  currtarget_genes = subsetByOverlaps(target_genes, currPeak)
  nOverlaps = length(currtarget_genes)

  # 1) take gene type as first criterion
  # -> prefer the type that is first in the `rule` list

  solution = unique(match(currtarget_genes$gene_type, rule))
  nSolutions = length(solution)
  if (nSolutions == nOverlaps) {
    # solution sucessfull
    mcols(currPeak)$geneType = currtarget_genes$gene_type[min(solution)]
    mcols(currPeak)$geneName = currtarget_genes$gene_name[min(solution)]
    mcols(currPeak)$geneID = currtarget_genes$gene_id[min(solution)]%>% sub("\\..*", "", .)
  }
  if (nSolutions < nOverlaps) {
    # no solution found
    # -> Stop and return NA
    mcols(currPeak)$geneType = NA
    mcols(currPeak)$geneName = NA
    mcols(currPeak)$geneID = NA
  }
  return(currPeak)
}),"GRangesList")
peaksRepoDoubleCleaned = unlist(peaksRepoDoubleCleaned)
peaksRepoDoubleCleaned = peaksRepoDoubleCleaned[!is.na(peaksRepoDoubleCleaned$geneID)]

# assign peaks

bsGene = c(peaksRepoSingle, peaksRepoDoubleCleaned)
bsGene = sortSeqlevels(bsGene)
bsGene = sort(bsGene)
bsGene = unique(bsGene) # why is the unique neccessary here?

# assign target_genes
target_genesGene = target_genes[target_genes$gene_id %in% bsGene$geneID]
```

```
#TODO: keep chunk?

df1 = data.frame(GeneType = (bsGene$geneType), type = "Peak") %>%
  mutate(GeneType = ifelse(grepl("pseudogene", GeneType), "pseudogene", GeneType)) %>%
  # mutate(GeneType = fct_lump_n(GeneType, n = 3)) %>%
  table() %>%
  as.data.frame() %>%
  mutate(label = paste0(format(Freq, big.mark = ",", decimal.mark = "."),
                        " (", format(round((Freq / sum(Freq))*100, digits = 2),
                                     big.mark = ",", decimal.mark = "."),")")) %>%
  mutate(GeneType = factor(GeneType, levels = c(GeneType[order(Freq)])))

df3 <- mutate(df1, GeneType = case_when(GeneType != "protein_coding" ~ "other", T ~ "protein_coding")) %
  group_by(GeneType)%>%
  summarize(Freq = sum(Freq))

p <- ggplot(df3, aes(y=Freq, x="", fill=GeneType)) +
     geom_col()+
     coord_polar(theta="y") +
  #    xlim(c(2, 4)) +
  geom_label(data = df1 %>% subset(GeneType == "protein_coding"), aes(y=Freq, x="", fill=GeneType,label
             position = position_stack(vjust = 0.5),
             show.legend = FALSE) +
  scale_fill_manual(values = c (farbe6, farbe4)) +
  theme_paper() +
  theme_nice_pie() +
  #theme(legend.position = "none") +
  guides(fill = guide_legend(reverse = TRUE)) +
  labs(y = NULL,
       x = NULL)
p
```



```
ggsave(p, filename = paste0(out, "Figure1C_bound_gene_types_AGO", Sys.Date(), ".pdf"), width = unit(8,
```

9.2 Assignment of binding sites to transcripts

```
### setting the hierarchical rule for assignment
rule = c("utr3", "utr5", "cds", "intron")
```

The transcript parts bound by the RBP are identified by overlapping the binding sites of protein-coding genes with the transcripts of these genes. The respective transcript region, such as intron, CDS or UTR can be deduced from these overlaps. Similar to the gene assignment, some binding sites might also overlap with multiple different annotated transcript parts. These are resolved by application of the hierarchical rule: utr3, utr5, cds, intron. Note that the majority vote system is not used here!

```
#-------------------------------------------------------------------------------
#  Assignment of binding sites to transcripts
#-------------------------------------------------------------------------------
# this is only done for protein coding genes
targetsProt = target_genes[target_genes$gene_type == "protein_coding"]
bsProt = bsGene[bsGene$geneType == "protein_coding"]
bsNonCodeing = bsGene[bsGene$geneType != "protein_coding"]
mcols(bsNonCodeing)$region = NA


### count the overlap of each binidng site within each part of the gene
# Count the overlaps of each binding site for each region of the transcript.
```

region	Freq
cds	6907
intron	6211
outside	25
utr3	11462
utr5	2128

```
cdseq = cds(annoDb)
intrns = unlist(intronsByTranscript(annoDb))
utrs3 = unlist(threeUTRsByTranscript(annoDb))
utrs5 = unlist(fiveUTRsByTranscript(annoDb))
regions = list(CDS = cdseq, Intron = intrns, UTR3 = utrs3, UTR5 = utrs5)
# Count the overlaps of each binding site fore each region of the transcript.
cdseq = regions$CDS %>% countOverlaps(bsProt,.)
intrns = regions$Intron %>% countOverlaps(bsProt,.)
utrs3 = regions$UTR3 %>% countOverlaps(bsProt,.)
utrs5 = regions$UTR5 %>% countOverlaps(bsProt,.)
countDf = data.frame(cds = cdseq, intron = intrns, utr3 = utrs3, utr5 = utrs5)


# sort by rule
countDf = countDf[, rule] %>%
  as.matrix() %>%
  cbind.data.frame(., outside = ifelse(rowSums(countDf) == 0, 1, 0) )
names = colnames(countDf)

# disable majority vote -> set all counts to 1
countDf = as.matrix(countDf)
countDf[countDf > 0] = 1

region = apply(countDf, 1, function(x){ names[which.max(x)] })

# add region annotation to binding sites object
mcols(bsProt)$region = region

kable(table(region))  %>%
  kable_material(c("striped", "hover"))
```

10 Save final binding sites

```
bs_annotated <- c(bsProt, bsNonCodeing)

saveRDS(bs_annotated, paste0(out, Sys.Date(), "-AGO_BS.rds"))
```

11 Session Info

```
sessionInfo()
```

```
## R version 4.2.2 (2022-10-31)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS Big Sur ... 10.16
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.2/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.2/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] grid      stats4    stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
##  [1] ggsci_2.9               ggpointdensity_0.1.0    tidyr_1.3.0
##  [4] tibble_3.1.8            patchwork_1.1.2         ggtext_0.1.2
##  [7] forcats_0.5.2           ComplexHeatmap_2.14.0   BindingSiteFinder_1.4.0
## [10] viridis_0.6.2           viridisLite_0.4.1       gridExtra_2.3
## [13] ggrepel_0.9.2           kableExtra_1.3.4        GenomicFeatures_1.50.4
## [16] UpSetR_1.4.0            reshape2_1.4.4          dplyr_1.0.10
## [19] AnnotationDbi_1.60.0    Biobase_2.58.0          ggplot2_3.4.0
## [22] rtracklayer_1.58.0      GenomicRanges_1.50.2    GenomeInfoDb_1.34.7
## [25] IRanges_2.32.0          S4Vectors_0.36.1        BiocGenerics_0.44.0
## [28] knitr_1.42
##
## loaded via a namespace (and not attached):
##   [1] backports_1.4.1             circlize_0.4.15
##   [3] Hmisc_4.7-2                 BiocFileCache_2.6.0
##   [5] systemfonts_1.0.4           plyr_1.8.8
##   [7] lazyeval_0.2.2              splines_4.2.2
##   [9] BiocParallel_1.32.5         digest_0.6.31
##  [11] foreach_1.5.2               ensembldb_2.22.0
##  [13] htmltools_0.5.4             magick_2.7.3
##  [15] fansi_1.0.4                 magrittr_2.0.3
##  [17] checkmate_2.1.0             memoise_2.0.1
##  [19] BSgenome_1.66.2             cluster_2.1.4
##  [21] doParallel_1.0.17           Biostrings_2.66.0
##  [23] matrixStats_0.63.0          svglite_2.1.1
##  [25] prettyunits_1.1.1           jpeg_0.1-10
##  [27] colorspace_2.1-0            blob_1.2.3
##  [29] rvest_1.0.3                 rappdirs_0.3.3
##  [31] textshaping_0.3.6           xfun_0.36
##  [33] crayon_1.5.2                RCurl_1.98-1.9
##  [35] survival_3.5-0              VariantAnnotation_1.44.0
##  [37] iterators_1.0.14            glue_1.6.2
##  [39] polyclip_1.10-4             gtable_0.3.1
##  [41] zlibbioc_1.44.0             XVector_0.38.0
##  [43] webshot_0.5.4               GetoptLong_1.0.5
##  [45] DelayedArray_0.24.0         car_3.1-1
##  [47] shape_1.4.6                 abind_1.4-5
##  [49] scales_1.2.1                DBI_1.1.3
##  [51] rstatix_0.7.1               Rcpp_1.0.10
##  [53] gridtext_0.1.5              progress_1.2.2
##  [55] htmlTable_2.4.1             clue_0.3-63
##  [57] foreign_0.8-84              bit_4.0.5
##  [59] Formula_1.2-4               htmlwidgets_1.6.1
##  [61] httr_1.4.4                  RColorBrewer_1.1-3
##  [63] ellipsis_0.3.2              pkgconfig_2.0.3
##  [65] XML_3.99-0.13               farver_2.1.1
##  [67] Gviz_1.42.0                 nnet_7.3-18
##  [69] dbplyr_2.3.0                deldir_1.0-6
##  [71] utf8_1.2.2                  labeling_0.4.2
##  [73] tidyselect_1.2.0            rlang_1.0.6
##  [75] munsell_0.5.0               tools_4.2.2
##  [77] cachem_1.0.6                cli_3.6.0
##  [79] generics_0.1.3              RSQLite_2.2.20
##  [81] broom_1.0.3                 evaluate_0.20
##  [83] stringr_1.5.0               fastmap_1.1.0
##  [85] ragg_1.2.5                  yaml_2.3.7
##  [87] bit64_4.0.5                 purrr_1.0.1
##  [89] KEGGREST_1.38.0             AnnotationFilter_1.22.0
##  [91] xml2_1.3.3                  biomaRt_2.54.0
##  [93] compiler_4.2.2              rstudioapi_0.14
##  [95] filelock_1.0.2              curl_5.0.0
##  [97] png_0.1-8                   ggsignif_0.6.4
##  [99] tweenr_2.0.2                stringi_1.7.12
## [101] highr_0.10                  lattice_0.20-45
## [103] ProtGenerics_1.30.0         Matrix_1.5-3
## [105] vctrs_0.5.2                 pillar_1.8.1
## [107] lifecycle_1.0.3             GlobalOptions_0.1.2
## [109] data.table_1.14.6           bitops_1.0-7
## [111] R6_2.5.1                    BiocIO_1.8.0
## [113] latticeExtra_0.6-30         codetools_0.2-18
## [115] dichromat_2.0-0.1           MASS_7.3-58.2
## [117] assertthat_0.2.1            SummarizedExperiment_1.28.0
## [119] rjson_0.2.21                withr_2.5.0
## [121] GenomicAlignments_1.34.0    Rsamtools_2.14.0
## [123] GenomeInfoDbData_1.2.9      parallel_4.2.2
## [125] hms_1.1.2                   rpart_4.1.19
## [127] rmarkdown_2.20              carData_3.0-5
## [129] MatrixGenerics_1.10.0       Cairo_1.6-0
## [131] ggpubr_0.5.0                biovizBase_1.46.0
## [133] ggforce_0.4.1               base64enc_0.1-3
## [135] interp_1.1-3                restfulr_0.0.15
```