mir181 KO AGO binding sites definition

Mirko Brüggemann, Melina Klostermann

13 September, 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)

here <- here::here()

source(paste0(here,"/Supporting_scripts/themes/theme_paper.R"))
source(paste0(here,"/Supporting_scripts/themes/CustomThemes.R"))


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

out <- paste0(here,"/Figure2/03_KOmir181_AGO_binding_site_definition/")
```

2 What was done?

- Here we define AGO binding sites on basis of the non-chimeric non-enriched miR-eCLIP data with the mir181 knockout.
- 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 <- readRDS(paste0(here, "/Supporting_scripts/annotation_preprocessing/annotation.rds"))
annoDb <- makeTxDbFromGRanges(annoDb)
gns <- readRDS(paste0(here, "/Supporting_scripts/annotation_preprocessing/gene_annotation.rds"))

# ----------------------------------------
# pureclip sites (from peak calling)
# ----------------------------------------
pureclip_sites <- "/Users/melinaklostermann/Documents/projects/AgoCLIP_miR181/pureclip/IP_WT_pureclip_si
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("KO", 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]]
```

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) == keep
    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)
```

–> 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
```

Table 1: Merge and combine

Option	nRanges
inputRanges	112,357
mergeCrosslinkSites	30,252
minCrosslinks	28,318
minClSites	27,670
centerIsClSite	27,065
centerIsSummit	23,449

```
bds = BSFDataSetFromBigWig(ranges = peaksFilteredPerGene, meta = colData)
bds
```

```
## Object of class BSFDataSet
## Contained ranges:  112.357
## ----> Number of chromosomes:  22
## ----> Ranges width:  1
## Contained Signal: 18,836,371
## Contained conditions:  WT
```

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

The final set of binding sites is computed with the same settings as the Ago2 binding sites (Figure1):

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

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.

seqnames	start	end	width	strand	scoreSum	scoreMean	scoreMax
chr1	4529032	4529038	7	+	20.37027	10.185135	13.34730
chr1	4529066	4529072	7	+	33.00360	16.501800	21.85370
chr1	6238329	6238335	7	+	36.87593	18.437965	32.46330
chr1	6245323	6245329	7	+	132.02399	22.003998	38.21990
chr1	6245604	6245610	7	+	28.79086	14.395430	23.35330
chr1	6245652	6245658	7	+	9.52553	4.762765	6.00678

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.

Finally, a binding sites is deemed reproducible if the thresholds are met for two out of three replicates.

8.1 Final binding sites

This leaves us with a final set of binding sites:

```
# -------------------------------------------------------
# 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 21852 reproducible binding sites.

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)
```

```r
bsGene = unique(bsGene) # why is the unique neccessary here?

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

9.2 Assignment of binding sites to transcripts

```r
### 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!

```r
#---------------------------------------------------------------------------
#  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.

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)] })
```

region	Freq
cds	5497
intron	4763
outside	25
utr3	9164
utr5	1774

```
# 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, "KOmir181_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_3.0.0             ggpointdensity_0.1.0    tidyr_1.3.0
##  [4] tibble_3.2.1            patchwork_1.1.2         ggtext_0.1.2
##  [7] forcats_1.0.0           ComplexHeatmap_2.14.0   BindingSiteFinder_1.4.0
## [10] viridis_0.6.3           viridisLite_0.4.2       gridExtra_2.3
## [13] ggrepel_0.9.3           kableExtra_1.3.4        GenomicFeatures_1.50.4
## [16] UpSetR_1.4.0            reshape2_1.4.4          dplyr_1.1.2
## [19] AnnotationDbi_1.60.2    Biobase_2.58.0          ggplot2_3.4.2
## [22] rtracklayer_1.58.0      GenomicRanges_1.50.2    GenomeInfoDb_1.34.9
## [25] IRanges_2.32.0          S4Vectors_0.36.2        BiocGenerics_0.44.0
## [28] knitr_1.43
##
```

```
## loaded via a namespace (and not attached):
##   [1] backports_1.4.1             circlize_0.4.15
##   [3] Hmisc_5.1-0                 BiocFileCache_2.6.1
##   [5] systemfonts_1.0.4           plyr_1.8.8
##   [7] lazyeval_0.2.2              BiocParallel_1.32.6
##   [9] digest_0.6.33               foreach_1.5.2
##  [11] ensembldb_2.22.0            htmltools_0.5.5
##  [13] fansi_1.0.4                 magrittr_2.0.3
##  [15] checkmate_2.2.0             memoise_2.0.1
##  [17] BSgenome_1.66.3             cluster_2.1.4
##  [19] doParallel_1.0.17           Biostrings_2.66.0
##  [21] matrixStats_1.0.0           svglite_2.1.1
##  [23] prettyunits_1.1.1           jpeg_0.1-10
##  [25] colorspace_2.1-0            blob_1.2.4
##  [27] rvest_1.0.3                 rappdirs_0.3.3
##  [29] xfun_0.39                   crayon_1.5.2
##  [31] RCurl_1.98-1.12             VariantAnnotation_1.44.1
##  [33] iterators_1.0.14            glue_1.6.2
##  [35] polyclip_1.10-4             gtable_0.3.3
##  [37] zlibbioc_1.44.0             XVector_0.38.0
##  [39] webshot_0.5.5               GetoptLong_1.0.5
##  [41] DelayedArray_0.24.0         shape_1.4.6
##  [43] scales_1.2.1                DBI_1.1.3
##  [45] Rcpp_1.0.11                 gridtext_0.1.5
##  [47] progress_1.2.2              htmlTable_2.4.1
##  [49] clue_0.3-64                 foreign_0.8-84
##  [51] bit_4.0.5                   Formula_1.2-5
##  [53] htmlwidgets_1.6.2           httr_1.4.6
##  [55] RColorBrewer_1.1-3          pkgconfig_2.0.3
##  [57] XML_3.99-0.14               farver_2.1.1
##  [59] Gviz_1.42.1                 nnet_7.3-19
##  [61] dbplyr_2.3.3                deldir_1.0-9
##  [63] here_1.0.1                  utf8_1.2.3
##  [65] tidyselect_1.2.0            rlang_1.1.1
##  [67] munsell_0.5.0               tools_4.2.2
##  [69] cachem_1.0.8                cli_3.6.1
##  [71] generics_0.1.3              RSQLite_2.3.1
##  [73] evaluate_0.21               stringr_1.5.0
##  [75] fastmap_1.1.1               yaml_2.3.7
##  [77] bit64_4.0.5                 purrr_1.0.1
##  [79] KEGGREST_1.38.0             AnnotationFilter_1.22.0
##  [81] xml2_1.3.5                  biomaRt_2.54.1
##  [83] compiler_4.2.2              rstudioapi_0.15.0
##  [85] filelock_1.0.2              curl_5.0.1
##  [87] png_0.1-8                   tweenr_2.0.2
##  [89] stringi_1.7.12              lattice_0.21-8
##  [91] ProtGenerics_1.30.0         Matrix_1.5-4.1
##  [93] vctrs_0.6.3                 pillar_1.9.0
##  [95] lifecycle_1.0.3             GlobalOptions_0.1.2
##  [97] data.table_1.14.8           bitops_1.0-7
##  [99] R6_2.5.1                    BiocIO_1.8.0
## [101] latticeExtra_0.6-30         codetools_0.2-19
## [103] dichromat_2.0-0.1           MASS_7.3-60
## [105] SummarizedExperiment_1.28.0 rprojroot_2.0.3
## [107] rjson_0.2.21                withr_2.5.0
## [109] GenomicAlignments_1.34.1    Rsamtools_2.14.0
## [111] GenomeInfoDbData_1.2.9      parallel_4.2.2
## [113] hms_1.1.3                   rpart_4.1.19
## [115] rmarkdown_2.23              MatrixGenerics_1.10.0
## [117] biovizBase_1.46.0           ggforce_0.4.1
## [119] base64enc_0.1-3             interp_1.1-4
## [121] restfulr_0.0.15
```