An end-to-end ChIP-seq workflow: from a peak BED to enriched motifs

Benjamin Jean-Marie Tremblay*

23 May 2026

Abstract

This vignette walks through a complete ChIP-seq motif analysis on real data. Starting from a peak BED file bundled in `inst/extdata`, we extract peak sequences from a BSgenome, discover motifs de novo with `motif_finder()`, deduplicate them with `merge_similar_lite()`, compare them against the JASPAR2022 plant collection via `MotifDb`, validate enrichment against a composition-matched background with `match_bkg()` and `enrich_motifs_lite()`, scan all peaks with `scan_sequences_lite()`, test for central positional bias with `motif_peaks()`, and look for motif clustering with `motif_coocc()`. The closing section assembles a summary table that combines all of the per-motif evidence into one ranked list.

Contents

1 Introduction **1**

2 Loading peaks from a BED file **2**

3 Extracting peak sequences and building a matched background **3**

4 De novo motif discovery **4**

5 Deduplicating the discovered set **6**

6 Comparing against JASPAR via MotifDb **7**

7 Validating enrichment against a composition-matched background **8**

8 Scanning all peaks and testing positional bias **9**

9 Motif clustering with `motif_coocc()` **10**

10 Summary table **13**

11 Session info **14**

References **16**

1 Introduction

This vignette ties together the analysis functions of the `universalmotif` package into a single end-to-end ChIP-seq workflow. The previous vignettes cover individual pieces in isolation: for a introduction to sequence

*benjamin.tremblay@uwaterloo.ca

motifs, see the introductory vignette; for the plumbing of motif objects (I/O, conversion, logos), see the motif manipulation vignette; and for the mechanics of scanning, shuffling, and enrichment, see the sequence searches vignette. This document assumes you have skimmed the first and last of those.

The dataset is a real ChIP-seq peak call set for the *Arabidopsis thaliana* B3-domain repressor VAL1, taken from Yuan et al. (2021). VAL1 and its paralogue VAL2 silence seed-maturation genes by binding the RY element (`CATGCA` / `TGCATG`) and recruiting Polycomb Repressive Complex 2. We expect a properly working motif-discovery pipeline, fed only the peaks, to recover the RY element and to match it against B3-domain transcription factors in JASPAR.

The workflow proceeds in eight steps:

1. Import the peak BED.
2. Extract peak sequences from a BSgenome.
3. Discover motifs *de novo* with `motif_finder()`.
4. Deduplicate the discovered set with `merge_similar_lite()`.
5. Compare the survivors to JASPAR via `compare_motifs_lite()`.
6. Validate enrichment against a composition-matched background with `match_bkg()` and `enrich_motifs_lite()`.
7. Map hit positions with `scan_sequences_lite()` and test for central bias with `motif_peaks()`.
8. Look for motif clustering with `motif_coocc()`.

We close with a summary table that ranks the discovered motifs by the combined evidence.

2 Loading peaks from a BED file

`universalmotif` does not ship its own BED importer because `rtracklayer::import()` already does the job well. The peak file lives in `inst/extdata`; `system.file()` resolves it to an absolute path inside the installed package.

```
library(rtracklayer)

bed.file <- system.file("extdata",
                        "VAL1-GFP_val1_calledPeaks.bed",
                        package = "universalmotif")
peaks <- import(bed.file, format = "BED")
peaks
#> GRanges object with 5318 ranges and 2 metadata columns:
#>          seqnames            ranges strand |        name     score
#>             <Rle>         <IRanges>  <Rle> | <character> <numeric>
#>      [1]        1       21526-22017      * |        <NA>        86
#>      [2]        1       48915-49492      * |        <NA>        86
#>      [3]        1     104136-106515      * |        <NA>      1000
#>      [4]        1     116060-116989      * |        <NA>       923
#>      [5]        1     118242-119259      * |        <NA>       909
#>      ...      ...               ...    ... .         ...       ...
#>   [5314]        3 23354908-23355637      * |        <NA>       840
#>   [5315]        3 23355969-23356509      * |        <NA>        86
#>   [5316]        3 23401260-23402126      * |        <NA>       224
#>   [5317]        3 23410489-23411522      * |        <NA>        86
#>   [5318]        3 23452605-23453345      * |        <NA>       151
#>   -------
#>   seqinfo: 5 sequences from an unspecified genome; no seqlengths
```

The BED file uses bare numeric chromosome names (`"1"` through `"5"`), whereas the matching BSgenome we will use in the next section uses the `"Chr1"`..`"Chr5"` convention. A single `seqlevels()` assignment fixes that, and we drop any mitochondrial or plastid contigs at the same time (there are none in this peak set, but it is

a useful defensive habit).

```
seqlevels(peaks) <- paste0("Chr", seqlevels(peaks))
peaks <- keepSeqlevels(peaks,
                       paste0("Chr", 1:5),
                       pruning.mode = "coarse")
summary(width(peaks))
#>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#>   255.0   598.0   807.0   984.4  1190.0  5337.0
length(peaks)
#> [1] 5318
```

3 Extracting peak sequences and building a matched background

We use the TAIR9 BSgenome and resize every peak to a fixed 500 bp window centred on the called peak. The fixed window matters because `motif_peaks()` later treats positions as comparable across sequences; if the peaks varied in length, the "centre" would mean different things in different rows.

```
library(BSgenome.Athaliana.TAIR.TAIR9)
#> Loading required package: BSgenome
#> Loading required package: BiocIO

peaks.500 <- resize(peaks, width = 500, fix = "center")
## Drop any windows that ran off a chromosome end.
peaks.500 <- trim(peaks.500)
peaks.500 <- peaks.500[width(peaks.500) == 500]

peak.seqs <- getSeq(Athaliana, peaks.500)
names(peak.seqs) <- paste0("peak_", seq_along(peak.seqs))
peak.seqs
#> DNAStringSet object of length 5318:
#>         width sequence                                          names
#>     [1]   500 GAATGGGATGGTAATAAAGAAA...ACGTTCTTTCTAGTAGTTATA peak_1
#>     [2]   500 GGATTGCTTCTCCTTGACAACT...TACAAATATAATCCTTGTTAA peak_2
#>     [3]   500 CTCCGGCACCGGAAAAGATCCG...TTTATACAAAAAAATAATATA peak_3
#>     [4]   500 CTATATTCAATTATTCATGTTT...AAAATAATTATAGTTAAAATA peak_4
#>     [5]   500 TGCTCCAGGAGGATGCGCTTAA...AGATAGAGACAGTGCTCCTAT peak_5
#>     ...   ... ...
#> [5314]   500 TTTGGTACAATTATTGAGAAAC...ACGGCGATACATTCATTATTT peak_5314
#> [5315]   500 CACTTATTTGACCCAAAAAGTT...TTGTTTTCATGCCTCAAGGTC peak_5315
#> [5316]   500 TTGCTTCATCAATCCTATATGT...GCTTGTTGTTGAATATGGGCC peak_5316
#> [5317]   500 AGAAAAAAAGGATTTAGATTGA...AGCTTAACAAAATCCCAACCA peak_5317
#> [5318]   500 AAATTGTAAATTGAACGGGGCT...AGTATTCTTATGATCGTCTCT peak_5318
```

Plant promoter sequences are strongly AT-skewed, so any background analysis on these peaks needs to control for composition. We build that control once, here, and then reuse it for both the *de novo* discovery in §4 and the enrichment validation in §7. `match_bkg()` accepts a `BSgenome` directly, so we can pass `genome = Athaliana` together with `exclude = peaks.500`, and it then samples random non-peak genomic windows internally, matching each peak on GC content and length.

```
set.seed(2026)
bkg.seqs <- match_bkg(peak.seqs, genome = Athaliana, exclude = peaks.500)
length(bkg.seqs)
#> [1] 5318
```

It is worth a quick sanity check that the peak set and the matched background really do have more or less indistinguishable GC distributions:

```
summary(letterFrequency(peak.seqs, "GC", as.prob = TRUE))
#>       G|C
#>  Min.   :0.2160
#>  1st Qu.:0.3160
#>  Median :0.3500
#>  Mean   :0.3598
#>  3rd Qu.:0.3940
#>  Max.   :0.5860
summary(letterFrequency(bkg.seqs,  "GC", as.prob = TRUE))
#>       G|C
#>  Min.   :0.2020
#>  1st Qu.:0.3180
#>  Median :0.3500
#>  Mean   :0.3596
#>  3rd Qu.:0.3940
#>  Max.   :0.5820
```

4 De novo motif discovery

`motif_finder()` is a self-contained k-mer enumeration discoverer, and (handily) it does not require an external MEME installation. By default it builds its own negative pool by shuffling the input at `shuffle.k = 2`, but passing `bkg.sequences = bkg.seqs` (our composition-matched genomic pool from §3) tends to work meaningfully better on plant-promoter ChIP-seq. The reason is that shuffling each peak preserves that peak's own AT-skew, so AT-rich k-mers cannot enrich against it, and everything past the most extreme signal ends up effectively hidden. Against a real genomic background instead, the lower-abundance motifs (G-box, E-box) come through alongside the dominant RY element. On this dataset (5,318 peaks of 500 bp each, two threads) the call returns in roughly 15 seconds, so there is no need to sub-sample.

```
discovered.df <- motif_finder(peak.seqs,
                              bkg.sequences = bkg.seqs,
                              nmotifs       = 5,
                              min.width     = 6,
                              max.width     = 12,
                              rng.seed      = 2026,
                              nthreads      = 2)
print(as.data.frame(discovered.df[, c("name", "consensus", "pval")]),
      row.names = FALSE)
#>       name    consensus          pval
#>   motif_1      CATGCAA 1.281703e-264
#>   motif_2    ATGCATGCA 9.067132e-252
#>   motif_3    ATGCATGTG  5.155211e-75
#>   motif_4      GTACGTR  8.765155e-70
#>   motif_5     CACGTGTC  7.214501e-69
#>   motif_6  TTYCATGCAAA  2.378602e-62
#>   motif_7   CATGCACATG  2.014108e-56
#>   motif_8      CATGSAG  3.485077e-40
#>   motif_9  AAAAAAAAAAA  1.238932e-19
#>  motif_10 GAGAGAGAGAGA  1.486656e-16
#>  motif_11   AAATAAAATA  1.753546e-09
#>  motif_12   ATATATATAT  9.577262e-08
```

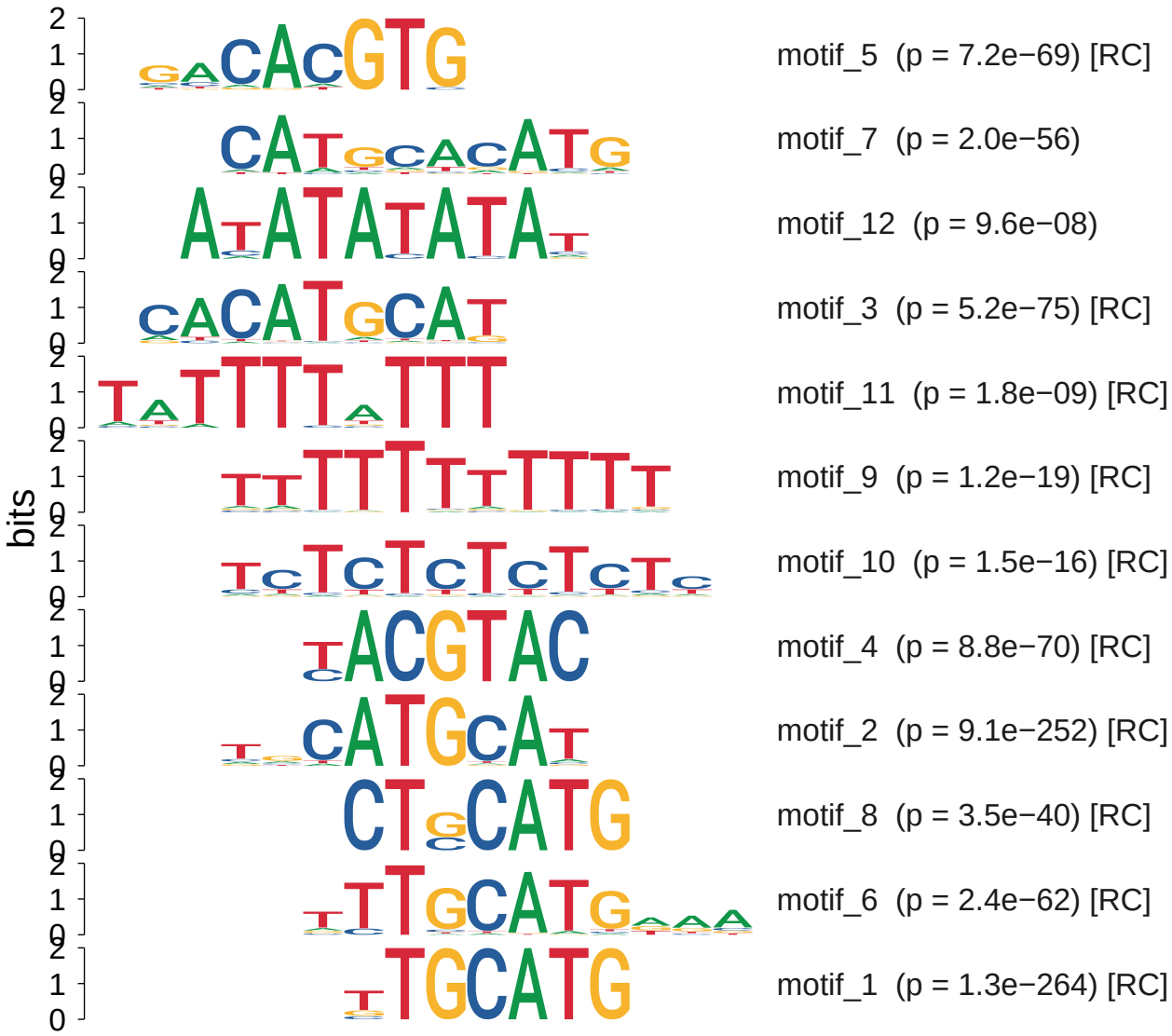
`motif_finder()` returns a `universalmotif_df` whose `motif` column wraps the actual motif objects in an

`AsIs` envelope. `to_list()` unwraps it back into a plain list, which is what every downstream function expects.

```
discovered <- to_list(discovered.df, extrainfo = FALSE)
#> Discarding unknown slot(s) 'rank', 'width', 'seqs_pos', 'seqs_neg',
#>   'sites_pos', 'sites_neg', 'n_pos', 'n_neg', 'pvalue', 'qvalue' (set
#>   `extrainfo=TRUE` to preserve these).
```

For viewing, we will create a copy which has the discovery p-value added to the name:

```
discovered.labelled <- discovered
for (i in seq_along(discovered.labelled)) {
  discovered.labelled[[i]]["name"] <- sprintf(
    "%s  (p = %.1e)",
    discovered.labelled[[i]]["name"],
    discovered.df$pval[i]
  )
}
view_motifs_lite(discovered.labelled, sort.by = "similarity",
             show.positions = FALSE, names.pos = "right")
```



With `sort.by = "similarity"`, `view_motifs_lite()` reorders the panels by hierarchical clustering on Pearson correlation between aligned columns, so visually related logos are adjacent. The RY-element variants (consensuses containing `CATGCA`) cluster together; so do the G-box (`CACGTG`) variants, GAGA repeats, and low-complexity AT-rich runs. The next two sections (enrichment against a composition-matched background, and positional bias) are what tell us which of these clusters carry a real binding signal and which only reflect background composition. Pass `sort.by = "ic"` (the default) for descending information content, or `sort.by = "none"` to keep `motif_finder()`'s p-value ranking.

5 Deduplicating the discovered set

A real motif-discovery run almost always returns several rotated or extended copies of the same underlying signal. These variations can be biologically meaningful, but for simplicity we will not dwell on them here. Instead we merge them with `merge_similar_lite()`, which collapses each cluster down to a single representative by hierarchical clustering on the q-values from a Pearson-based comparison. It is often wise to trim the merged motifs afterwards as well, since merging tends to fold mismatching flanks into low-IC positions.

```
discovered.merged <- merge_similar_lite(discovered, qvalue = 0.01)
discovered.merged <- trim_motifs(discovered.merged)

length(discovered)
#> [1] 12
length(discovered.merged)
#> [1] 7
```

`merge_similar_lite()` names each merged motif by joining the names of its contributing motifs with `+`, which becomes unwieldy quite quickly when several distinct binding signals each produce their own cluster (the RY element, the bZIP G-box, and so on). So we relabel each merged cluster by inspecting its consensus, so that the rest of the vignette can refer to motifs by what they *are*:

```
label_merged <- function(consensus) {
  if (grepl("CATGCA", consensus))                    "RY"
  else if (grepl("CACGTG", consensus))               "G_box"
  else if (grepl("ACGTAC|GTACGT", consensus))        "bZIP"
  else if (grepl("AAAAAA|ATATAT|AAATAA", consensus)) "AT_rich"
  else if (grepl("GAGAGA", consensus))               "GA_repeat"
  else                                               consensus
}
new.names <- vapply(discovered.merged,
                    function(m) m["name"], character(1))
for (i in seq_along(discovered.merged)) {
  new.names[i] <- label_merged(
    discovered.merged[[i]]["consensus"]
  )
  discovered.merged[[i]]["name"] <- new.names[i]
}
new.names
#> [1] "RY"        "bZIP"      "G_box"     "AT_rich"   "GA_repeat" "AT_rich"
#> [7] "AT_rich"
```

```
view_motifs_lite(discovered.merged, sort.by = "similarity",
             show.positions = FALSE, names.pos = "right")
```

6 Comparing against JASPAR via MotifDb

For the cross-database comparison we use `MotifDb`, which already ships JASPAR2022 plant motifs and is therefore the lightest path to a reference set. We narrow it to *Arabidopsis* JASPAR2022 entries and convert the resulting list to `universalmotif` objects.

```
suppressPackageStartupMessages(library(MotifDb))

ath.jaspar <- query(query(MotifDb, "Athaliana"), "jaspar2022")
length(ath.jaspar)
#> [1] 568
ath.motifs <- convert_motifs(ath.jaspar)
```

`compare_motifs_lite()` in long-format mode treats indexed entries of `motifs` as queries and the whole input as the target database, so we concatenate the discovered set with the reference and pass the query indices via `compare.to`.

```
combined <- c(discovered.merged, ath.motifs)
n.disc   <- length(discovered.merged)

cmp <- compare_motifs_lite(combined,
                           compare.to = seq_len(n.disc),
                           qvalue     = 0.2,
                           nthreads   = 2)
cmp <- cmp[cmp$subject != cmp$target, ]
cmp <- cmp[order(cmp$qvalue), ]
print(head(cmp[, c("subject", "target", "score", "qvalue",
                   "subject.consensus", "target.consensus")], 10),
      row.names = FALSE)
#>    subject target     score       qvalue subject.consensus target.consensus
```

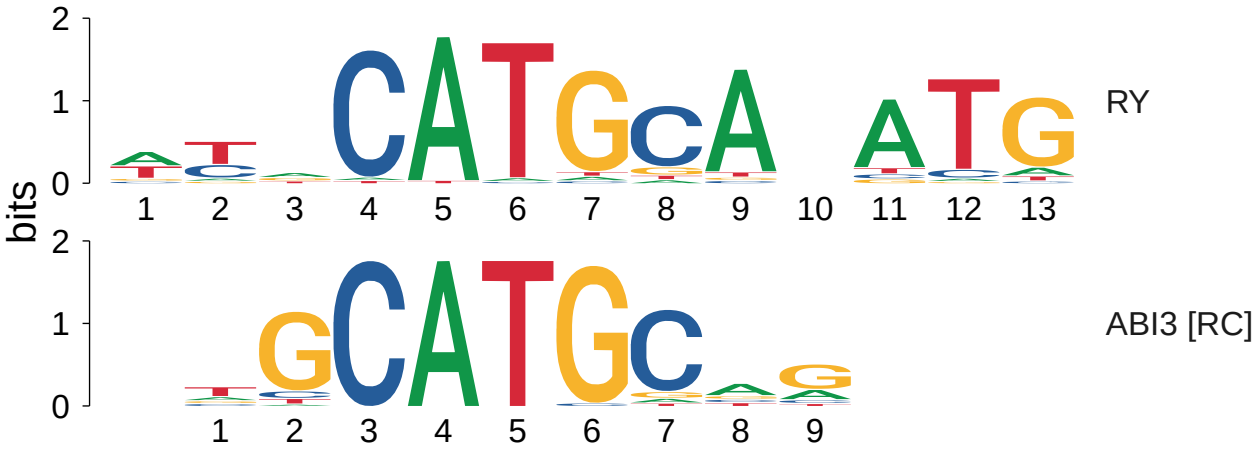
```
#>  GA_repeat   BPC1 0.9952740 2.144867e-08      GAGAGAGAGAGA     GAGAGAGAGAGA
#>  GA_repeat   BPC6 0.9920294 1.119204e-07      GAGAGAGAGAGA     TCTCTCTCTCTC
#>  GA_repeat   BPC5 0.9900327 8.038615e-07      GAGAGAGAGAGA     GAGAGAGAGAGA
#>    AT_rich   CDF5 0.9840743 3.485935e-05       AAAAAAAAAAA      TTTTTTTTTTT
#>    AT_rich DOF5.8 0.9735805 7.730949e-05       AAAAAAAAAAA      TTTTTNNNNTN
#>    AT_rich DOF3.6 0.9544933 1.749803e-04       AAAAAAAAAAA      TTTTTNNYTTT
#>    AT_rich DOF3.4 0.9317020 4.494503e-04       AAAAAAAAAAA      NNNYTYNTTTT
#>      G_box   ABF3 0.9924118 9.372888e-04          CACGTGTC         NACGTGTC
#>      G_box  DPBF3 0.9919857 9.893625e-04          CACGTGTC         CACGTGKC
#>      G_box   ABI5 0.9882467 9.893625e-04          CACGTGTC         GMCACGTG
```

Most of the strong matches will be for the AT-rich or GA-rich low-complexity discovered motifs (DOF, BPC, and so on) rather than for the RY element itself; the *Arabidopsis* genome is AT-skewed, and `motif_finder()` reports every strong k-mer signal, not only the one that drove the immunoprecipitation. It is the enrichment and positional-bias tests in the next two sections that separate the genuine binding signal from these incidental low-complexity hits.

For the figure we pick the RY motif and pair it with its best JASPAR match in the B3-domain family. ABI3, FUS3, and LEC2 are the three B3-domain TFs that JASPAR2022 has for *Arabidopsis*, and any of them is a biologically sensible neighbour for VAL1.

```
ref.names <- vapply(ath.motifs, function(m) m["name"], character(1))
ry.cmp <- cmp[cmp$subject == "RY", ]
ry.best <- ry.cmp[order(ry.cmp$qvalue), ][1, ]

top.disc <- discovered.merged[[match("RY", new.names)]]
top.ref  <- ath.motifs[[match(ry.best$target, ref.names)]]
view_motifs_lite(c(top.disc, top.ref), show.positions.once = FALSE,
             names.pos = "right")
```



7 Validating enrichment against a composition-matched background

We can now use `enrich_motifs_lite()` to validate our new set of merged motifs. It accepts an explicit `bkg.sequences` argument, so we can hand it a background that controls for the AT-skew of plant promoters. We already built exactly that pool back in §3 (`bkg.seqs`, from `match_bkg()` against a universe of random non-peak genomic windows), and so the enrichment step simply reuses it.

So we feed the matched background straight into `enrich_motifs_lite()`:

```
enr <- enrich_motifs_lite(motifs        = discovered.merged,
                          sequences     = peak.seqs,
                          bkg.sequences = bkg.seqs,
                          pvalue        = 1e-4,
                          qvalue        = 0.05,
                          nthreads      = 2)
print(enr[, c("motif", "consensus", "enrichment", "qvalue")],
      row.names = FALSE)
#>       motif     consensus enrichment        qvalue
#>          RY WYNCATGCANATG   2.816361 2.733202e-149
#>        bZIP       GTACGTR   3.236515  5.205173e-73
#>       G_box      CACGTGTC   2.432018  6.234305e-73
#>    AT_rich   AAAAAAAAAAA   1.307190  2.137234e-25
#>  GA_repeat  GAGAGAGAGAGA   1.311024  1.241422e-22
#>  AT_rich.1    AAATAAAATA   1.338129  4.805244e-15
#>  AT_rich.2    ATATATATAT   1.377899  2.264233e-13
```

With a GC-matched background, the AT-rich low-complexity motifs drop to barely above 1-fold enrichment (since they are common in essentially every AT-skewed plant-promoter window), while the genuine binding motifs stand out at several fold over background. That contrast all but disappears if you compare against a uniform shuffled null instead, which is really why a matched background of this kind matters so much for plant ChIP-seq.

For the alternative dinucleotide-shuffle null (`shuffle.k = 2`), or for a single-shuffle background, see the `match_bkg()` and shuffling sections of the sequence searches vignette.

8 Scanning all peaks and testing positional bias

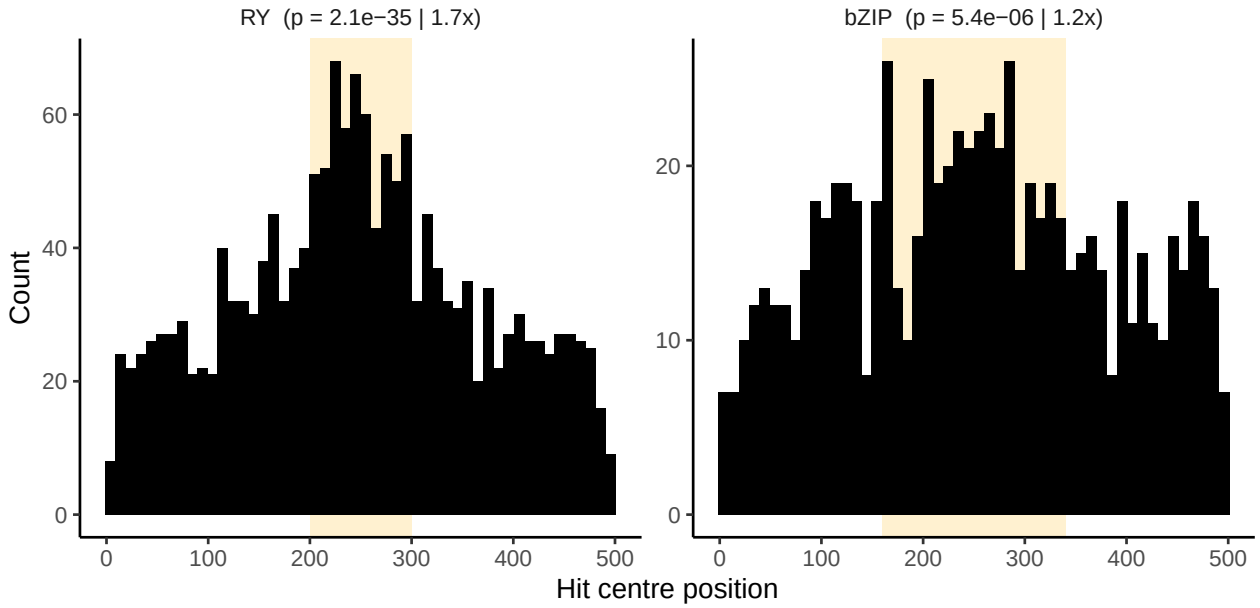
To map hits at single-base resolution across every peak we run `scan_sequences_lite()` with the discovered motifs. Because every window is a fixed 500 bp, the motif positions are directly comparable from one row to the next.

```
hits <- scan_sequences_lite(discovered.merged,
                            peak.seqs,
                            pvalue         = 1e-4,
                            return.granges = FALSE,
                            nthreads       = 2)
nrow(hits)
#> [1] 26085
table(motif = hits$motif)
#> motif
#>   AT_rich AT_rich.1 AT_rich.2 GA_repeat     G_box        RY      bZIP
#>      8736      1639      3256      7456      1710      2257      1031
```

`motif_peaks()` then asks, for each motif in turn, whether its hits cluster near the centre of the peak. It is a non-parametric, CentriMo-style test: at each window size it compares the observed number of hits inside the window against the number we would expect under a uniform-position null.

```
pk <- motif_peaks(hits, seq.length = 500, mode = "central")
print(pk[, c("motif", "best.center", "best.window", "qvalue")],
      row.names = FALSE)
#>  motif best.center best.window       qvalue
#>     RY         250         100 1.435792e-34
#>   bZIP         250         180 1.901773e-05
```

```
plot_motif_peaks(pk)
```



A motif that drives binding should peak sharply at position 250, the centre of the 500 bp window. The RY element does (very strongly, since VAL1 is the immunoprecipitated factor); the G-box / bZIP motifs are also centrally enriched but with a broader best window, consistent with bZIP TFs co-binding nearby without being the primary signal. The low-complexity AT-rich motifs are filtered out by the `motif_peaks()` default `qvalue = 0.1` threshold.

9 Motif clustering with `motif_coocc()`

`motif_coocc()` asks whether motif pairs appear together in the same peak (and, optionally, within `max.distance` bp of each other) more often than expected by chance. For VAL1 the interesting question is not so much "which pair?" as "does the RY element cluster with itself inside peaks?". B3-domain TFs often bind cooperatively to tandem RY elements, and ChIP-seq peaks are wide enough to accommodate several copies. We turn on `self.pairs` so the (motif, motif) self-pairs are included:

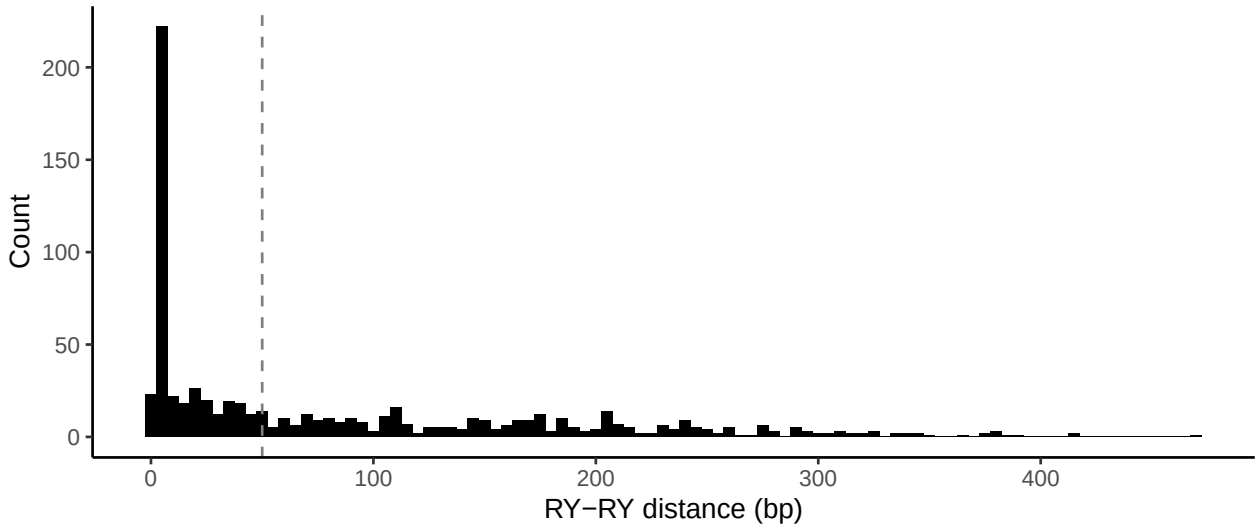
```
co <- motif_coocc(discovered.merged,
                  hits         = hits,
                  n.sequences  = length(peak.seqs),
                  max.distance = 50L,
                  self.pairs   = TRUE)
print(co[, c("motif_a", "motif_b", "both", "both.clustered",
             "median.distance", "qvalue")],
      row.names = FALSE)
#>    motif_a   motif_b both both.clustered median.distance        qvalue
#>    AT_rich AT_rich.1  887            671             0.0 6.257471e-111
#>         RY AT_rich.2  403            141            14.0  6.314864e-09
#>       bZIP AT_rich.2  194             63            23.0  4.483329e-05
#>         RY AT_rich.1  469             88            26.5  4.191138e-04
#>  AT_rich.1 AT_rich.2  294             88            26.0  4.395228e-04
#>    AT_rich AT_rich.2  461            134            28.0  4.418209e-03
#>         RY   AT_rich  747            168            32.0  5.528165e-03
#>         RY      bZIP  274            117            17.0  4.054286e-02
#>       bZIP     G_box  172             55            30.0  4.641932e-01
```

```
#>    AT_rich GA_repeat  838           299           16.0  5.553360e-01
#>      G_box GA_repeat  425           106           22.0  5.573071e-01
#>       bZIP   AT_rich  321            74           29.0  9.929953e-01
#>         RY     G_box  344           123           17.0  9.999972e-01
#>       bZIP AT_rich.1  174            35           30.0  9.999972e-01
#>  GA_repeat AT_rich.1  452           123           23.0  9.999972e-01
#>       bZIP GA_repeat  258            53           26.0  9.999972e-01
#>      G_box AT_rich.2  175            25           32.0  9.999972e-01
#>      G_box AT_rich.1  232            38           35.0  9.999972e-01
#>      G_box   AT_rich  409            82           27.5  9.999972e-01
#>  GA_repeat AT_rich.2  325            69           28.0  9.999972e-01
#>         RY GA_repeat  560           145           26.0  9.999972e-01
#>         RY        RY  440           290            3.0            NA
#>       bZIP      bZIP  193           154            1.0            NA
#>      G_box     G_box  437           382            2.0            NA
#>    AT_rich   AT_rich 1536          1465            1.0            NA
#>  GA_repeat GA_repeat 1382          1332            2.0            NA
#>  AT_rich.1 AT_rich.1  266           150            5.0            NA
#>  AT_rich.2 AT_rich.2  505           465            2.0            NA
```

The (`RY`, `RY`) self-pair shows how often the RY element appears in tight clusters within a peak (`both.clustered` counts sequences where two RY hits land within 50 bp of each other, with `median.distance` giving the typical spacing). A high `both.clustered` for the RY self-pair, with a small `median.distance`, is the cooperative-binding signature: multiple RY copies packed together at the binding locus. The cross-motif rows give the rest of the picture: pairwise co-occurrence of the RY motif with each low-complexity neighbour is generally weak (no obligate cofactor), and the strongest cross-pairs are between the low-complexity motifs themselves, reflecting plant promoter composition rather than specific cobinding.

`motif_coocc()` reports the summary statistic, but it is well worth plotting the actual distribution too. For every peak carrying two or more RY hits, we compute all of the pairwise distances between the RY hit starts, and histogram them.

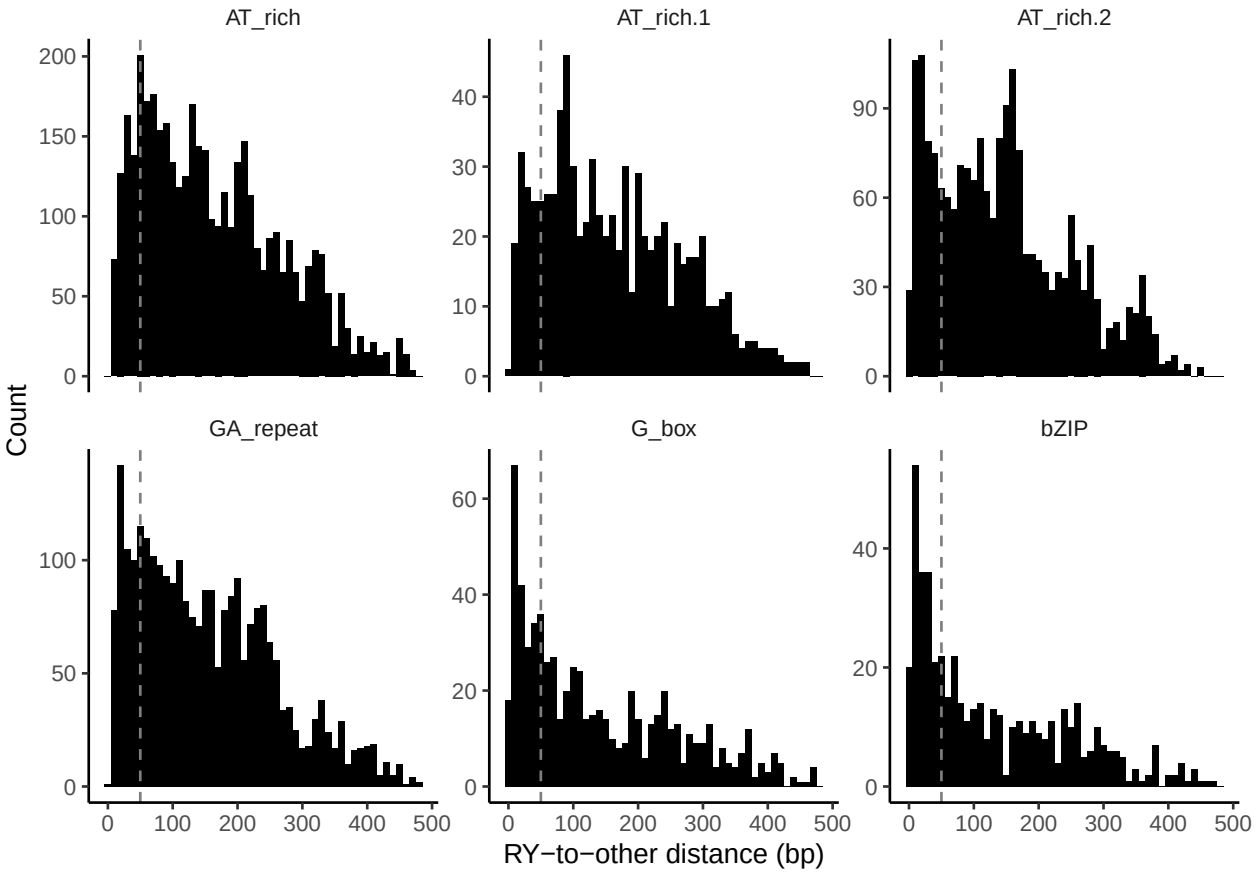
```
ry.hits  <- hits[hits$motif == "RY", ]
ry.dists <- unlist(lapply(split(ry.hits$start, ry.hits$sequence.i),
                          function(s) if (length(s) < 2L)
                            integer(0) else as.integer(dist(s))))
length(ry.dists)
#> [1] 742
ggplot2::ggplot(data.frame(distance = ry.dists),
                ggplot2::aes(x = .data$distance)) +
  ggplot2::geom_histogram(binwidth = 5, fill = "black", colour = NA) +
  ggplot2::geom_vline(xintercept = 50, linetype = "dashed",
                      colour = "grey50") +
  ggplot2::labs(x = "RY-RY distance (bp)", y = "Count") +
  ggplot2::theme_bw() +
  ggplot2::theme(panel.grid         = ggplot2::element_blank(),
                 panel.border       = ggplot2::element_blank(),
                 axis.line.x.bottom = ggplot2::element_line(colour = "black"),
                 axis.line.y.left   = ggplot2::element_line(colour = "black"))
```



The dashed line marks the 50 bp threshold used in `max.distance` above. A heavy tail at very short spacings (under 20 bp) is the direct signature of cooperative B3-domain binding to tandem RY elements.

The cross-motif case repeats the same calculation, this time measuring the distance from each RY hit to every hit of every other motif in the same peak. A pile-up at short distances would point to a TF whose binding co-localises with RY, whereas a flat distribution would indicate only incidental co-occurrence, driven by nothing more than both motifs being common across the peak set.

```
ry.by.seq <- split(ry.hits$start, ry.hits$sequence.i)
other.motifs <- setdiff(unique(hits$motif), "RY")
cross.dists <- do.call(rbind, lapply(other.motifs, function(om) {
  om.hits   <- hits[hits$motif == om, ]
  om.by.seq <- split(om.hits$start, om.hits$sequence.i)
  shared    <- intersect(names(ry.by.seq), names(om.by.seq))
  d <- unlist(lapply(shared, function(s)
    abs(as.integer(outer(ry.by.seq[[s]], om.by.seq[[s]], "-")))))
  if (!length(d)) return(NULL)
  data.frame(other = om, distance = d, stringsAsFactors = FALSE)
}))
ggplot2::ggplot(cross.dists, ggplot2::aes(x = .data$distance)) +
  ggplot2::geom_histogram(binwidth = 10, fill = "black", colour = NA) +
  ggplot2::geom_vline(xintercept = 50, linetype = "dashed",
                      colour = "grey50") +
  ggplot2::facet_wrap(~ other, scales = "free_y") +
  ggplot2::labs(x = "RY-to-other distance (bp)", y = "Count") +
  ggplot2::theme_bw() +
  ggplot2::theme(panel.grid         = ggplot2::element_blank(),
                 panel.border       = ggplot2::element_blank(),
                 strip.background   = ggplot2::element_rect(fill = NA,
                                                            colour = NA),
                 axis.line.x.bottom = ggplot2::element_line(colour = "black"),
                 axis.line.y.left   = ggplot2::element_line(colour = "black"))
```



None of the cross-motif panels shows the dramatic lone 0-5 bp spike that the (RY, RY) self-pair did. The AT-rich and GA-repeat panels do show a slightly raised abundance within the 50 bp window (which might hint at some relationship between the RY motifs and structural elements), though they also carry plenty of hits well outside it. The G_box and bZIP panels, by comparison, do show a concentration of hits at short distances, which hints at genuine colocalisation between RY and the bZIP-family motifs; this is biologically quite plausible for seed-maturation regulatory regions, where RY and G-box elements are known to co-occur. The overall impression, then, is that RY's strongest signal is its own self-clustering, with only a secondary and much milder bZIP / G-box neighbourship alongside it.

10 Summary table

The final step is to pull everything together into one compact ranked summary, with a single row per discovered motif that gathers up the per-motif results (its best JASPAR match, its enrichment q, its positional q) and adds a "best partner" (excluding self) column drawn from the co-occurrence table.

```
disc.names <- new.names

best.match <- vapply(disc.names, function(nm) {
  i <- which(cmp$subject == nm)
  if (!length(i)) NA_character_ else cmp$target[i[1]]
}, character(1))

enrich.q <- vapply(disc.names, function(nm) {
  i <- which(enr$motif == nm)
  if (!length(i)) NA_real_ else enr$qvalue[i[1]]
}, numeric(1))
```

```
pos.q <- vapply(disc.names, function(nm) {
  i <- which(pk$motif == nm)
  if (!length(i)) NA_real_ else pk$qvalue[i[1]]
}, numeric(1))

best.partner <- vapply(disc.names, function(nm) {
  ix <- which((co$motif_a == nm | co$motif_b == nm) &
              co$motif_a != co$motif_b)
  if (!length(ix)) return(NA_character_)
  ix <- ix[which.min(co$qvalue[ix])]
  if (co$motif_a[ix] == nm) co$motif_b[ix] else co$motif_a[ix]
}, character(1))

summary.tbl <- data.frame(
  motif         = disc.names,
  jaspar.match  = best.match,
  enrich.q      = signif(enrich.q, 3),
  positional.q  = signif(pos.q, 3),
  best.partner  = best.partner,
  stringsAsFactors = FALSE
)
rownames(summary.tbl) <- NULL
print(summary.tbl, row.names = FALSE)
#>      motif jaspar.match  enrich.q positional.q best.partner
#>         RY         ABI3 2.73e-149     1.44e-34    AT_rich.2
#>       bZIP         SPL7  5.21e-73     1.90e-05    AT_rich.2
#>      G_box         ABF3  6.23e-73           NA         bZIP
#>    AT_rich         CDF5  2.14e-25           NA    AT_rich.1
#>  GA_repeat         BPC1  1.24e-22           NA      AT_rich
#>    AT_rich         CDF5  2.14e-25           NA    AT_rich.1
#>    AT_rich         CDF5  2.14e-25           NA    AT_rich.1
```

Taking the combined evidence together, the analysis has clearly recovered the dominant VAL1 RY motif, along with a couple of secondary motifs that may well be binding sites for co-regulating TFs. The remaining repetitive motifs, although they are significantly enriched, fail the positional test, and are most likely incidental structural features of the kinds of sequences that VAL1 tends to bind.

11 Session info

```
sessionInfo()
#> R version 4.6.1 (2026-06-24)
#> Platform: x86_64-pc-linux-gnu
#> Running under: Ubuntu 24.04.5 LTS
#>
#> Matrix products: default
#> BLAS:   /home/biocbuild/bbs-3.24-bioc/R/lib/libRblas.so
#> LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.12.0  LAPACK version 3.12.0
#>
#> locale:
#>  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
#>  [3] LC_TIME=en_GB              LC_COLLATE=C
#>  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
```

```
#>  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
#>  [9] LC_ADDRESS=C               LC_TELEPHONE=C
#> [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
#>
#> time zone: America/New_York
#> tzcode source: system (glibc)
#>
#> attached base packages:
#> [1] stats4    stats     graphics  grDevices utils     datasets  methods
#> [8] base
#>
#> other attached packages:
#>  [1] MotifDb_1.55.0
#>  [2] BSgenome.Athaliana.TAIR.TAIR9_1.3.1000
#>  [3] BSgenome_1.81.1
#>  [4] BiocIO_1.23.3
#>  [5] rtracklayer_1.73.0
#>  [6] GenomeInfoDb_1.49.2
#>  [7] GenomicRanges_1.65.4
#>  [8] Biostrings_2.81.9
#>  [9] Seqinfo_1.3.2
#> [10] XVector_0.53.0
#> [11] IRanges_2.47.5
#> [12] S4Vectors_0.51.10
#> [13] BiocGenerics_0.59.13
#> [14] generics_0.1.4
#> [15] universalmotif_1.31.65
#>
#> loaded via a namespace (and not attached):
#>  [1] gtable_0.3.6             rjson_0.2.23             xfun_0.62
#>  [4] ggplot2_4.0.3            vctrs_0.7.3              tools_4.6.1
#>  [7] bitops_1.1-0             curl_8.1.0               parallel_4.6.1
#> [10] tibble_3.3.1             pkgconfig_2.0.3          BiocBaseUtils_1.15.1
#> [13] data.table_1.18.6.1      RColorBrewer_1.1-3       S7_0.2.2
#> [16] cigarillo_1.3.1          lifecycle_1.0.5          compiler_4.6.1
#> [19] farver_2.1.2             Rsamtools_2.29.0         tinytex_0.61
#> [22] codetools_0.2-20         htmltools_0.5.9          RCurl_1.98-1.20
#> [25] yaml_2.3.12              pillar_1.11.1            crayon_1.5.3
#> [28] MASS_7.3-66              BiocParallel_1.47.0      tidyselect_1.2.1
#> [31] digest_0.6.39            dplyr_1.2.1              restfulr_0.0.17
#> [34] bookdown_0.48            labeling_0.4.3           fastmap_1.2.0
#> [37] grid_4.6.1               cli_3.6.6                magrittr_2.0.5
#> [40] dichromat_2.0-1          XML_3.99-0.25            withr_3.0.3
#> [43] scales_1.4.0             UCSC.utils_1.9.0         rmarkdown_2.32
#> [46] httr_1.4.9               matrixStats_1.5.0        otel_0.3.0
#> [49] evaluate_1.0.5           knitr_1.52               rlang_1.3.0
#> [52] Rcpp_1.1.2               glue_1.8.1               splitstackshape_1.4.8.1
#> [55] jsonlite_2.0.0           R6_2.6.1                 MatrixGenerics_1.25.0
#> [58] GenomicAlignments_1.49.2
```

References

Yuan, L., X. Song, L. Zhang, Y. Yu, Z. Liang, Y. Lei, J. Ruan, B. Tan, J. Liu, and C. Li. 2021. "The Transcriptional Repressors VAL1 and VAL2 Recruit PRC2 for Genome-Wide Polycomb Silencing in Arabidopsis." *Nucleic Acids Research* 49 (1): 98–113. https://doi.org/10.1093/nar/gkaa1129.