nearBynding Vignette

Veronica Busa

2025-02-05

Contents

Introduction

nearBynding is a package designed to discern annotated RNA structures proximal to protein binding. nearBynding allows users to annotate RNA structure contexts via CapR or input their own annotations in BED/bedGraph format. It accomodates protein binding information from CLIP-seq experiments as either aligned CLIP-seq reads or peak-called intervals. This vignette will walk you through:

```
* The external software necessary to support this pipeline
* Creating a concatenated transcriptome
* Extracting and folding RNA from the transcriptome via CapR
* Mapping protein-binding and RNA structure information onto a transcriptome
* Running StereoGene to identify RNA structure proximal to protein binding
* Visualizing binding results
* Determining the distance between binding contexts
```

Before running any of these examples, it is highly recommended that the user establishes a new empty directory and uses `setwd()` to make certain that all outputs are deposited there. Some of the functions below create multiple output files.

Installation

```
if(!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
BiocManager::install("nearBynding")
```

External Software Dependencies

Add all dependency directories to your PATH after installation.

bedtools

bedtools is available for installation here.

Installation instructions will vary by operating system.

CapR

Download the zip file from the github repository, unzip the file, and move it to a directory where you want to permanently store the function.

In the command line, access the folder where the unzipped file is stored.

```
cd CapR-master
make
./CapR
```

If installation is successful, the final line will output

```
Error: The number of argument is invalid.
```

StereoGene

Download the zip file from the github repository, unzip the file, and move it to a directory where you want to permanently store the function.

In the command line, access the folder where the unzipped file is stored.

```
cd stereogene-master
cd src
make
./stereogene -h
```

If installation is successful, the final line will output a menu of argument options.

1. Concatenate the Transcriptome

Although nearBynding is designed to support whole-genome analyses, we will exclusively be evaluating protein-coding genes of chromosomes 4 and 5 through this vignette.

First, a list of transcripts must be identified for analysis. A recommended criterium for selection is that the transcripts be expressed in the cell type used for CLIP-seq experiments. For this vignette, 50 random transcripts have been selected, and the 3'UTR structure of each transcript will be used for analysis, though any region of a transcript such as 5'UTR or CDS could be assessed instead.

This step creates a chain file that will be used to map the selected regions of transcripts end-to-end, excluding the intergenic regions and undesired transcripts that comprise the majority of the genome.

```
# load transcript list
load(system.file("extdata/transcript_list.Rda", package="nearBynding"))
# get GTF file
gtf<-system.file("extdata/Homo_sapiens.GRCh38.chr4&5.gtf",
                    package="nearBynding")

GenomeMappingToChainFile(genome_gtf = gtf,
                         out_chain_name = "test.chain",
                         RNA_fragment = "three_prime_utr",
                         transcript_list = transcript_list,
                         alignment = "hg38")
```

A file containing the sizes of each concatenated chromosome in the chain file will be required for downstream analysis.

```
getChainChrSize(chain = "test.chain",
                out_chr = "chr4and5_3UTR.size")
```

2. Fold RNA via CapR

In order to fold the 3'UTRs, the sequences must first be extracted. This is achieved with the following code:

```
ExtractTranscriptomeSequence(transcript_list = transcript_list,
                    ref_genome = "Homo_sapiens.GRCh38.dna.primary_assembly.fa",
                    genome_gtf = gtf,
                    RNA_fragment = "three_prime_utr",
                    exome_prefix = "chr4and5_3UTR")
```

The reference genome can be found through Ensembl, but for users who do not want to download that 3.2GB file for the sake of this vignette, the FASTA output of the above code is available via:

```
chr4and5_3UTR.fa <- system.file("extdata/chr4and5_3UTR.fa",
                                package="nearBynding")
```

These sequences can then be submitted to CapR for folding.

```
runCapR(in_file = chr4and5_3UTR.fa)
```

Warning: This step can take hours or even days depending on how many transcripts are submitted, how long the RNA fragments are, and the maximum distance between base-paired nucleotides submitted to the CapR algorithm.

3. Map to Transcriptome

The nearBynding pipeline can accomodate either a BAM file of aligned CLIP-seq reads or a BED file of peak intervals. BAM files must be sorted and converted to a bedGraph file first, whereas BED files can be read-in directly.

```
bam <- system.file("extdata/chr4and5.bam", package="nearBynding")
sorted_bam<-sortBam(bam, "chr4and5_sorted")

CleanBAMtoBG(in_bam = sorted_bam)
```

BAM file input

Map Protein Binding and RNA Structure to the Transcriptome BED or bedGraph files can then be mapped onto the concatenated transcriptome using the chain file created by `GenomeMappingToChainFile()`. This way, only the protein binding from transcriptomic regions of interest will be considered in the final protein binding analysis.

```
liftOverToExomicBG(input = "chr4and5_sorted.bedGraph",
                   chain = "test.chain",
                   chrom_size = "chr4and5_3UTR.size",
                   output_bg = "chr4and5_liftOver.bedGraph")
```

For BED file inputs, use the additional argument `format = "BED"`.

The RNA structure information from the CapR output also needs to mapped onto the concatenated transcriptome. There are six different binding contexts calculated by CapR – *stem*, *hairpin*, *multibranch*, *exterior*, *internal*, and *bulge*. `processCapRout()` parses the CapR output, converts it into six separate bedGraph files, and then performs `liftOverToExomic()` for all the files.

For this sake of this vignette, the CapR outfile is available:

```
processCapRout(CapR_outfile = system.file("extdata/chr4and5_3UTR.out",
                                          package="nearBynding"),
               chain = "test.chain",
               output_prefix = "chr4and5_3UTR",
               chrom_size = "chr4and5_3UTR.size",
               genome_gtf = gtf,
               RNA_fragment = "three_prime_utr")
```

It is possible for users to input their own RNA structure information rather than using CapR. The information should be in BED file format and can be input into `liftOverToExomicBG()` to map the RNA structure data to the same transcriptome as the protein binding data.

4. Calculate Cross-correlation via StereoGene

This is the process that directly answers the question, "What does RNA structure look like around where the protein is binding?" StereoGene is used to calculate the cross-correlation between RNA structure and protein binding in order to visualize the RNA structure landscape surrounding protein binding.

If CapR is used to determine RNA structure, `runStereogeneOnCapR()` initiates StereoGene for a given protein against all CapR-generated RNA structure contexts.

For the sake of this vignette, use `outfiles()` to pull the StereoGene output files to your local directory if you do not want to run StereoGene.

```
runStereogeneOnCapR(protein_file = "chr4and5_liftOver.bedGraph",
                    chrom_size = "chr4and5_3UTR.size",
                    name_config = "chr4and5_3UTR.cfg",
                    input_prefix = "chr4and5_3UTR")
```

If external RNA structure data is being tested, `runStereogene()` initiates analysis for a given protein and a single RNA structure context.

Note: The input track file order matters! The correct order is:

1) RNA structure
2) protein binding

Otherwise, data visualization will be inverted and all downstream analysis will be backwards.

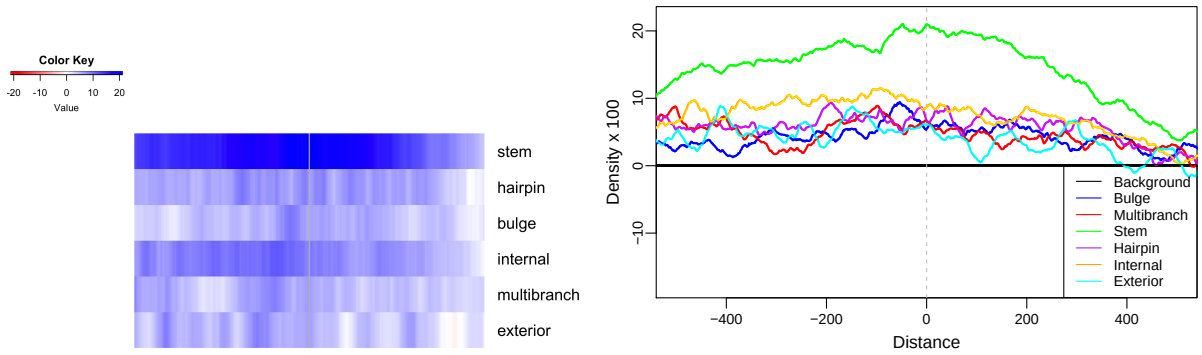
```
runStereogene(track_files = c("chr4and5_3UTR_stem_liftOver.bedGraph",
                              "chr4and5_liftOver.bedGraph"),
              name_config = "chr4and5_3UTR.cfg")
```

5. Visualize Results

The cross-correlation output of StereoGene can be visualized as either a heatmap or a line plot. Examples of both are below.

For CapR-derived RNA structure, all six contexts can be viewed simultaneously.

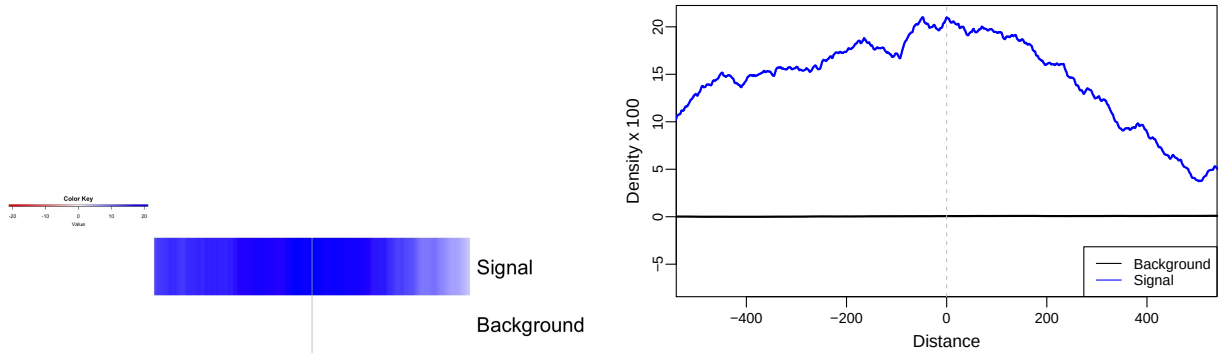
```
visualizeCapRStereogene(CapR_prefix = "chr4and5_3UTR",
                        protein_file = "chr4and5_liftOver",
                        heatmap = TRUE,
                        out_file = "all_contexts_heatmap",
                        x_lim = c(-500, 500))
visualizeCapRStereogene(CapR_prefix = "chr4and5_3UTR",
                        protein_file = "chr4and5_liftOver",
                        x_lim = c(-500, 500),
                        out_file = "all_contexts_line",
                        y_lim = c(-18, 22))
```



Warning: This step may take up to an hour for a full transcriptome.

Alternatively, a single context can be viewed at a time.

```
visualizeStereogene(context_file = "chr4and5_3UTR_stem_liftOver",
                    protein_file = "chr4and5_liftOver",
                    out_file = "stem_line",
                    x_lim = c(-500, 500))
visualizeStereogene(context_file = "chr4and5_3UTR_stem_liftOver",
                    protein_file = "chr4and5_liftOver",
                    heatmap = TRUE,
                    out_file = "stem_heatmap",
                    x_lim = c(-500, 500))
```



Although this specific, limited example does not provide a particularly visually stimulating image, larger data sets (which provide many more StereoGene windows) result in narrower peaks and less noise.

6. Calculate Distance

In order to determine the similarity of two binding contexts, we can calculate the Wasserstein distance between curves. A small value suggests two binding contexts are very similar, whereas larger values suggest substantial differences.

For example, calculate the distance between the stem and hairpin contexts visualized above.

```
bindingContextDistance(RNA_context = "chr4and5_3UTR_stem_liftOver",
                       protein_file = "chr4and5_liftOver",
                       RNA_context_2 = "chr4and5_3UTR_hairpin_liftOver")
#> [1] 11.75237
```

Now compare it to the distance between internal and hairpin contexts.

```
bindingContextDistance(RNA_context = "chr4and5_3UTR_internal_liftOver",
                       protein_file = "chr4and5_liftOver",
                       RNA_context_2 = "chr4and5_3UTR_hairpin_liftOver")
#> [1] 1.788653
```

We can see that the stem context is less similar to the hairpin context than the internal context, and this is reflected in the calculated distances.

Questions? Comments? Please email Veronica Busa at vbusa1@jhmi.edu

```
sessionInfo()
#> R Under development (unstable) (2025-01-22 r87618)
#> Platform: x86_64-apple-darwin20
#> Running under: macOS Monterey 12.7.6
#>
```

```
#> Matrix products: default
#> BLAS:   /Library/Frameworks/R.framework/Versions/4.5-x86_64/Resources/lib/libRblas.0.dylib
#> LAPACK: /Library/Frameworks/R.framework/Versions/4.5-x86_64/Resources/lib/libRlapack.dylib;  LAPACK
#>
#> locale:
#> [1] C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
#>
#> time zone: America/New_York
#> tzcode source: internal
#>
#> attached base packages:
#> [1] stats4    stats     graphics  grDevices utils     datasets  methods
#> [8] base
#>
#> other attached packages:
#>  [1] Rsamtools_2.23.1     Biostrings_2.75.3    XVector_0.47.2
#>  [4] GenomicRanges_1.59.1 GenomeInfoDb_1.43.4  IRanges_2.41.2
#>  [7] S4Vectors_0.45.2     BiocGenerics_0.53.6  generics_0.1.3
#> [10] nearBynding_1.17.0
#>
#> loaded via a namespace (and not attached):
#>  [1] tidyselect_1.2.1
#>  [2] dplyr_1.1.4
#>  [3] blob_1.2.4
#>  [4] R.utils_2.12.3
#>  [5] bitops_1.0-9
#>  [6] fastmap_1.2.0
#>  [7] RCurl_1.98-1.16
#>  [8] GenomicAlignments_1.43.0
#>  [9] XML_3.99-0.18
#> [10] digest_0.6.37
#> [11] transport_0.15-4
#> [12] lifecycle_1.0.4
#> [13] plyranges_1.27.0
#> [14] KEGGREST_1.47.0
#> [15] TxDb.Hsapiens.UCSC.hg38.knownGene_3.20.0
#> [16] RSQLite_2.3.9
#> [17] magrittr_2.0.3
#> [18] compiler_4.5.0
#> [19] rlang_1.1.5
#> [20] tools_4.5.0
#> [21] yaml_2.3.10
#> [22] data.table_1.16.4
#> [23] rtracklayer_1.67.0
#> [24] knitr_1.49
#> [25] S4Arrays_1.7.2
#> [26] bit_4.5.0.1
#> [27] curl_6.2.0
#> [28] DelayedArray_0.33.5
#> [29] abind_1.4-8
#> [30] BiocParallel_1.41.0
#> [31] KernSmooth_2.23-26
#> [32] R.oo_1.27.0
```

```
#> [33] grid_4.5.0
#> [34] caTools_1.18.3
#> [35] colorspace_2.1-1
#> [36] ggplot2_3.5.1
#> [37] scales_1.3.0
#> [38] gtools_3.9.5
#> [39] SummarizedExperiment_1.37.0
#> [40] cli_3.6.3
#> [41] rmarkdown_2.29
#> [42] crayon_1.5.3
#> [43] httr_1.4.7
#> [44] rjson_0.2.23
#> [45] DBI_1.2.3
#> [46] cachem_1.1.0
#> [47] parallel_4.5.0
#> [48] AnnotationDbi_1.69.0
#> [49] restfulr_0.0.15
#> [50] matrixStats_1.5.0
#> [51] vctrs_0.6.5
#> [52] Matrix_1.7-2
#> [53] jsonlite_1.8.9
#> [54] bit64_4.6.0-1
#> [55] GenomicFeatures_1.59.1
#> [56] glue_1.8.0
#> [57] codetools_0.2-20
#> [58] gtable_0.3.6
#> [59] BiocIO_1.17.1
#> [60] UCSC.utils_1.3.1
#> [61] munsell_0.5.1
#> [62] tibble_3.2.1
#> [63] pillar_1.10.1
#> [64] htmltools_0.5.8.1
#> [65] gplots_3.2.0
#> [66] GenomeInfoDbData_1.2.13
#> [67] R6_2.5.1
#> [68] evaluate_1.0.3
#> [69] lattice_0.22-6
#> [70] Biobase_2.67.0
#> [71] R.methodsS3_1.8.2
#> [72] png_0.1-8
#> [73] memoise_2.0.1
#> [74] TxDb.Hsapiens.UCSC.hg19.knownGene_3.2.2
#> [75] Rcpp_1.0.14
#> [76] SparseArray_1.7.5
#> [77] xfun_0.50
#> [78] MatrixGenerics_1.19.1
#> [79] pkgconfig_2.0.3
```