Using crisprDesign to design gRNAs for CRISPRbe

Jean-Philippe Fortin, Luke Hoberecht

Introduction

In this tutorial, We illustrate the CRISPR base editing (CRISPRbe) functionalities of `crisprDesign` by designing and characterizing gRNAs targeting the human gene KRAS using the cytidine base editor BE4max (Koblan et al. 2018).

Installation

See the Installation tutorial to learn how to install the packages necessary for this tutorial: `crisprDesign`, `crisprDesignData`

Terminology

See the CRISPRko design tutorial to get familiar with the terminology used throughout this tutorial.

CRISPR base editing with BE4max

Loading packages

We first load the necessary packages for this tutorial:

```
library(crisprBase)
library(crisprDesign)
library(crisprDesignData)
library(BSgenome.Hsapiens.UCSC.hg38)
```

Creating the GuideSet

We first load the BE4max `BaseEditor` object from the `crisprBase` package:

```
data(BE4max, package="crisprBase")
BE4max
```

```
## Class: BaseEditor
##   CRISPR Nuclease name: SpCas9
##       Target type: DNA
##       Metadata: list of length 2
##       PAMs: NGG, NAG, NGA
##       Weights: 1, 0.2593, 0.0694
##       Spacer length: 20
##       PAM side: 3prime
##         Distance from PAM: 0
##       Prototype protospacers: 5'--SSSSSSSSSSSSSSSSSSS[NGG]--3', 5'--SSSSSSSSSSSSSSSSSSS[NAG]--3', 5
```

```
##   Base editor name: BE4max
##       Editing strand: original
##       Maximum editing weight: C2T at position -15
```

The editing probabilities of the base editor BE4max are stored in a matrix where rows correspond to the different nucleotide substitutions, and columns correspond to the genomic coordinate relative to the PAM site. The `editingWeights` function from `crisprBase` retrieves those probabilities. One can see that C to T editing is optimal around 15 nucleotides upstream of the PAM site for the BE4max base editor:

```
crisprBase::editingWeights(BE4max)["C2T",]
```

```
##   -36   -35   -34   -33   -32   -31   -30   -29   -28   -27   -26   -25   -24
## 0.007 0.007 0.008 0.018 0.010 0.020 0.014 0.012 0.023 0.013 0.024 0.022 0.034
##   -23   -22   -21   -20   -19   -18   -17   -16   -15   -14   -13   -12   -11
## 0.022 0.021 0.035 0.058 0.162 0.318 0.632 0.903 1.000 0.870 0.620 0.314 0.163
##   -10    -9    -8    -7    -6    -5    -4    -3    -2    -1
## 0.100 0.056 0.033 0.019 0.018 0.024 0.017 0.005 0.002 0.001
```

Let's create the `GuideSet` containing gRNAs targeting KRAS.

We first load the data containing gene regions for the human genome from `crisprDesignData`:

```
data("txdb_human", package="crisprDesignData")
```

For more information on `txdb_human` and how to create similar gene annotation objects, see the Building a gene annotation object tutorial.

We will also load the `BSgenome` package containing DNA sequences for the hg38 genome:

```
library(BSgenome.Hsapiens.UCSC.hg38)
```

We retrive the genomic coordinates of the KRAS CDS

```
gr <- queryTxObject(txObject=txdb_human,
                    featureType="cds",
                    queryColumn="gene_symbol",
                    queryValue="KRAS")
```

and design all possigle gRNAs using the function `findSpacers`:

```
bsgenome <- BSgenome.Hsapiens.UCSC.hg38
gs <- findSpacers(gr,
                  bsgenome=bsgenome,
                  crisprNuclease=BE4max)
```

Annotating the GuideSet

Next, we annotate our candidate gRNAs to assess quality. There are several functions in `crisprDesign` that provide annotation for features that are nonspecific to CRISPRbe, for which we refer the reader to the CRISPRko design with Cas9 tutorial for more information. The sections below will cover annotation functions that are of particular interest to, or deserve extra care for CRISPRbe applications.

Adding edited alleles

The function `addEditedAlleles` finds, characterizes, and scores predicted edited alleles for each gRNA and a chosen transcript. It requires a transcript-specific annotation that can be obtained with the `getTxInfoDataFrame` function. Here, we perform the analysis using the primary isoform of KRAS (Ensembl transcript ID: ENST00000311936).

We first get the transcript table for our transcript

```
txid <- "ENST00000311936"
txTable <- getTxInfoDataFrame(tx_id=txid,
                              txObject=txdb_human,
                              bsgenome=bsgenome)
head(txTable)
```

```
## DataFrame with 6 rows and 10 columns
##           chr       pos         nuc          aa aa_number      exon  pos_plot
##   <character> <numeric> <character> <character> <integer> <integer> <integer>
## 1       chr12  25250929           C          NA        NA         1        31
## 2       chr12  25250928           T          NA        NA         1        32
## 3       chr12  25250927           A          NA        NA         1        33
## 4       chr12  25250926           G          NA        NA         1        34
## 5       chr12  25250925           G          NA        NA         1        35
## 6       chr12  25250924           C          NA        NA         1        36
##    pos_mrna   pos_cds      region
##   <integer> <integer> <character>
## 1         1        NA        5UTR
## 2         2        NA        5UTR
## 3         3        NA        5UTR
## 4         4        NA        5UTR
## 5         5        NA        5UTR
## 6         6        NA        5UTR
```

and then add the edited alleles annotation to the `GuideSet`:

```
editingWindow <- c(-20,-8)
gs <- addEditedAlleles(gs,
                       baseEditor=BE4max,
                       txTable=txTable,
                       editingWindow=editingWindow)
```

```
## [addEditedAlleles] Obtaining edited alleles at each gRNA target site.
```

```
## [addEditedAlleles] Adding functional consequences to alleles.
```

The `editingWindow` argument specifies the window of editing that we are interested in. When not provided, it uses the default window provided in the `BaseEditor` object. Note that providing large windows can exponentially increase computing time as the number of possible alleles grows exponentially.

Let's retrieve the edited alleles for the first gRNA:

```
alleles <- editedAlleles(gs)[[1]]
```

We get a `DataFrame` object with useful metadata:

```
metadata(alleles)
```

```
## $wildtypeAllele
##        spacer_1
## "AAAGAAAAGATGA"
##
## $start
## [1] 25209851
##
## $end
## [1] 25209863
##
```

```
## $chr
## [1] "chr12"
##
## $strand
## [1] "-"
##
## $editingWindow
## [1] -20  -8
##
## $wildtypeAmino
## [1] "SMMMKKKEEEKKK"
```

The `wildtypeAllele` reports the unedited nucleotide sequence of the region specified by the editing window (with respect to the gRNA PAM site). It is always reported from the 5' to 3' direction on the strand corresponding to the gRNA strand. The `start` and `end` fields specify the corresponding coordinates on the transcript.

Let's look at the edited alleles:

```
head(alleles)
```

```
## DataFrame with 6 rows and 4 columns
##              seq       score     variant            aa
##   <DNAStringSet>   <numeric> <character>   <character>
## 1  AAAGAAAACATGA 0.000922509    missense SMMMNNNEEEKKK
## 2  AAAAAAAAAATAA 0.000000000    missense SIIIKKKKKKKKK
## 3  AAACAAAAAATAA 0.000000000    missense SIIIKKKQQQKKK
## 4  AAAGAAAAAATAA 0.000000000    missense SIIIKKKEEEKKK
## 5  AAAAAAAACATAA 0.000000000    missense SIIINNNKKKKKK
## 6  AAACAAAACATAA 0.000000000    missense SIIINNNQQQKKK
```

The `DataFrame` is ordered by descending values in the `score` column. This `score` represents the likelihood of the edited allele to occur relative to all possible edited alleles, and is calculated using the editing weights stored in the `BE4max` object. The `seq` column represents the edited nucleotide sequences. As with the `wildtypeAllele` in the metadata, they are always reported from the 5' to 3' direction on the strand corresponding to the gRNA strand.

The `variant` column describes the functional consequence of the editing event (silent, nonsense or missense mutation). If an edited allele results in multiple editing events, as can happen when multiple bases are edited, the most consequential mutation (nonsense over missense, missense over silent) is reported. Finally, the `aa` column reports the resulting edited amino acid sequence, with each single letter code mapping to its corresponding nucleotide (`*` for termination).

Note that `addEditedAlleles` also appended several gRNA-level aggregate scores to the `GuideSet` object:

```
head(gs)
```

```
## GuideSet object with 6 ranges and 11 metadata columns:
##            seqnames    ranges strand |          protospacer            pam
##               <Rle> <IRanges>  <Rle> |       <DNAStringSet> <DNAStringSet>
##   spacer_1    chr12  25209843      - | AAAGAAAAGATGAGCAAAGA            TGG
##   spacer_2    chr12  25209843      - | AAAGAAAAGATGAGCAAAGA            TGG
##   spacer_3    chr12  25209896      + | TTCTCGAACTAATGTATAGA            AGG
##   spacer_4    chr12  25209896      + | TTCTCGAACTAATGTATAGA            AGG
##   spacer_5    chr12  25215438      - | AAATGCATTATAATGTAATC            TGG
##   spacer_6    chr12  25215477      - | AGCAAAGAAGAAAAGACTCC            TGG
##             pam_site  cut_site      region
##            <numeric> <numeric> <character>
```

```
##   spacer_1  25209843  25209846    region_8
##   spacer_2  25209843  25209846   region_10
##   spacer_3  25209896  25209893    region_8
##   spacer_4  25209896  25209893   region_10
##   spacer_5  25215438  25215441    region_4
##   spacer_6  25215477  25215480    region_4
##
##
##   spacer_1 AAAGAAAACATGA:0.000922509:missense:...,AAAAAAAAAATAA:0.000000000:missense:...,AAACAAAAAAT
##   spacer_2 AAAGAAAACATGA:0.000922509:missense:...,AAAAAAAAAATAA:0.000000000:missense:...,AAACAAAAAAT
##   spacer_3          TTCTTGAACTAAT:0.392378:missense:...,TTTTTGAACTAAT:0.167936:missense:...,TTCTTGAAT
##   spacer_4          TTCTTGAACTAAT:0.392378:missense:...,TTTTTGAACTAAT:0.167936:missense:...,TTCTTGAAT
##   spacer_5        AAATGTATTATAA:0.9199562:not_targeting,AAATGGATTATAA:0.0496776:not_targeting,AAATGAA
##   spacer_6     AGTAAAGAAGAAA:0.29281668:not_targeting,AGGAAAGAAGAAA:0.01197049:not_targeting,AGAAAAG
##            score_missense score_nonsense score_silent    maxVariant
##                 <numeric>      <numeric>    <numeric>   <character>
##   spacer_1    0.000922509     0.00000000            0      missense
##   spacer_2    0.000922509     0.00000000            0      missense
##   spacer_3    0.944195218     0.00931016            0      missense
##   spacer_4    0.944195218     0.00931016            0      missense
##   spacer_5    0.000000000     0.00000000            0 not_targeting
##   spacer_6    0.000000000     0.00000000            0 not_targeting
##            maxVariantScore
##                  <numeric>
##   spacer_1     0.000922509
##   spacer_2     0.000922509
##   spacer_3     0.944195218
##   spacer_4     0.944195218
##   spacer_5              NA
##   spacer_6              NA
##   -------
##   seqinfo: 640 sequences (1 circular) from hg38 genome
##   crisprNuclease: SpCas9
```

The `score_missense`, `score_nonsense` and `score_silent` columns report aggregated scores for each mutation type. They are calculated by summing all scores of a given mutation type across the set of edited alleles for a given gRNA. The `maxVariant` column indicates the most probable mutation type for the given gRNA based on the maximum aggregated score, which is stored in `maxVariantScore`. In our example, the highest score for `spacer_4` is `score_nonsense`, and so `maxVariant` is set to `nonsense`.

Session Info

```
sessionInfo()
```

```
## R version 4.2.1 (2022-06-23)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS Catalina 10.15.7
##
## 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] stats4    stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
##  [1] BSgenome.Hsapiens.UCSC.hg38_1.4.4 BSgenome_1.65.2
##  [3] rtracklayer_1.57.0                Biostrings_2.65.2
##  [5] XVector_0.37.0                    GenomicRanges_1.49.1
##  [7] GenomeInfoDb_1.33.5               IRanges_2.31.2
##  [9] S4Vectors_0.35.1                  crisprDesignData_0.99.17
## [11] crisprDesign_0.99.133             crisprScore_1.1.14
## [13] crisprScoreData_1.1.3             ExperimentHub_2.5.0
## [15] AnnotationHub_3.5.0               BiocFileCache_2.5.0
## [17] dbplyr_2.2.1                      BiocGenerics_0.43.1
## [19] crisprBowtie_1.1.1                crisprBase_1.1.5
## [21] crisprVerse_0.99.8                rmarkdown_2.15.2
##
## loaded via a namespace (and not attached):
##  [1] rjson_0.2.21                  ellipsis_0.3.2
##  [3] Rbowtie_1.37.0                bit64_4.0.5
##  [5] lubridate_1.8.0               interactiveDisplayBase_1.35.0
##  [7] AnnotationDbi_1.59.1          fansi_1.0.3
##  [9] xml2_1.3.3                    codetools_0.2-18
## [11] cachem_1.0.6                  knitr_1.40
## [13] jsonlite_1.8.0                Rsamtools_2.13.4
## [15] png_0.1-7                     shiny_1.7.2
## [17] BiocManager_1.30.18           readr_2.1.2
## [19] compiler_4.2.1                httr_1.4.4
## [21] basilisk_1.9.2                assertthat_0.2.1
## [23] Matrix_1.4-1                  fastmap_1.1.0
## [25] cli_3.3.0                     later_1.3.0
## [27] htmltools_0.5.3               prettyunits_1.1.1
## [29] tools_4.2.1                   glue_1.6.2
## [31] GenomeInfoDbData_1.2.8        dplyr_1.0.9
## [33] rappdirs_0.3.3                tinytex_0.41
## [35] Rcpp_1.0.9                    Biobase_2.57.1
## [37] vctrs_0.4.1                   crisprBwa_1.1.3
## [39] xfun_0.32                     stringr_1.4.1
## [41] mime_0.12                     lifecycle_1.0.1
## [43] restfulr_0.0.15               XML_3.99-0.10
## [45] zlibbioc_1.43.0               basilisk.utils_1.9.1
## [47] vroom_1.5.7                   VariantAnnotation_1.43.3
## [49] hms_1.1.2                     promises_1.2.0.1
## [51] MatrixGenerics_1.9.1          parallel_4.2.1
## [53] SummarizedExperiment_1.27.1   RMariaDB_1.2.2
## [55] yaml_2.3.5                    curl_4.3.2
## [57] memoise_2.0.1                 reticulate_1.25
## [59] biomaRt_2.53.2                stringi_1.7.8
## [61] RSQLite_2.2.16                BiocVersion_3.16.0
## [63] highr_0.9                     BiocIO_1.7.1
## [65] randomForest_4.7-1.1          GenomicFeatures_1.49.6
## [67] filelock_1.0.2                BiocParallel_1.31.12
## [69] rlang_1.0.4                   pkgconfig_2.0.3
```

```
## [71] matrixStats_0.62.0          bitops_1.0-7
## [73] evaluate_0.16               lattice_0.20-45
## [75] purrr_0.3.4                 GenomicAlignments_1.33.1
## [77] bit_4.0.4                   tidyselect_1.1.2
## [79] magrittr_2.0.3              R6_2.5.1
## [81] generics_0.1.3              DelayedArray_0.23.1
## [83] DBI_1.1.3                   pillar_1.8.1
## [85] KEGGREST_1.37.3             RCurl_1.98-1.8
## [87] tibble_3.1.8                dir.expiry_1.5.0
## [89] crayon_1.5.1                utf8_1.2.2
## [91] tzdb_0.3.0                  progress_1.2.2
## [93] grid_4.2.1                  blob_1.2.3
## [95] digest_0.6.29               xtable_1.8-4
## [97] httpuv_1.6.5                Rbwa_1.1.0
```

References

Koblan, Luke W, Jordan L Doman, Christopher Wilson, Jonathan M Levy, Tristan Tay, Gregory A Newby, Juan Pablo Maianti, Aditya Raguram, and David R Liu. 2018. "Improving Cytidine and Adenine Base Editors by Expression Optimization and Ancestral Reconstruction." *Nature Biotechnology* 36 (9): 843–46.