Package 'TAPseq'

May 18, 2024

Type Package

Title Targeted scRNA-seq primer design for TAP-seq

Version 1.16.0

Description Design primers for targeted single-cell RNA-seq used by TAP-seq. Create sequence templates for target gene panels and design gene-specific primers using Primer3. Potential off-
targets can be estimated with BLAST. Requires working installations of Primer3 and BLASTn.

URL https://github.com/argschwind/TAPseq

BugRepots https://github.com/argschwind/TAPseq/issues

biocViews SingleCell, Sequencing, Technology, CRISPR, PooledScreens

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.2.3

VignetteBuilder knitr

Depends R (>= 4.0.0)

Imports methods, GenomicAlignments, GenomicRanges, IRanges, BiocGenerics, S4Vectors (>= 0.20.1), GenomeInfoDb, BSgenome, GenomicFeatures, Biostrings, dplyr, tidyr, BiocParallel

Suggests testthat, BSgenome.Hsapiens.UCSC.hg38, knitr, rmarkdown, ggplot2, Seurat, glmnet, cowplot, Matrix, rtracklayer, BiocStyle

SystemRequirements Primer3 (>= 2.5.0), BLAST+ (>=2.6.0)

git_url https://git.bioconductor.org/packages/TAPseq

git_branch RELEASE_3_19

git_last_commit e3d22d8

git_last_commit_date 2024-04-30

Repository Bioconductor 3.19

Date/Publication 2024-05-17

Author Andreas R. Gschwind [aut, cre]
(<https://orcid.org/0000-0002-0769-6907>),
Lars Velten [aut] (<https://orcid.org/0000-0002-1233-5874>),
Lars M. Steinmetz [aut]

Maintainer Andreas R. Gschwind <andreas.gschwind@stanford.edu>

Contents

`accessors` *Accessors for TsIO objects*

Description

A set of functions for getting/setting/modifying the data stored in `TsIO` or `TsIOList` class objects.

Usage

```
sequence_id(x)

sequence_id(x) <- value

target_sequence(x)

target_sequence(x) <- value

beads_oligo(x)

beads_oligo(x) <- value

reverse_primer(x)

reverse_primer(x) <- value

target_annot(x)

target_annot(x) <- value

product_size_range(x)

product_size_range(x) <- value

primer_num_return(x)

primer_num_return(x) <- value

min_primer_region(x)

min_primer_region(x) <- value

primer_opt_tm(x)

primer_opt_tm(x) <- value

primer_min_tm(x)

primer_min_tm(x) <- value

primer_max_tm(x)

primer_max_tm(x) <- value

sequence_template(x)

tapseq_primers(x)
```

```
pcr_products(x)
```

Arguments

x	A TsIO or TsIOList class object.
value	A valid value to assign to the chosen slot.

Value

Returns the stored value(s) of a slot, or sets a new value

Examples

```
# chr11 primers example data
data("chr11_primers")

# slot values of TsIO objects can be accessed using accessor functions
tsio <- chr11_primers[[1]]
sequence_id(tsio)
sequence_id(tsio) <- "Gene1"
sequence_id(tsio)

# some slots can only be obtained, but not set as filling these is part of the TAPseq workflow
tapseq_primers(tsio)
pcr_products(tsio)

# sequence templates can be created
sequence_template(tsio)

# values of TsIOList object slots can be extracted as well, but not set
tsio_list <- chr11_primers[1:2]
sequence_id(tsio_list)
target_sequence(tsio_list)
target_annot(tsio_list)
tapseq_primers(tsio_list)
pcr_products(tsio_list)
sequence_template(tsio_list)
```

---

bone_marrow_genex *Mouse bone marrow 10x data*

Description

Subset of a 10x mouse bone marrow dataset taken from Baccin et al., 2019 (https://www.nature.com/articles/s41556-019-0439-6). Contains gene expression and cell type data for 362 cells.

Usage

```
bone_marrow_genex
```

Format

object of Seurat class.

---

checkPrimers *Check primers for complementarity*

Description

Check a TAP-seq primer set, i.e. outer or inner primers for a target gene panel, for potential complementarity issues when multiplexing. Uses Primer3's check_primers functionality.

Usage

```
checkPrimers(
  object,
  primer_opt_tm = 63,
  primer_min_tm = 59,
  primer_max_tm = 66,
  thermo_params_path = NA,
  primer3_core = getOption("TAPseq.primer3_core")
)

## S4 method for signature 'TsIO'
checkPrimers(
  object,
  primer_opt_tm = 63,
  primer_min_tm = 59,
  primer_max_tm = 66,
  thermo_params_path = NA,
  primer3_core = getOption("TAPseq.primer3_core")
)

## S4 method for signature 'TsIOList'
checkPrimers(
  object,
  primer_opt_tm = 63,
  primer_min_tm = 59,
  primer_max_tm = 66,
  thermo_params_path = NA,
  primer3_core = getOption("TAPseq.primer3_core")
)
```

Arguments

object	A TsIO or TsIOList object containing designed primers.

primer_opt_tm, primer_min_tm, primer_max_tm
: Optimal, minumum and maximum primer melting temperature. Should be the same values that were used when designing the primers.

thermo_params_path
: Optional path (character) to the primer3_config directory. Only required when using Primer3 < 2.5.0.

primer3_core
: Path (character) to the primer3_core executable. Usually this is inferred when loading/attaching the package.

Value

A data.frame with check_primers results.

Methods (by class)

- checkPrimers(TsIO): Check primers from TsIO objects.
- checkPrimers(TsIOList): Check primers from TsIOList objects.

See Also

http://primer3.org/manual.html for Primer3 manual.

Examples

```
library(ggplot2)

# chr11 primers example data
data("chr11_primers")

# pick best primers based on predicted off-targets for subset of all primers
best_primers <- pickPrimers(chr11_primers, n = 1, by = "off_targets")

# check for complementarity
## Not run:
comp <- checkPrimers(best_primers)

# plot complementarity scores for every pair. the lines indicate complementarity scores of 47,
# the default value applied by Primer3 to identify high complementarity primer pairs
ggplot(comp, aes(x = primer_pair_compl_any_th, y = primer_pair_compl_end_th)) +
  geom_hline(aes(yintercept = 47), colour = "darkgray", linetype = "dashed") +
  geom_vline(aes(xintercept = 47), colour = "darkgray", linetype = "dashed") +
  geom_point(alpha = 0.25) +
  theme_bw()

## End(Not run)
```

check_tool_installation *Check required tools*

Description

Check if required software is installed and return paths to executables (if found).

Usage

```
check_tool_installation()
```

Value

A `list` containing paths to required tools, if found, else NA.

chr11_genes *Chromosome 11 genes*

Description

GENCODE exon annotations of all protein-coding genes within a genomic region on human chromosome 11.

Usage

```
chr11_genes
```

Format

object of `GRanges` class.

chr11_polyA_sites *Chromosome 11 polyA sites*

Description

Example polyadenylation sites for expressed protein-coding genes within human chromosome 11 genomic region. This dataset was created using `inferPolyASites` on available K562 Drop-seq data. In a real-world application these sites would have to be pruned manually before further use.

Usage

```
chr11_polyA_sites
```

Format

object of `GRanges` class.

chr11_primers *Chromosome 11 primers*

Description

Example of a `TsIOList` object containing input and output for chromosome 11 genes primer design.

Usage

```
chr11_primers
```

Format

object of `TsIOList` class.

chr11_truncated_txs *Chromosome 11 truncated transcripts*

Description

Annotations of target gene transcripts within human chromosome 11 region that were truncated at inferred polyA sites using `truncateTxsPolyA`.

Usage

```
chr11_truncated_txs
```

Format

object of `GRangesList` class.

chr11_truncated_txs_seq *Chromosome 11 truncated transcript sequences*

Description

Sequences of truncated transcripts within human chromosome 11 region that were extracted using `getTxsSeq`.

Usage

```
chr11_truncated_txs_seq
```

Format

object of `DNAStringSet` class.

createIORecord *Create boulder IO record*

Description

Takes a `TsIO` or `TsIOList` object and converts it into a boulder IO record for Primer3. Essentially it converts it into a list of character vectors that each contain the tag and the value in the form: "TAG=VALUE". More on this format can be found in the Primer3 manual.

Usage

```
createIORecord(object, thermo_params_path = NA)

## S4 method for signature 'TsIO'
createIORecord(object, thermo_params_path = NA)

## S4 method for signature 'TsIOList'
createIORecord(object, thermo_params_path = NA)
```

Arguments

`object`	TsIO of TsIOList object for which a Primer3 boulder IO record should be created.
`thermo_params_path`	Optional path (character) to the `primer3_config` directory. Only required when using Primer3 < 2.5.0.

Details

This function is usually not needed by the user, because functions such as designPrimers handle IO record generation. However, this function can for instance be useful to generate IO records, write them to a file and pass them to Primer3 in the conventional way.

Value

A character vector containing the lines of the IO record.

Methods (by class)

- createIORecord(TsIO): Create IO record from TsIO objects.
- createIORecord(TsIOList): Create IO record from TsIO objects.

See Also

http://primer3.org/manual.html for Primer3 manual.

Examples

```
# chromosome 11 truncated transcript sequences
data("chr11_truncated_txs_seq")

# create TsIOList object for primer desing from sequence templates
obj <- TAPseqInput(chr11_truncated_txs_seq, product_size_range = c(350, 500))

# create boulder IO record
boulder_io <- createIORecord(obj)
head(boulder_io, 11)
```

---

designPrimers *Design primers*

Description

Design primers based on TsIO or TsIOList objects. Creates boulder-IO records, passes input to Primer3 and parses the output.

Usage

```
designPrimers(
  object,
  thermo_params_path = NA,
  primer3_core = getOption("TAPseq.primer3_core")
)

## S4 method for signature 'TsIO'
designPrimers(
```

```
  object,
  thermo_params_path = NA,
  primer3_core = getOption("TAPseq.primer3_core")
)

## S4 method for signature 'TsIOList'
designPrimers(
  object,
  thermo_params_path = NA,
  primer3_core = getOption("TAPseq.primer3_core")
)
```

Arguments

object	TsIO or TsIOList object for which primers should be designed.
thermo_params_path	Optional path (character) to the primer3_config directory. Only required when using Primer3 < 2.5.0.
primer3_core	Path (character) to the primer3_core executable. Usually this is inferred when loading/attaching the package.

Value

A new TsIO or TsIOList object containing Primer3 output.

Methods (by class)

- designPrimers(TsIO): Design primers using Primer3 from a TsIO object
- designPrimers(TsIOList): Design primers using Primer3 from a TsIOList object

See Also

http://primer3.org/manual.html for Primer3 manual and TsIO for TsIO class objects.

Examples

```
# chromosome 11 truncated transcript sequences and annotations
data("chr11_truncated_txs_seq")

# create TsIOList object for the first two sequence templates
tapseq_io <- TAPseqInput(chr11_truncated_txs_seq[1:2], product_size_range = c(350, 500))

# design primers
## Not run:
tapseq_io <- designPrimers(tapseq_io)

## End(Not run)

# designed primers are stored in the tapseq_primers slot
tapseq_primers(tapseq_io)
```

`estimateOffTargets` *Estimate primer off-targets using BLAST*

Description

Functions to use BLAST to align TAP-seq primers against a genome and chromosome reference to estimate potential off-target binding sites.

Usage

```
createBLASTDb(
  genome,
  annot,
  blastdb,
  standard_chromosomes = TRUE,
  tx_id = "transcript_id",
  tx_name = "transcript_name",
  gene_name = "gene_name",
  gene_id = "gene_id",
  title = "TAP-seq_GT_DB",
  verbose = FALSE,
  makeblastdb = getOption("TAPseq.makeblastdb")
)

blastPrimers(
  object,
  blastdb,
  max_mismatch = 0,
  min_aligned = 0.75,
 primer_targets = c("transcript_id", "transcript_name", "gene_id", "gene_name"),
  tmpdir = tempdir(),
  blastn = getOption("TAPseq.blastn")
)

## S4 method for signature 'TsIO'
blastPrimers(
  object,
  blastdb,
  max_mismatch = 0,
  min_aligned = 0.75,
 primer_targets = c("transcript_id", "transcript_name", "gene_id", "gene_name"),
  tmpdir = tempdir(),
  blastn = getOption("TAPseq.blastn")
)

## S4 method for signature 'TsIOList'
blastPrimers(
```

```
  object,
  blastdb,
  max_mismatch = 0,
  min_aligned = 0.75,
 primer_targets = c("transcript_id", "transcript_name", "gene_id", "gene_name"),
  tmpdir = tempdir(),
  blastn = getOption("TAPseq.blastn")
)
```

Arguments

genome	A BSgenome (or DNAStringSet) object containing the sequences of all chromosomes to obtain genome and transcript sequences.
annot	A GRanges object containing all exons of transcripts to be considered.
blastdb	TAP-seq BLAST database created with createBLASTDb.
standard_chromosomes	(logical) Specifies whether only standard chromosomes should be included in output genome sequences (e.g. chr1-22, chrX, chrY, chrM for homo sapiens).
tx_id, tx_name, gene_name, gene_id	(character) Column names in annot metadata containing transcript id, transcript name, gene name and gene id information.
title	Optional title for BLAST database.
verbose	(logical) If TRUE, additional information from makeblastdb is printed to the console. Default: FALSE.
makeblastdb	Path to the makeblastdb executable. Usually this is inferred when loading/attaching the package.
object	A TsIO or TsIOList object containing designed primers.
max_mismatch	Maximum number of mismatches allowed for off-target hits (default: 0).
min_aligned	Minimum portion of the primer sequence starting from the 3' end that must align for off-target hits (default: 0.75).
primer_targets	Specifies what should be used to identify primer targets for off-target identification. I.e. to what does the sequence_id in TsIO objects refer? Can be a subset of transcript_id, transcript_name, gene_id or gene_name. By default all 4 are checked. Set to NULL to disable any off-target identification. See Details for more information.
tmpdir	Directory needed to store temporary files.
blastn	Path (character) to the blastn executable. Usually this is inferred when loading/attaching the package.

Details

createBLASTDb creates a BLAST database containing genome and transcriptome sequences, which is required by blastPrimers. The created database contains both sequence files for BLAST and annotations to process the results.

Use `blastPrimers` to align designed TAP-seq primers against the created database to estimate off-target priming potential. Only hits where at least a specified portion of the sequence involving the 3' end of the primer aligns with not more than a certain number of mismatches are considered.

`blastPrimers` counts the number of genes in which a primer has 1) exonic hits or 2) intronic hits, or 3) the number of hits in intergenic regions of the genome. The exonic and intronic counts should be interptreted as: "In how many genes does a primer have exonic (or intronic) hits?".

If a BLAST hit falls in both intronic and exonic regions of a given gene (i.e. exonic for one transcript, intronic for another transcript), only the exonic hit is counted for that gene. If a primer has for instance 3 BLAST hits in one gene, 2 exonic and 1 intronic, then one exonic hit and one intronic hit is counted for that gene.

If sequence IDs of the designed primers (`sequence_id`) refer to the target gene/transcripts and can be found in the BLAST database annotations via `primer_targets`, then only off-target hits are counted. This is usually the case if input for primer design was produced from target gene annotations.

Value

For `createBLASTDb` a directory containing the BLAST database. For `blastPrimers` a `TsIO` or `TsIOList` object with the number of potential off-targets added to the TAP-seq primer metadata.

Functions

- `createBLASTDb()`: Create a genome and transcriptome TAP-seq BLAST database
- `blastPrimers(TsIO)`: BLAST primers in a `TsIO` object
- `blastPrimers(TsIOList)`: BLAST primers in a `TsIOList` object

Examples

```
## Not run:
library(BSgenome)

# human genome (hg38) BSgenome object
hg38 <- getBSgenome("BSgenome.Hsapiens.UCSC.hg38")

# get annotations for BLAST
annot_url <- paste0("ftp://ftp.ebi.ac.uk/pub/databases/gencode/Gencode_human/release_32/",
                    "gencode.v32.annotation.gtf.gz")
annot <- import(annot_url, format = "gtf")
blast_exons <- annot[annot$type == "exon" & annot$gene_type == "protein_coding"]

# build BLAST database
blastdb <- file.path(tempdir(), "blastdb")
createBLASTDb(genome = hg38, annot = blast_exons, blastdb = blastdb)

# chr11 primers example data (already contains off-targets, but we can overwrite them)
data("chr11_primers")
chr11_primers <- chr11_primers[1:3]  # only use a small subset for this example

# run blast to identify potential off-targets
```

```
chr11_primers <- blastPrimers(chr11_primers, blastdb = blastdb)
tapseq_primers(chr11_primers)

# allow 1 mismatch between primer and off-target
chr11_primers <- blastPrimers(chr11_primers, blastdb = blastdb, max_mismatch = 1)
tapseq_primers(chr11_primers)

## End(Not run)
```

---

exportPrimers *Export TAP-seq primers*

Description

A set of functions for TAP-seq primer export. Convert primers stored in TsIO or TsIOList objects to a simple data.frame for easier export. Or create BED format tracks for primers and write them to files for viewing in a genome browser (e.g. IGV).

Usage

```
createPrimerTrack(object, color = 1)

## S4 method for signature 'TsIO'
createPrimerTrack(object, color = 1)

## S4 method for signature 'TsIOList'
createPrimerTrack(object, color = 1)

exportPrimerTrack(..., con)

primerDataFrame(object)

## S4 method for signature 'TsIO'
primerDataFrame(object)

## S4 method for signature 'TsIOList'
primerDataFrame(object)
```

Arguments

object	A TsIO or TsIOList object containing designed primers.
color	Color used for the track (Default: black). Can be any of the three kinds of R color specifications.
...	One or more primer BED tracks created by createPrimerTrack.
con	Connection to which tracks are written. Typically a .bed file.

Value

For `createPrimerTrack` a `data.frame` with the primer track in BED format.

Functions

- `createPrimerTrack(TsIO)`: Create primer BED track from `TsIO` objects
- `createPrimerTrack(TsIOList)`: Create primer BED track from `TsIOList` objects
- `exportPrimerTrack()`: Export primer BED tracks files
- `primerDataFrame(TsIO)`: Create a `data.frame` with primer data from `TsIO`
- `primerDataFrame(TsIOList)`: Create a `data.frame` with primer data from `TsIOList`

Examples

```
# chr11 primers example data
data("chr11_primers")

# pick best primers based on predicted off-targets
best_primers <- pickPrimers(chr11_primers, n = 1, by = "off_targets")

# primers data can be exported to a simple data.frame to e.g. write them to a .csv file
primers_df <- primerDataFrame(best_primers)
head(primers_df)


# primer binding sites in transcript sequences can be converted to genomic coordinates to create
# a BED track to visualize primers in a genome browser (e.g. IGV)

# create primer BED track with a fancy color
track <- createPrimerTrack(best_primers[1:5], color = "steelblue3")

# tracks can be written to .bed files using a little helper function (replace con = "" by a file)
exportPrimerTrack(track, con = "")

## Not run:
# one can easily export primer tracks for multiple TsIO or TsIOList objects (e.g. inner and
# outer nested primers) to one .bed file using different colors for each object. see vignette for
# a practical example:
vignette("tapseq_primer_design", package = "TAPseq")

obj1 <- best_primers[1:5]
obj2 <- best_primers[6:10]
exportPrimerTrack(createPrimerTrack(obj1, color = "steelblue3"),
                  createPrimerTrack(obj2, color = "goldenrod1"),
                  con = "path/to/file.bed")


## End(Not run)
```

getTxsSeq *Get transcript sequences*

Description

Extract the DNA sequences of all exons of transcript models and concatenate to one sequence per transcript. This is basically a wrapper for `extractTranscriptSeqs`, which makes sure that the exons are correctly sorted according to their position in the transcript (3' to 5').

Usage

```
getTxsSeq(transcripts, genome)

## S4 method for signature 'GRangesList'
getTxsSeq(transcripts, genome)

## S4 method for signature 'GRanges'
getTxsSeq(transcripts, genome)
```

Arguments

`transcripts`	A `GRanges` or `GRangesList` object containing exons of transcripts for which sequences should be extracted. All exons in a `GRanges` object are assumed to belong to the same transcript. Multiple transcripts can be provided in a `GRangesList` object.
`genome`	A `BSgenome` or `DNAStringSet` object containing chromosome sequences which should be used to extract transcript sequences. Although using a `BSgenome` object is the easiest way, the genome sequence could also be loaded from a FASTA file using `readDNAStringSet`.

Value

A `DNAString` or `DNAStringSet` object containing the transcript sequence(s).

Methods (by class)

- `getTxsSeq(GRangesList)`: Obtain transcript sequence from `GRangesList` input
- `getTxsSeq(GRanges)`: Obtain transcript sequence from `GRanges` input

Examples

```
library(BSgenome)

# protein-coding exons of transcripts within chr11 region
data("chr11_genes")
target_txs <- split(chr11_genes, f = chr11_genes$transcript_id)
```

```
# human genome (hg38) BSgenome object (needs to be installed separately from Bioconductor)
hg38 <- getBSgenome("BSgenome.Hsapiens.UCSC.hg38")

# get sequences for all target transcripts on chr11
txs_seqs <- getTxsSeq(target_txs, genome = hg38)
```

`get_gt_sequences` *Get genome and transcriptome sequences*

Description

Get DNA sequences of all chromosomes and all annotated transcripts of a genome. This function is used to create the sequences in `createBLASTDb`.

Usage

```
get_gt_sequences(
  genome,
  annot = NULL,
  tx_id = "transcript_id",
  tx_name = "transcript_name",
  gene_name = "gene_name",
  gene_id = "gene_id",
  include_genome = TRUE,
  standard_chromosomes = TRUE
)
```

Arguments

`genome`	A `BSgenome` (or `DNAStringSet`) object containing the chromosome sequences to obtain genome and / or transcript sequences.
`annot`	A `GRanges` object containing all exons of transcripts to be considered. If not specified, no transcript sequences will be included in the output fasta file.
`tx_id, tx_name, gene_name, gene_id`	(character) Column names in annot metadata containing transcript id, transcript name, gene name and gene id information. These column are mandatory, but can contain internal names (e.g. "transcript-1" or "gene-1").
`include_genome`	(logical) Specifies whether the genome sequence should be included in the output fasta file.
`standard_chromosomes`	(logical) Specifies whether only standard chromosomes should be included in output genome sequences (e.g. chr1-22, chrX, chrY, chrM for homo sapiens).
`compress`	(logical) Create a gzipped output fasta file.

Value

A `DNAStringSet` object containing the genome and transcriptome sequences.

`inferPolyASites` *Infer polyA sites from droplet sequencing data*

Description

Infer polyA sites from 10X, Drop-seq or similar 3' enriched sequencing data. Simple function that looks for peaks in read coverage to estimate potential polyA sites. Default parameters are chosen because they work reasonably well with the example data, but they should typically be empirically selected by verifying the output.

Usage

```
inferPolyASites(
  genes,
  bam,
  polyA_downstream = 100,
  min_cvrg = 0,
  wdsize = 200,
  by = 1,
  extend_downstream = 0,
  perc_threshold = 0.9,
  parallel = FALSE
)
```

Arguments

`genes`	`GRangesList` object containing annotations of genes for which polyA sites are to be estimated.
`bam`	Path to .bam file containing aligned reads used for polyA site estimation.
`polyA_downstream`	(numeric) How far downstream of a peak in coverage are polyA sites expected? Somewhat depends on input DNA fragment size. (default: 100). Importantly, this value should not be larger than half of the window size (wdsize), else polyA sites might be moved outside of the transcripts, even if they were extended using the extend_downstream parameter.
`min_cvrg`	(numeric) Minimal coverage for peaks to be considered for polyA site estimation (default: 0).
`wdsize`	(numeric) Window size to estimate sequencing coverage along transcripts (default: 200).
`by`	(numeric) Steps in basepairs in which the sliding window should be moved along transcripts to estimate smooth coverage (default: 1).
`extend_downstream`	(numeric) To which amount should transcript annotations be extended downstream when estimating polyA sites (default: 0). A reasonable value (e.g. 100-200 bp) allows to account for polyA sites that fall a few basepairs downstream of terminal exons.

perc_threshold (numeric) Only sequencing coverage peaks within perc_threshold percentile of coverage are considered for polyA site estimation (default: 0.9). Avoids that small peaks that in coverage are considered, resulting in manby false polyA sites.

parallel (logical) Triggers parallel computing using the BiocParallel-package package. This requires that a parallel back-end was registered prior to executing the function. (default: FALSE).

Value

A GRanges object containing coordinates of estimated polyadenylation sites.

Examples

```
library(GenomicRanges)

# protein-coding exons of genes within chr11 region
data("chr11_genes")
target_genes <- split(chr11_genes, f = chr11_genes$gene_name)

# subset of target genes for quick example
target_genes <- target_genes[18:27]

# bam file containing aligned Drop-seq reads
dropseq_bam <- system.file("extdata", "chr11_k562_dropseq.bam", package = "TAPseq")

# infer polyA sites for all target genes with adjusted parameters. parameter values depend on the
# input data and at this stage it's best to try different settings and check the results
polyA_sites <- inferPolyASites(target_genes, bam = dropseq_bam, polyA_downstream = 50,
                               wdsize = 100, min_cvrg = 1, parallel = TRUE)
```

---

parsePrimer3Output *Parse Primer3 Output*

Description

Parse Primer3 output and add to input TsIO or TsIOList object. This function is usually not used by the user, as designPrimers handles Primer3 output parsing.

Usage

```
parsePrimer3Output(object, primer3_output)
```

Arguments

object The TsIO or TsIOList object used to design primers. No errors or warnings if this is a different TsIO or TsIOList object!

primer3_output Character vector containing raw Primer3 output.

Value

TsIO or TsIOList object with added Primer3 output

Examples

```
## Not run:
# chromosome 11 truncated transcript sequences
data("chr11_truncated_txs_seq")

# create TsIOList object for the first two sequence templates
tapseq_io <- TAPseqInput(chr11_truncated_txs_seq[1:2], product_size_range = c(350, 500))

# create boulder IO records
io_record <- createIORecord(tapseq_io)

# design primers and store raw Primer3 output
primer3_core <- getOption("TAPseq.primer3_core")
primer3_output <- system2(command = primer3_core, input = io_record, stdout = TRUE)

# parse output and add it to input TsIO object(s)
tapseq_io <- parsePrimer3Output(tapseq_io, primer3_output)
tapseq_primers(tapseq_io)

## End(Not run)
```

---

pickPrimers *Pick best TAP-seq primers*

Description

Pick based primers from designed primers for every target based on Primer3 penalty score or off-target priming estimated with blastPrimers.

Usage

```
pickPrimers(object, n = 1, by = c("penalty", "off_targets"))

## S4 method for signature 'TsIO'
pickPrimers(object, n = 1, by = c("penalty", "off_targets"))

## S4 method for signature 'TsIOList'
pickPrimers(object, n = 1, by = c("penalty", "off_targets"))
```

Arguments

object	A TsIO or TsIOList object containing designed primers.
n	The number of top primers to pick (default: 1, which returns the best primer).
by	Attribute by which primers should be picked. Can be either penalty or off_targets.

Details

If by is set to off_targets top primers are picked based on the lowest number of exonic, intronic and intergenic off-targets (in that priority).

Value

A TsIO or TsIOList object containing the picked primers.

Methods (by class)

- pickPrimers(TsIO): Pick best primers in a TsIO object
- pickPrimers(TsIOList): Pick best primers per target in a TsIOList object

Examples

```
# chr11 primers examples
data("chr11_primers")

# pick the best primer per gene based on the fewest exonic, intronic and intergenic off-targets
# (in that order)
best_primers <- pickPrimers(chr11_primers, by = "off_targets")
tapseq_primers(best_primers)

# pick the best two primers per gene based on the lowest penalty score computed by Primer3
best_primers <- pickPrimers(chr11_primers, n = 2, by = "penalty")
tapseq_primers(best_primers)
```

---

selectTargetGenes *Select target genes*

Description

Select target genes that serve as markers for cell populations using a linear model with lasso regularization. How well a selected set of target genes discriminates between cell populations can be assessed in an intuitive way using UMAP visualization.

Usage

```
selectTargetGenes(object, targets = NULL, expr_percentile = c(0.6, 0.99))

plotTargetGenes(object, target_genes, npcs = 15)
```

Arguments

object	Seurat object containing single-cell RNA-seq data from which best marker genes for different cell populations should be learned. Needs to contain population identities for all cell.
targets	Desired number of target genes. Approximately this many target genes will be returned. If set to NULL, the optimal number of target genes will be estimated using a cross-valdation approach. Warning: The number of target genes might end up being very large!
expr_percentile	Expression percentiles that candidate target genes need to fall into. Default is 60% to 99%, which excludes bottom 60% and top 1% expressed genes from markers.
target_genes	(character) Target gene names.
npcs	(integer) Number of principal components to use for UMAP.

Value

A character vector containing selected target gene identifiers.

Examples

```
library(Seurat)

# example of mouse bone marrow 10x gene expression data
data("bone_marrow_genex")

# identify approximately 100 target genes that can be used to identify cell populations
target_genes <- selectTargetGenes(bone_marrow_genex, targets = 100)

# automatically identify the number of target genes to best identify cell populations using
# cross-validation. caution: this can lead to very large target gene panels!
target_genes_cv <- selectTargetGenes(bone_marrow_genex)

# create UMAP plots to compare cell type identification based on full dataset and selected 100
# target genes
plotTargetGenes(bone_marrow_genex, target_genes = target_genes)
```

TAPseq *TAPseq: R-package to design primers for TAP-seq*

Description

This package provides functions to select transcript isoforms and design PCR primers for TAP-seq.

Installation

In order to use the full functionality, Primer3 and BLAST need to be installed and added to PATH. Furthermore, the `primer3_config` directory containing important files for Primer3 should be located in the same directory as the `primer3_core` executable. If this is not practical, all functions interacting with Primer3 have arguments to specify the paths to these files.

For more information on installation see: https://github.com/argschwind/TAPseq.

`TAPseqInput` *Create TAPseq input from target sequences*

Description

This function creates input for TAP-seq primer design from a DNAStringSet containing the target sequences (typically transcript sequences).

Usage

```
TAPseqInput(
  target_sequences,
  product_size_range,
  beads_oligo = NA,
  reverse_primer = "CTACACGACGCTCTTCCGATCT",
  target_annot = NULL,
  primer_num_return = 5,
  min_primer_region = 100,
  primer_opt_tm = 63,
  primer_min_tm = 59,
  primer_max_tm = 66
)
```

Arguments

`target_sequences`
: A named `DNAStringSet` object containing all target sequences.

`product_size_range`
: Numerical vector of length 2 specifying the desired length of the resulting amplicons.

`beads_oligo`
: Beads-oligo-dT sequence for the used droplet sequencing protocol (10x, Drop-seq). If nothing is specified (`beads_oligo = NA`), the 10x V3 Beads-oligo-dT sequence is used. Can be changed if primers are for instance designed for Drop-seq. Any barcode bases need to be replaced by `N`.

`reverse_primer`
: Reverse primer sequence used for all PCR reactions. Default is the 10x primer sequence: `CTACACGACGCTCTTCCGATCT`.

target_annot (optional) A named GRangesList object with transcript annotations in case the targets are transcripts of gene loci. If provided, each GRanges within the list should contain all exons of one targeted transcripts. Names need to be the same as for target_sequences.

primer_num_return
How many forward primers should be designed? (default: 5)

min_primer_region
Minimum sequence length required for primer design. Mostly relevant in case a sequence template is too short to allow the specified product_size_range.

primer_opt_tm, primer_min_tm, primer_max_tm
Optimal, minumum and maximum primer melting temperature. Set to NA to use Primer3s default values.

Value

TsIOList object.

Examples

```
# chromosome 11 truncated transcript sequences and annotations
data("chr11_truncated_txs_seq")

# create TsIOList object for primer design from target sequences
obj <- TAPseqInput(chr11_truncated_txs_seq, product_size_range = c(350, 500))
obj

# transcript annotations can be added for optional genome browser tracks of designed primers
data("chr11_truncated_txs")
obj <- TAPseqInput(chr11_truncated_txs_seq, product_size_range = c(350, 500),
                   target_annot = chr11_truncated_txs)

# create input for primer design with Drop-seq instead of default 10x
ds_oligo <- "TTTTTTTAAGCAGTGGTATCAACGCAGAGTACNNNNNNNNNNNNNNNNNNNNTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT"
ds_rev_primer <- "AAGCAGTGGTATCAACGCAGAGT"
ds_obj <- TAPseqInput(chr11_truncated_txs_seq, beads_oligo = ds_oligo,
                      reverse_primer = ds_rev_primer, product_size_range = c(350, 500),
                      primer_opt_tm = 62, primer_min_tm = 57, primer_max_tm = 65)
```

---

truncateTxsPolyA *Truncate transcripts at polyA sites*

Description

Truncate transcripts at overlapping polyadenylation (polyA) sites to infer likely 3' ends of transcripts. This is crucial to correctly design TAP-seq primers that amplify fragments of specific lengths. Typically the exons of all annotated transcripts per target gene are provided as input. If a polyA site overlaps a single transcript of a given gene, this transcript is truncated and returned. In case a polyA site overlaps multiple transcripts of the same gene, a "metatranscript" consisting of

all annotated exons of the overlapping transcripts is generated and truncated. No statements about expressed transcripts can be made if no overlapping polyA sites are found for any transcripts of a gene. In that case a "meta transcript" consisting of the merged exons of that gene is generated and returned.

Usage

```
truncateTxsPolyA(
  transcripts,
  polyA_sites,
  extend_3prime_end = 0,
  polyA_select = c("downstream", "upstream", "score"),
  transcript_id = "transcript_id",
  gene_id = "gene_id",
  exon_number = "exon_number",
  ignore_strand = FALSE,
  parallel = FALSE
)

## S4 method for signature 'GRanges'
truncateTxsPolyA(
  transcripts,
  polyA_sites,
  extend_3prime_end = 0,
  polyA_select = c("downstream", "upstream", "score"),
  transcript_id = "transcript_id",
  gene_id = "gene_id",
  exon_number = "exon_number",
  ignore_strand = FALSE,
  parallel = FALSE
)

## S4 method for signature 'GRangesList'
truncateTxsPolyA(
  transcripts,
  polyA_sites,
  extend_3prime_end = 0,
  polyA_select = c("downstream", "upstream", "score"),
  transcript_id = "transcript_id",
  gene_id = "gene_id",
  exon_number = "exon_number",
  ignore_strand = FALSE,
  parallel = FALSE
)
```

Arguments

transcripts	A GRanges or GRangesList object containing exons of the transcripts to be truncated. Transcripts for multiple genes can be provided as GRanges objects

within a GRangesList.

polyA_sites — A GRanges object containing the polyA sites. This needs to contain a metadata entry names "score" if the option polyA_select = "score" is used. PolyA sites can be either obtained via running inferPolyASites or imported from an existing .bed file (BEDFile).

extend_3prime_end — Specifies how far (bp) 3' ends of transcripts should be extended when looking for overlapping polyA sites (default = 0). This enables capturing of polyA sites that occur downstream of annotated 3' ends.

polyA_select — Specifies which heuristic should be used to select the polyA site used to truncate the transcripts if multiple overlapping polyA sites are found. By default "downstream" is used which chooses the most downstream polyA site. "score" selects the polyA site with the highest score, which corresponds to the read coverage when using inferPolyASites to estimate polyA sites.

transcript_id — (character) Name of the column in the metadata of transcripts providing transcript id for each exon (default: "transcript_id"). Set to NULL to ignore transcript ids and assume that all exons per gene belong to the same transcript.

gene_id, exon_number — (character) Optional names of columns in metadata of transcripts containing gene id and exon number. These are only used to create new metadata when merging multiple transcripts into a meta transcript.

ignore_strand — (logical) Specifies whether the strand of polyA sites should be ignored when looking for overlapping polyA sites. Default is FALSE and therefore only polyA sites on the same strand as the transcripts are considered. PolyA sites with strand * has the same effect as ignore_strand = TRUE.

parallel — (logical) Triggers parallel computing using the BiocParallel package. This requires that a parallel back-end was registered prior to executing the function. (default: FALSE).

Value

Either a GRanges or GRangesList object containing the truncated transcripts.

Methods (by class)

- truncateTxsPolyA(GRanges): Truncate transcripts of one gene provided as GRanges object
- truncateTxsPolyA(GRangesList): Truncate transcripts of multiple genes provided as GRangesList

Examples

```
library(GenomicRanges)

# protein-coding exons of genes within chr11 region
data("chr11_genes")
target_genes <- split(chr11_genes, f = chr11_genes$gene_name)

# only retain first 2 target genes, because truncating transcripts is currently computationally
# quite costly. try using BiocParallel for parallelization (see ?truncateTxsPolyA).
```

```
target_genes <- target_genes[1:2]

# example polyA sites for these genes
data("chr11_polyA_sites")

# truncate target genes at most downstream polyA site (default)
truncated_txs <- truncateTxsPolyA(target_genes, polyA_sites = chr11_polyA_sites)

# change polyA selection to "score" (read coverage of polyA sites) and extend 3' end of target
# genes by 50 bp (see ?truncateTxsPolyA).
truncated_txs <- truncateTxsPolyA(target_genes, polyA_sites = chr11_polyA_sites,
                                  polyA_select = "score", extend_3prime_end = 50)
```

`TsIO-class` *TsIO class*

Description

TsIO objects store TAP-seq Primer3 input and output.

Usage

```
TsIO(
  sequence_id,
  target_sequence,
  beads_oligo,
  reverse_primer,
  product_size_range,
  target_annot = NULL,
  primer_num_return = 5,
  min_primer_region = 100,
  primer_opt_tm = NA,
  primer_min_tm = NA,
  primer_max_tm = NA
)

## S4 method for signature 'TsIO'
sequence_id(x)

## S4 replacement method for signature 'TsIO'
sequence_id(x) <- value

## S4 method for signature 'TsIO'
target_sequence(x)

## S4 replacement method for signature 'TsIO'
target_sequence(x) <- value
```

```
## S4 method for signature 'TsIO'
beads_oligo(x)

## S4 replacement method for signature 'TsIO'
beads_oligo(x) <- value

## S4 method for signature 'TsIO'
reverse_primer(x)

## S4 replacement method for signature 'TsIO'
reverse_primer(x) <- value

## S4 method for signature 'TsIO'
target_annot(x)

## S4 replacement method for signature 'TsIO'
target_annot(x) <- value

## S4 method for signature 'TsIO'
product_size_range(x)

## S4 replacement method for signature 'TsIO'
product_size_range(x) <- value

## S4 method for signature 'TsIO'
primer_num_return(x)

## S4 replacement method for signature 'TsIO'
primer_num_return(x) <- value

## S4 method for signature 'TsIO'
min_primer_region(x)

## S4 replacement method for signature 'TsIO'
min_primer_region(x) <- value

## S4 method for signature 'TsIO'
primer_opt_tm(x)

## S4 replacement method for signature 'TsIO'
primer_opt_tm(x) <- value

## S4 method for signature 'TsIO'
primer_min_tm(x)

## S4 replacement method for signature 'TsIO'
primer_min_tm(x) <- value
```

```
## S4 method for signature 'TsIO'
primer_max_tm(x)

## S4 replacement method for signature 'TsIO'
primer_max_tm(x) <- value

## S4 method for signature 'TsIO'
sequence_template(x)

## S4 method for signature 'TsIO'
tapseq_primers(x)

## S4 method for signature 'TsIO'
pcr_products(x)
```

Arguments

`sequence_id`	Name (`character`) of the target sequence, e.g. the gene name. It's adviced to use meaningful sequence ids to savely assign designed primers to their targets.
`target_sequence`	A `DNAString` or `character` object containing the target sequence for which primers should be designed. Usually a transcript sequence.
`beads_oligo`	Beads-oligo-dT sequence for the used droplet sequencing protocol (10x, Drop-seq).
`reverse_primer`	Reverse primer sequence used for all PCR reactions.
`product_size_range`	Numerical vector of length 2 specifying the desired length of the resulting amplicons.
`target_annot`	(optional) A `GRanges` object with transcript annotation in case the target is a transcript of a gene locus. If provided, it should contain all exons of the targeted transcript.
`primer_num_return`	How many forward primers should be designed? (default: 5)
`min_primer_region`	Minimum sequence length required for primer design. Mostly relevant in case the sequence template is too short to allow the specified `product_size_range`.
`primer_opt_tm`, `primer_min_tm`, `primer_max_tm`	Optimal, minumum and maximum primer melting temperature.
`x`	A `TsIO` object.
`value`	A valid value to assign to the chosen slot.
`tapseq_primers`	Slot where designed TAP-seq primers are stored. Not set by user.
`pcr_products`	Slot where PCR products of primers are stored. Not set by user.

Details

The TsIO class is based on the Boulder IO records used by Primer3 (Primer3 manual). These objects are used to store the sequence templates and parameters needed for TAP-seq primer design. Primers designed with Primer3 are also stored in the same TsIO objects.

Use `TsIO()` to construct a new TsIO object from scratch.

Value

A `TsIO` object.

Methods (by generic)

- `sequence_id(TsIO)`: Get sequence_id
- `sequence_id(TsIO) <- value`: Set sequence_id
- `target_sequence(TsIO)`: Get target_sequence
- `target_sequence(TsIO) <- value`: Set target_sequence
- `beads_oligo(TsIO)`: Get beads_oligo
- `beads_oligo(TsIO) <- value`: Set beads_oligo
- `reverse_primer(TsIO)`: Get reverse_primer
- `reverse_primer(TsIO) <- value`: Set reverse_primer
- `target_annot(TsIO)`: Get target_annot
- `target_annot(TsIO) <- value`: Set target_annot
- `product_size_range(TsIO)`: Get product_size_range
- `product_size_range(TsIO) <- value`: Set product_size_range
- `primer_num_return(TsIO)`: Get primer_num_return
- `primer_num_return(TsIO) <- value`: Set primer_num_return
- `min_primer_region(TsIO)`: Get min_primer_region
- `min_primer_region(TsIO) <- value`: Set min_primer_region
- `primer_opt_tm(TsIO)`: Get primer_opt_tm
- `primer_opt_tm(TsIO) <- value`: Set primer_opt_tm
- `primer_min_tm(TsIO)`: Get primer_min_tm
- `primer_min_tm(TsIO) <- value`: Set primer_min_tm
- `primer_max_tm(TsIO)`: Get primer_max_tm
- `primer_max_tm(TsIO) <- value`: Set primer_max_tm
- `sequence_template(TsIO)`: Create sequence_template
- `tapseq_primers(TsIO)`: Get tapseq_primers
- `pcr_products(TsIO)`: Get pcr_products

See Also

`http://primer3.org/manual.html` for Primer3 manual.

Examples

```
# get example transcript sequence
data("chr11_truncated_txs_seq")
tx_seq <- chr11_truncated_txs_seq[[1]]
tx_id <- names(chr11_truncated_txs_seq)[1]

# 10x beads-oligo-dt sequence
beads_oligo <- "CTACACGACGCTCTTCCGATCTNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT"

# reverse primer used in all PCR reactions
reverse_primer <- "CTACACGACGCTCTTCCGATCT"

# create TsIO object
obj <- TsIO(sequence_id = tx_id, target_sequence = tx_seq, beads_oligo = beads_oligo,
            reverse_primer = reverse_primer, product_size_range = c(350, 500))

# slot values can be accessed using accessor functions
sequence_id(obj)
sequence_id(obj) <- "Gene1"
sequence_id(obj)

# the sequence template (target sequence + reverse complement of beads-oligo-dt) for primer
# design can be viewed as well
sequence_template(obj)
```

`TsIOList-class` *TsIOList class*

Description

TsIOList class is a container to store multiple `TsIO` objects. This enables storing of Primer3 input and output for multiple target genes.

Usage

```
TsIOList(...)

## S4 method for signature 'TsIOList'
sequence_id(x)

## S4 method for signature 'TsIOList'
target_sequence(x)

## S4 method for signature 'TsIOList'
sequence_template(x)

## S4 method for signature 'TsIOList'
target_annot(x)
```

```
## S4 method for signature 'TsIOList'
tapseq_primers(x)

## S4 method for signature 'TsIOList'
pcr_products(x)
```

Arguments

...	Multiple TsIO objects from which a TsIOList object should be created.
x	A TsIOList object.

Value

A TsIOList object.

Methods (by generic)

- sequence_id(TsIOList): Get sequence_id
- target_sequence(TsIOList): Get target_sequence
- sequence_template(TsIOList): Create sequence_template
- target_annot(TsIOList): Get target_annot
- tapseq_primers(TsIOList): Get tapseq_primers
- pcr_products(TsIOList): Get pcr_products

See Also

TsIO

Examples

```
# get example transcript sequences
data("chr11_truncated_txs_seq")
txs_seqs <- chr11_truncated_txs_seq[1:2]
txs_ids <- names(txs_seqs)

# 10x beads-oligo-dt sequence
beads_oligo <- "CTACACGACGCTCTTCCGATCTNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT"

# reverse primer used in all PCR reactions
reverse_primer <- "CTACACGACGCTCTTCCGATCT"

# create TsIO objects
tsio1 <- TsIO(sequence_id = txs_ids[1], target_sequence = txs_seqs[[1]],
              beads_oligo = beads_oligo, reverse_primer = reverse_primer,
              product_size_range = c(350, 500))

tsio2 <- TsIO(sequence_id = txs_ids[2], target_sequence = txs_seqs[[2]],
              beads_oligo = beads_oligo, reverse_primer = reverse_primer,
```

```
                product_size_range = c(350, 500))

# create TsIOList object
obj <- TsIOList(tsio1 = tsio1, tsio2 = tsio2)

# it's noteworthy to mention that when creating a TsIOList from a DNAStringSet of target
# sequences, it's easier to use TAPseqInput()
?TAPseqInput

# as with TsIO objects, some values can be accessed using accessor functions
sequence_template(obj)
```

Index