Package ‘amplican’

May 29, 2024

Type Package

Title Automated analysis of CRISPR experiments

Description `amplican` performs alignment of the amplicon reads, normalizes gathered data, calculates multiple statistics (e.g. cut rates, frameshifts) and presents results in form of aggregated reports. Data and statistics can be broken down by experiments, barcodes, user defined groups, guides and amplicons allowing for quick identification of potential problems.

Version 1.26.0

URL https://github.com/valenlab/amplican

BugReports https://github.com/valenlab/amplican/issues

biocViews ImmunoOncology, Technology, Alignment, qPCR, CRISPR

License GPL-3

LinkingTo Rcpp

Depends R (>= 3.5.0), methods, BiocGenerics (>= 0.22.0), Biostrings (>= 2.44.2), pwalign, data.table (>= 1.10.4-3)

Imports Rcpp, utils (>= 3.4.1), S4Vectors (>= 0.14.3), ShortRead (>= 1.34.0), IRanges (>= 2.10.2), GenomicRanges (>= 1.28.4), GenomeInfoDb (>= 1.12.2), BiocParallel (>= 1.10.1), gtable (>= 0.2.0), gridExtra (>= 2.2.1), ggplot2 (>= 3.3.4), ggthemes (>= 3.4.0), waffle (>= 0.7.0), stringr (>= 1.2.0), stats (>= 3.4.1), matrixStats (>= 0.52.2), Matrix (>= 1.2-10), dplyr (>= 0.7.2), rmarkdown (>= 1.16), knitr (>= 1.16), cluster (>= 2.1.4)

RoxygenNote 7.3.1

Suggests testthat, BiocStyle, GenomicAlignments

Collate 'helpers_general.R' 'AlignmentsExperimentSet-class.R'
  'RcppExports.R' 'helpers_rmd.R' 'amplicanReport.R'
  'helpers_directory.R' 'helpers_warnings.R' 'helpers_filters.R'
  'helpers_alignment.R' 'amplicanAlign.R' 'amplican.R'
  'amplicanFilter.R' 'amplicanNormalize.R' 'amplicanSummarize.R'
  'ggforce_bezier.R' 'helpers_plots.R'

VignetteBuilder knitr
Contents

AlignmentExperimentSet-class ........................................... 4
alphabetQuality ............................................................... 7
amplican ................................................................. 8
amplicanAlign ............................................................ 8
amplicanConsensus ......................................................... 11
amplicanFilter ............................................................. 12
amplicanMap .............................................................. 13
amplicanNormalize ........................................................ 14
amplicanOverlap .......................................................... 15
amplicanPipeline .......................................................... 16
amplicanPipelineConservative ......................................... 19
amplicanReport ............................................................ 23
amplicanSummarize ....................................................... 24
amplican_print_reads ..................................................... 25
assignedCount ............................................................. 26
barcodeData ............................................................... 26
barcodeData<- ........................................................... 27
checkConfigFile .......................................................... 28
checkFileWriteAccess ................................................... 28
checkPrimers ............................................................. 29
cHECKTarget ............................................................ 29
cigarsToEvents ........................................................... 30
comb_along ............................................................... 31
cumsummw ............................................................... 31
decode ............................................................... 32
defGR ............................................................ 32
experimentData .......................................................... 33
experimentData<- ......................................................... 34
extractEvents ............................................................. 34
findEOP ............................................................... 35
findLQR ............................................................... 36
findPD ............................................................... 37
# Contents

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>flipRanges</td>
<td>38</td>
</tr>
<tr>
<td>fwdReads</td>
<td>38</td>
</tr>
<tr>
<td>fwdReads&lt;-</td>
<td>39</td>
</tr>
<tr>
<td>fwdReadsType</td>
<td>39</td>
</tr>
<tr>
<td>fwdReadsType&lt;-</td>
<td>40</td>
</tr>
<tr>
<td>geom_bezier</td>
<td>40</td>
</tr>
<tr>
<td>getEventInfo</td>
<td>44</td>
</tr>
<tr>
<td>getEvents</td>
<td>44</td>
</tr>
<tr>
<td>get_left_primer</td>
<td>45</td>
</tr>
<tr>
<td>get_right_primer</td>
<td>46</td>
</tr>
<tr>
<td>get_seq</td>
<td>46</td>
</tr>
<tr>
<td>goodAvgQuality</td>
<td>47</td>
</tr>
<tr>
<td>goodBaseQuality</td>
<td>47</td>
</tr>
<tr>
<td>is_hdr_strict</td>
<td>48</td>
</tr>
<tr>
<td>lookupAlignment</td>
<td>48</td>
</tr>
<tr>
<td>makeAlignment</td>
<td>49</td>
</tr>
<tr>
<td>metaplot_deletions</td>
<td>51</td>
</tr>
<tr>
<td>metaplot_insertions</td>
<td>52</td>
</tr>
<tr>
<td>metaplot_mismatches</td>
<td>53</td>
</tr>
<tr>
<td>pairToEvents</td>
<td>54</td>
</tr>
<tr>
<td>plot_amplicon</td>
<td>54</td>
</tr>
<tr>
<td>plot_cuts</td>
<td>55</td>
</tr>
<tr>
<td>plot_deletions</td>
<td>56</td>
</tr>
<tr>
<td>plot_height</td>
<td>57</td>
</tr>
<tr>
<td>plot_heterogeneity</td>
<td>58</td>
</tr>
<tr>
<td>plot_insertions</td>
<td>59</td>
</tr>
<tr>
<td>plot_mismatches</td>
<td>60</td>
</tr>
<tr>
<td>plot_variants</td>
<td>61</td>
</tr>
<tr>
<td>readCounts</td>
<td>63</td>
</tr>
<tr>
<td>readCounts&lt;-</td>
<td>63</td>
</tr>
<tr>
<td>revComp</td>
<td>64</td>
</tr>
<tr>
<td>rveReads</td>
<td>64</td>
</tr>
<tr>
<td>rveReads&lt;-</td>
<td>65</td>
</tr>
<tr>
<td>rveReadsType</td>
<td>65</td>
</tr>
<tr>
<td>rveReadsType&lt;-</td>
<td>66</td>
</tr>
<tr>
<td>unassignedCount</td>
<td>67</td>
</tr>
<tr>
<td>unassignedData</td>
<td>67</td>
</tr>
<tr>
<td>unassignedData&lt;-</td>
<td>68</td>
</tr>
<tr>
<td>upperGroups</td>
<td>68</td>
</tr>
<tr>
<td>writeAlignments</td>
<td>69</td>
</tr>
</tbody>
</table>

# Index

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>70</td>
</tr>
</tbody>
</table>
AlignmentsExperimentSet-class

An S4 class to represent alignments from multiple experiments

Description

Class AlignmentsExperimentSet holds data from multiple alignments for many experiments. Allows to examine alignments in great detail.

Usage

AlignmentsExperimentSet(...)

## S4 method for signature 'AlignmentsExperimentSet'
length(x)

## S4 method for signature 'AlignmentsExperimentSet'
fwdReads(x)

## S4 replacement method for signature 'AlignmentsExperimentSet'
fwdReads(x) <- value

## S4 method for signature 'AlignmentsExperimentSet'
rveReads(x)

## S4 replacement method for signature 'AlignmentsExperimentSet'
rveReads(x) <- value

## S4 method for signature 'AlignmentsExperimentSet'
fwdReadsType(x)

## S4 replacement method for signature 'AlignmentsExperimentSet'
fwdReadsType(x) <- value

## S4 method for signature 'AlignmentsExperimentSet'
rveReadsType(x)

## S4 replacement method for signature 'AlignmentsExperimentSet'
rveReadsType(x) <- value

## S4 method for signature 'AlignmentsExperimentSet'
unassignedData(x)

## S4 replacement method for signature 'AlignmentsExperimentSet'
unassignedData(x) <- value

## S4 method for signature 'AlignmentsExperimentSet'
readCounts(x)

## S4 replacement method for signature 'AlignmentsExperimentSet'
readCounts(x) <- value

## S4 method for signature 'AlignmentsExperimentSet'
experimentData(x)

## S4 replacement method for signature 'AlignmentsExperimentSet'
experimentData(x) <- value

## S4 method for signature 'AlignmentsExperimentSet'
barcodeData(x)

## S4 replacement method for signature 'AlignmentsExperimentSet'
barcodeData(x) <- value

## S4 method for signature 'AlignmentsExperimentSet'
unassignedCount(x)

## S4 method for signature 'AlignmentsExperimentSet'
assignedCount(x)

## S4 method for signature 'AlignmentsExperimentSet'
names(x)

## S4 method for signature 'AlignmentsExperimentSet'
c(x, ...)

## S4 method for signature 'AlignmentsExperimentSet,numeric,missing,missing'
x[i, j, ..., drop = TRUE]

## S3 method for class 'AlignmentsExperimentSet'
as.list(x, ...)

## S4 method for signature 'AlignmentsExperimentSet'
x$name

## S4 method for signature 'AlignmentsExperimentSet'
writeAlignments(x, file = '', aln_format = "txt")

## S4 method for signature 'AlignmentsExperimentSet'
lookupAlignment(x, ID, read_id = 1)

## S4 method for signature 'AlignmentsExperimentSet'
extractEvents(object, use_parallel = FALSE)
### Arguments

... pass any number of AlignmentsExperimentSet objects, make sure experiment IDs can be unique after merging

- **x, object**: (AlignmentsExperimentSet)
  - Value: Represents assignment values for setter methods.

- **i, j, name, drop**: (numeric, missing, character, missing) AlignmentsExperimentSet object can be subsetted using names of the experiments eg. `x$name` or `x[i]`, resulting in AlignmentsExperimentSet object that has only one experiment. During this sub-setting, values of unassignedData and barcodeData are dropped.

- **file**: (connection or string) Destination file. When empty, defaults to standard output.

- **aln_format**: ("txt" or "fasta") Specifies format of the file.

- **ID**: (string) Experiment Identifier

- **read_id**: (numeric) Read Identifier. Reads are sorted by frequency. Defaults to 1, most abundant read.

- **use_parallel**: (boolean) When using `extractEvents` you can use multicore back-end through `register` as this is very slow function (despite vectorization).

### Value

depending on the function used

### Slots

- **fwdReads, rveReads** (list) Named list where each element is of class `PairwiseAlignmentsSingleSubject`. Names correspond to the experiment ID. Contains alignments of reads against amplicons.

- **fwdReadsType, rveReadsType** (list) Named list where each element is of logical vector, so far TRUE corresponds to HDR events. Names correspond to the experiment ID. Contains type of read - HDR/NHEJ.

- **readCounts** (list) Named list where each element is numeric vector that describes how many reads are compressed into unique representation before alignment in `fwdReads` and/or `rveReads`.

- **unassignedData** (data.frame) Contains reads that failed to be assigned to any of the experiments. Alignment of forward against reverse reads may give hint whether these reads are compromised in any way.

- **experimentData** (data.frame) Expands on configuration file and provides information about cut rates, frameshifts, PRIMER DIMER detection etc. Each row corresponds to experiment ID.

- **barcodeData** (data.frame) Information that is gathered on the barcode level is gathered in this data.frame, mainly quality filtering statistics.

### View alignments

Write out all alignments in "fasta" or "txt" format:

```r
code
writeAlignments(x, file = "", aln_format = "txt")
```

Write out human readable alignments for given experiment and read_id:

```r
code
lookupAlignment(x, ID, read_id = 1)
```
Coercion based on events

Coerce to data.frame compatible with GRanges:
as.data.frame(x)

Examples

```r
exampleAlignments <- pwalign::pairwiseAlignment(
  Biostrings::DNAStringSet(c("ACTGACTG", "CGACGACG"), "ACGTACGTACGT"),
  new("AlignmentsExperimentSet",
    fwdReads = list(ID_1 = exampleAlignments, ID_2 = exampleAlignments),
    rveReads = list(ID_1 = exampleAlignments, ID_2 = exampleAlignments),
    fwdReadsType = list(ID_1 = c(FALSE, FALSE), ID_2 = c(FALSE, FALSE)),
    rveReadsType = list(ID_1 = c(FALSE, FALSE), ID_2 = c(FALSE, FALSE)),
    readCounts = list(ID_1 = c(2, 20), ID_2 = c(30, 100)),
    unassignedData = NULL,
    experimentData = data.frame(ID = c("ID_1", "ID_2"),
      Barcode = c("B1", "B1"),
      whatever = c(50, 100)),
    barcodeData = data.frame(Barcode = "B1", statistic1 = 100))
# Coercion
extractEvents(AlignmentsExperimentSet())
GenomicRanges::GRanges(extractEvents(AlignmentsExperimentSet()))
```

### alphabetQuality
This filters out sequences which have nonstandard nucleotides.

**Description**

This filters out sequences which have nonstandard nucleotides.

**Usage**

```r
alphabetQuality(reads, batch_size = 1e+07)
```

**Arguments**

- `reads` (ShortRead object) Loaded reads from fastq.
- `batch_size` (numeric) How many reads to process at a time.

**Value**

(boolean) Logical vector with the valid rows as TRUE.
amplican
Automated analysis of CRISPR experiments.

Description

Main goals:

1. Flexible pipeline for analysis of the CRISPR Mi-Seq or Hi-Seq data.
2. Compatible with GRanges and data.table style.
3. Precise quantification of mutation rates.
4. Prepare automatic reports as .Rmd files that are flexible and open for manipulation.
5. Provide specialized plots for deletions, insertions, mismatches, variants, heterogeneity of the reads.

Details

To learn more about amplican, start with the vignettes: `browseVignettes(package = "amplican")`

Author(s)

Maintainer: Eivind Valen <eivind.valen@gmail.com> [copyright holder]
Authors:

- Kornel Labun <kornel.labun@gmail.com>

See Also

Useful links:

- [https://github.com/valenlab/amplican](https://github.com/valenlab/amplican)
- Report bugs at [https://github.com/valenlab/amplican/issues](https://github.com/valenlab/amplican/issues)

amplicanAlign
Align reads to amplicons.

Description

`amplicanAlign` takes a configuration files, fastq reads and output directory to prepare alignments and summary. It uses global Needleman-Wunsch algorithm with parameters optimized for CRISPR experiment. After alignments, object of `AlignmentsExperimentSet` is returned that allows for coercion into GRanges (plus is for forward and minus for reverse reads). It is also possible to output alignments in other, additional formats.
amplicanAlign

Usage

```r
amplicanAlign(
    config,
    fastq_folder,
    use_parallel = FALSE,
    average_quality = 30,
    min_quality = 20,
    filter_n = FALSE,
    batch_size = 1e+06,
    scoring_matrix = Biostrings::nucleotideSubstitutionMatrix(match = 5, mismatch = -4,
                                                            baseOnly = FALSE, type = "DNA"),
    gap_opening = 25,
    gap_extension = 0,
    fastqfiles = 0.5,
    primer_mismatch = 0,
    donor_mismatch = 3,
    donor_strict = FALSE,
)
```

Arguments

config (string) The path to your configuration file. For example: `system.file("extdata", "config.txt", package = "amplican")`. Configuration file can contain additional columns, but first 11 columns have to follow the example config specification.

fastq_folder (string) Path to FASTQ files. If not specified, FASTQ files should be in the same directory as config file.

use_parallel (boolean) Set to TRUE, if you have registered multicore back-end.

average_quality (numeric) The FASTQ file have a quality for each nucleotide, depending on sequencing technology there exist many formats. This package uses `readFastq` to parse the reads. If the average quality of the reads fall below value of average_quality then sequence is filtered. Default is 0.

min_quality (numeric) Similar as in average_quality, but depicts the minimum quality for ALL nucleotides in given read. If one of nucleotides has quality BELOW min_quality, then the sequence is filtered. Default is 20.

filter_n (boolean) Whether to filter out reads containing N base.

batch_size (numeric) How many reads to analyze at a time? Needed for filtering of large fastq files.

scoring_matrix (matrix) Default is 'NUC44'. Pass desired matrix using `nucleotideSubstitutionMatrix`.

gap_opening (numeric) The opening gap score.

gap_extension (numeric) The gap extension score.

fastqfiles (numeric) Normally you want to use both FASTQ files. But in some special cases, you may want to use only the forward file, or only the reverse file. Possible options:
amplicanAlign

- 0 Use both FASTQ files.
- 0.5 Use both FASTQ files, but only for one of the reads (forward or reverse) is required to have primer perfectly matched to sequence - eg. use when reverse reads are trimmed of primers, but forward reads have forward primer in the sequence.
- 1 Use only the forward FASTQ file.
- 2 Use only the reverse FASTQ file.

primer_mismatch

(numeric) Decide how many mismatches are allowed during primer matching of the reads, that groups reads by experiments. When primer_mismatch = 0 no mismatches are allowed, which can increase number of unassigned read.

donor_mismatch

(numeric) How many events of length 1 (mismatches, deletions and insertions of length 1) are allowed when aligning toward the donor template. This parameter is only used when donor template is specified. The higher the parameter the less strict will be algorithm accepting read as HDR. Set to 0 if only perfect alignments to the donor template marked as HDR, unadvised due to error rate of the sequencers.

donor_strict

(logical) Applies more strict algorithm for HDR detection. Only these reads that have all of the donor events will count as HDR. Tolerates ‘donor_mismatch’ level of noise, but no indels are allowed. Use this when your reads should span over the whole window of the donor events. Might be more time consuming.

Value

(AlignmentsExperimentSet) Check AlignmentsExperimentSet class for details. You can use lookupAlignment to examine alignments visually.

See Also

Other analysis steps: amplicanConsensus(), amplicanFilter(), amplicanMap(), amplicanNormalize(), amplicanOverlap(), amplicanPipeline(), amplicanPipelineConservative(), amplicanReport(), amplicanSummarize()

Examples

# path to example config file
config <- system.file("extdata", "config.csv", package = "amplican")
# path to example fastq files
fastq_folder <- system.file("extdata", package = "amplican")
aln <- amplicanAlign(config, fastq_folder)
aln
amplicanConsensus

Extract consensus out of forward and reverse events.

Description

When forward and reverse reads are in agreement on the events (e.g. deletion) amplicanConsensus will mark forward event as TRUE indicating that he represents consensus. In cases where forward and reverse read agree only partially, for example, they share the same start of the deletion, but they have different end amplicanConsensus will pick the version of read with higher alignment score, in situation where both of the reads overlap expected cut site, otherwise both events will be rejected and marked FALSE. When there are events only on one of the strands they will be rejected.

Usage

amplicanConsensus(aln, cfgT, overlaps = "overlaps", promiscuous = TRUE)

Arguments

- aln: (data.frame) Contains relevant events in GRanges style.
- cfgT: (data.frame) Should be table containing at least positions of primers in the amplicons and their identifiers
- overlaps: (character) Specifies which metadata column of aln indicates which events are overlapping expected cut site.
- promiscuous: (boolean) Allows to relax consensus rules. When TRUE will allow Indels that are not confirmed by the other strand (when both are used).

Details

In situation where you have only forward or only reverse reads don’t use this function and assign all TRUE to all of your events.

Consensus out of the forward + reverse reads is required for amplicanSummary, and amplicanConsensus requires amplicanOverlap.

Value

(bolean vector) Where TRUE means that given event represents consensus out of forward and reverse reads.

See Also

Other analysis steps: amplicanAlign(), amplicanFilter(), amplicanMap(), amplicanNormalize(), amplicanOverlap(). amplicanPipeline(). amplicanPipelineConservative(). amplicanReport(), amplicanSummarize()
Examples

```r
file_path <- system.file("test_data", "test_aln.csv", package = "amplican")
aln <- data.table::fread(file_path)
cfgT <- data.table::fread(
    system.file("test_data", "test_cfg.csv", package = "amplican")))
all(aln$consensus == amplicanConsensus(aln, cfgT))
```

Description

Very often alignments return deletions that are not real deletions, but rather artifact of incomplete reads eg.:

```
ACTGAAAA------- <- this "deletion" should be filtered
ACTG----ACTGACTG
```

We call them Events Overlapping Primers and filter them together with reads that are potentially PRIMER DIMERS. This filter will also remove all events coming from reads with low alignment score - potential Off-targets.

Usage

```r
amplicanFilter(aln, cfgT, PRIMER_DIMER)
```

Arguments

- `aln` (data.frame) Should contain events from alignments in GRanges style with columns eg. seqnames, width, start, end.
- `cfgT` (data.frame) Needs columns Forward_Primer, ReversePrimer and Amplicon.
- `PRIMER_DIMER` (numeric) Value specifying buffer for PRIMER DIMER detection. For a given read it will be recognized as PRIMER DIMER when alignment will introduce gap of size bigger than:

  \[
  \text{length of amplicon} - (\text{lengths of PRIMERS} + \text{PRIMER_DIMER value})
  \]

Value

(aln) Reduced by events classified as PRIMER DIMER or overlapping primers.

See Also

- `findPD` and `findEOP`

Other analysis steps: `amplicanAlign()`, `amplicanConsensus()`, `amplicanMap()`, `amplicanNormalize()`, `amplicanOverlap()`, `amplicanPipeline()`, `amplicanPipelineConservative()`, `amplicanReport()`, `amplicanSummarize()`
amplicanMap

Examples

```r
file_path <- system.file("extdata", "results", "alignments", "raw_events.csv", package = "amplican")
aln <- data.table::fread(file_path)
cfgT <- data.table::fread(
  system.file("extdata", "results", "config_summary.csv", package = "amplican"))
amplicanFilter(aln, cfgT, 30)
```

amplicanMap  
*Map events to their respective relative coordinates specified with UP-PER case.*

Description

Translate coordinates of `GRanges` events so that they can be relative to the amplicon. As point zero we assume first left sided UPPERCASE letter in the amplicon. Be wary that events for amplicons without expected cut sites are filtered. Don’t use this function, if you don’t have expected cut sites specified and don’t use any of the metaplots.

Usage

```r
amplicanMap(aln, cfgT)
```

Arguments

- **aln** (data.frame) List of events to map to the relative coordinates.
- **cfgT** (data.frame) config table

Value

`(GRanges)` Same as events, but the coordinates are relative to the expected cut sites.

See Also

Other analysis steps: `amplicanAlign()`, `amplicanConsensus()`, `amplicanFilter()`, `amplicanNormalize()`, `amplicanOverlap()`, `amplicanPipeline()`, `amplicanPipelineConservative()`, `amplicanReport()`, `amplicanSummarize()`

Examples

```r
# example config
config <- read.csv(system.file("extdata", "config.csv", package = "amplican"))
# example events
events <- read.csv(system.file("extdata", "results", "alignments", "raw_events.csv", package = "amplican"))
```
# make events relative to the UPPER case
amplicanMap(events, config)

## amplicanNormalize

Remove events that can be found in Controls.

### Description
This function can adjust events for small differences between known annotations (amplicon sequences) and real DNA of the strain that was sequenced. Events from the control are grouped by add and their frequencies are calculated in respect to number of total reads in that groups. In next step events from the control are filtered according to min_freq, all events below are treated as sequencing errors and rejected. Finally, all events that can be found in treatment group that find their exact match (by non skipped columns) in control group are removed. All events from control group are returned back.

### Usage
amplicanNormalize(
  aln,
  cfgT,
  add = c("guideRNA", "Group"),
  skip = c("counts", "score", "seqnames", "read_id", "strand", "overlaps", "consensus"),
  min_freq = 0.01
)

### Arguments
- **aln** (data.frame) Contains events from alignments.
- **cfgT** (data.frame) Config table with information about experiments.
- **add** (character vector) Columns from cfgT that should be included in event table for normalization matching. Defaults to c("guideRNA", "Group") , which means that only those events created by the same guideRNA in the same Group will be removed if found in Control.
- **skip** (character vector) Specifies which columns of aln to skip.
- **min_freq** (numeric) All events from control group below this frequency will be not included in filtering. Use this to filter out background noise and sequencing errors.

### Value
(data.frame) Same as aln, but events are normalized. Events from Control are not changed. Additionally columns from add are added to the data.frame.
See Also

Other analysis steps: amplicanAlign(), amplicanConsensus(), amplicanFilter(), amplicanMap(), amplicanOverlap(), amplicanPipeline(), amplicanPipelineConservative(), amplicanReport(), amplicanSummarize()

Examples

```r
aln <- data.frame(seqnames = 1:5, start = 1, end = 2, width = 2,
                   counts = 101:105)
cfgT <- data.frame(ID = 1:5, guideRNA = rep("ACTG", 5),
                   Reads_Filtered = c(2, 2, 3, 3, 4),
                   Group = c("A", "A", "B", "B", "B"),
                   Control = c(TRUE, FALSE, TRUE, FALSE, FALSE))
# all events are same as in the control group, therefore are filtered out
# events from control groups stay
amplicanNormalize(aln, cfgT)
# events that are different from control group are preserved
aln[2, "start"] <- 3
amplicanNormalize(aln, cfgT)
```

Description

To determine which deletions, insertions and mismatches (events) are probably created by CRISPR we check whether they overlap expected cut sites. Expected cut sites should be specified in UPPER CASE letters in the amplicon sequences.

Usage

```
amplicanOverlap(aln, cfgT, cut_buffer = 5, relative = FALSE)
```

Arguments

- **aln** (data.frame) Contains relevant events in GRanges style.
- **cfgT** (data.frame) Contains amplicon sequences.
- **cut_buffer** (numeric) Number of bases that should expand 5’ and 3’ of the specified expected cut sites.
- **relative** (boolean) Sets whether events are relative to the position of the target site.

Value

(bolean vector) Where TRUE means that given event overlaps cut site.
amplicanPipeline

Wraps main package functionality into one function.

Description

amplicanPipeline is convenient wrapper around all functionality of the package with the most robust settings. It will generate all results in the result_folder and also knit prepared reports into 'reports' folder.

Usage

```r
amplicanPipeline(
  config,
  fastq_folder,
  results_folder,
  knit_reports = TRUE,
  write_alignments_format = "None",
  average_quality = 30,
  min_quality = 0,
  filter_n = FALSE,
  batch_size = 1e+07,
  use_parallel = FALSE,
  scoring_matrix = Biostrings::nucleotideSubstitutionMatrix(match = 5, mismatch = -4,
    baseOnly = FALSE, type = "DNA"),
  gap_opening = 25,
  gap_extension = 0,
  fastqfiles = 0.5,
  primer_mismatch = 2,
  donor_mismatch = 3,
  donor_strict = FALSE,
  PRIMER_DIMER = 30,
  event_filter = TRUE,
  cut_buffer = 5,
)`
amplicanPipeline

promiscuous_consensus = TRUE,
normalize = c("guideRNA", "Group"),
min_freq = min_freq_default,
continue = TRUE
)

Arguments

config (string) The path to your configuration file. For example: system.file("extdata", "config.txt", package = "amplican"). Configuration file can contain additional columns, but first 11 columns have to follow the example config specification.

fastq_folder (string) Path to FASTQ files. If not specified, FASTQ files should be in the same directory as config file.

results_folder (string) Where do you want to store results? The package will create files in that folder so make sure you have writing permissions.

knit_reports (boolean) whether function should "knit" all reports automatically for you (it is time consuming, be patient), when false reports will be prepared, but not knitted

write_alignments_format (character vector) Whether amplicanPipeline should write alignments results to separate files. Alignments are also always saved as .rds object of AlignmentsExperimentSet class. Possible options are:

• "fasta" outputs alignments in fasta format where header indicates experiment ID, read id and number of reads
• "txt" simple format, read information followed by forward read and amplicon sequence followed by reverse read with its amplicon sequence eg.:

   ID: ID_1 Count: 7
   ACTGAAAAA--------
   ACTG----ACTGACTG

   ------G~ACTG
   ACTGACTGACTG

• "None" Don’t write any alignments to files.
• c("fasta", "txt") There are also possible combinations of above formats, pass a vector to get alignments in multiple formats.

average_quality (numeric) The FASTQ file have a quality for each nucleotide, depending on sequencing technology there exist many formats. This package uses readFastq to parse the reads. If the average quality of the reads fall below value of average_quality then sequence is filtered. Default is 0.

min_quality (numeric) Similar as in average_quality, but depicts the minimum quality for ALL nucleotides in given read. If one of nucleotides has quality BELOW min_quality, then the sequence is filtered. Default is 20.

filter_n (boolean) Whether to filter out reads containing N base.
amplican Pipeline

**batch_size**  
(numeric) How many reads to analyze at a time? Needed for filtering of large fastq files.

**use_parallel**  
(boolean) Set to TRUE, if you have registered multicore back-end.

**scoring_matrix**  
(matrix) Default is ‘NUC44’. Pass desired matrix using `nucleotideSubstitutionMatrix`.

**gap_opening**  
(numeric) The opening gap score.

**gap_extension**  
(numeric) The gap extension score.

**fastqfiles**  
(numeric) Normally you want to use both FASTQ files. But in some special cases, you may want to use only the forward file, or only the reverse file. Possible options:

- 0 Use both FASTQ files.
- 0.5 Use both FASTQ files, but only for one of the reads (forward or reverse) is required to have primer perfectly matched to sequence - eg. use when reverse reads are trimmed of primers, but forward reads have forward primer in the sequence.
- 1 Use only the forward FASTQ file.
- 2 Use only the reverse FASTQ file.

**primer_mismatch**  
(numeric) Decide how many mismatches are allowed during primer matching of the reads, that groups reads by experiments. When `primer_mismatch = 0` no mismatches are allowed, which can increase number of unassigned read.

**donor_mismatch**  
(numeric) How many events of length 1 (mismatches, deletions and insertions of length 1) are allowed when aligning toward the donor template. This parameter is only used when donor template is specified. The higher the parameter the less strict will be algorithm accepting read as HDR. Set to 0 if only perfect alignments to the donor template marked as HDR, unadvised due to error rate of the sequencers.

**donor_strict**  
(logical) Applies more strict algorithm for HDR detection. Only these reads that have all of the donor events will count as HDR. Tolerates ‘donor_mismatch’ level of noise, but no indels are allowed. Use this when your reads should span over the whole window of the donor events. Might be more time consuming.

**PRIMER_DIMER**  
(numeric) Value specifying buffer for PRIMER DIMER detection. For a given read it will be recognized as PRIMER DIMER when alignment will introduce gap of size bigger than:

\[
\text{length of amplicon} - (\text{lengths of PRIMERS} + \text{PRIMER_DIMER value})
\]

**event_filter**  
(logical) Whether detection of offtarget reads, should be enabled.

**cut_buffer**  
The number of bases by which extend expected cut sites (specified as UPPER case letters in the amplicon) in 5’ and 3’ directions.

**promiscuous_consensus**  
(boolean) Whether rules of `amplicanConsensus` should be promiscuous. When promiscuous, we allow indels that have no confirmation on the other strand.

**normalize**  
(character vector) If column 'Control' in config table has all FALSE/0 values then normalization is skipped. Otherwise, normalization is strict, which means events that are found in 'Control' TRUE group will be removed in 'Control' FALSE group. This parameter by default uses columns 'guideRNA' and 'Group'
to impose additional restrictions on normalized events eg. only events created by the same 'guideRNA' in the same 'Group' will be normalized.

min_freq (numeric) All events below this frequency are treated as sequencing errors and rejected. This parameter is used during normalization through `amplicanNormalize`.

continue (boolean) Default TRUE, decides whether to continue failed ampliCan runs. In case of FALSE, all contents in 'results' folder will be removed.

Value

(invisible) results_folder path

See Also

Other analysis steps: `amplicanAlign()`, `amplicanConsensus()`, `amplicanFilter()`, `amplicanMap()`, `amplicanNormalize()`, `amplicanOverlap()`, `amplicanPipelineConservative()`, `amplicanReport()`, `amplicanSummarize()`

Examples

```r
# path to example config file
config <- system.file("extdata", "config.csv", package = "amplican")
# path to example fastq files
fastq_folder <- system.file("extdata", package = "amplican")
# output folder
results_folder <- tempdir()

#full analysis, not knitting files automatically
amplicanPipeline(config, fastq_folder, results_folder, knit_reports = FALSE)
```

Description

`amplicanPipelineConservative` wraps main package functionality into one function.

`amplicanPipelineIndexHopping` is identical as `amplicanPipeline` except that default `min_freq` threshold is set to 0.15. Setting this threshold higher will decrease risks of inadequate normalization in cases of potential Index Hopping, potentially decreasing precision of true editing rate calling. Index Hopping can be mitigated with use of unique dual indexing pooling combinations. However, in cases when you might expect Index Hopping to occur you should use this function instead of `amplicanPipeline`.

amplicanPipelineConservative

Wraps main package functionality into one function.
Usage

```r
amplicanPipelineConservative(
  config,
  fastq_folder,
  results_folder,
  knit_reports = TRUE,
  write_alignments_format = "None",
  average_quality = 30,
  min_quality = 0,
  filter_n = FALSE,
  batch_size = 1e+07,
  use_parallel = FALSE,
  scoring_matrix = Biostrings::nucleotideSubstitutionMatrix(match = 5, mismatch = -4,
    baseOnly = FALSE, type = "DNA"),
  gap_opening = 25,
  gap_extension = 0,
  fastqfiles = 0.5,
  primer_mismatch = 2,
  donor_mismatch = 3,
  donor_strict = FALSE,
  PRIMER_DIMER = 30,
  event_filter = TRUE,
  cut_buffer = 5,
  promiscuous_consensus = TRUE,
  normalize = c("guideRNA", "Group"),
  min_freq = min_freq_default,
  continue = TRUE
)
```

Arguments

- `config` (string) The path to your configuration file. For example: `system.file("extdata", "config.txt", package = "amplican")`. Configuration file can contain additional columns, but first 11 columns have to follow the example config specification.
- `fastq_folder` (string) Path to FASTQ files. If not specified, FASTQ files should be in the same directory as config file.
- `results_folder` (string) Where do you want to store results? The package will create files in that folder so make sure you have writing permissions.
- `knit_reports` (boolean) whether function should "knit" all reports automatically for you (it is time consuming, be patient), when false reports will be prepared, but not knitted
- `write_alignments_format` (character vector) Whether `amplicanPipeline` should write alignments results to separate files. Alignments are also always saved as .rds object of `AlignmentsExperimentSet` class. Possible options are:
  - "fasta" outputs alignments in fasta format where header indicates experiment ID, read id and number of reads
• "txt" simple format, read information followed by forward read and amplicon sequence followed by reverse read with its amplicon sequence eg.:

ID: ID_1  Count: 7  
ACTGAAAAA--------  
ACTG-----ACTGACTG  
-------G-ACTG  
ACTGACTGACTG

• "None" Don’t write any alignments to files.
• c("fasta", "txt") There are also possible combinations of above formats, pass a vector to get alignments in multiple formats.

average_quality
(numeric) The FASTQ file have a quality for each nucleotide, depending on sequencing technology there exist many formats. This package uses readFastq to parse the reads. If the average quality of the reads fall below value of average_quality then sequence is filtered. Default is 0.

min_quality
(numeric) Similar as in average_quality, but depicts the minimum quality for ALL nucleotides in given read. If one of nucleotides has quality BELOW min_quality, then the sequence is filtered. Default is 20.

filter_n
(boolean) Whether to filter out reads containing N base.

batch_size
(numeric) How many reads to analyze at a time? Needed for filtering of large fastq files.

use_parallel
(boolean) Set to TRUE, if you have registered multicore back-end.

scoring_matrix
(matrix) Default is ‘NUC44’. Pass desired matrix using nucleotideSubstitutionMatrix.

gap_opening
(numeric) The opening gap score.

gap_extension
(numeric) The gap extension score.

fastqfiles
(numeric) Normally you want to use both FASTQ files. But in some special cases, you may want to use only the forward file, or only the reverse file. Possible options:

• 0 Use both FASTQ files.
• 0.5 Use both FASTQ files, but only for one of the reads (forward or reverse) is required to have primer perfectly matched to sequence - eg. use when reverse reads are trimmed of primers, but forward reads have forward primer in the sequence.
• 1 Use only the forward FASTQ file.
• 2 Use only the reverse FASTQ file.

primer_mismatch
(numeric) Decide how many mismatches are allowed during primer matching of the reads, that groups reads by experiments. When primer_mismatch = 0 no mismatches are allowed, which can increase number of unassigned read.

donor_mismatch
(numeric) How many events of length 1 (mismatches, deletions and insertions of length 1) are allowed when aligning toward the donor template. This parameter is only used when donor template is specified. The higher the parameter the
amplicanPipelineConservative

less strict will be algorithm accepting read as HDR. Set to 0 if only perfect alignments to the donor template marked as HDR, unadvised due to error rate of the sequencers.

donor_strict  (logical) Applies more strict algorithm for HDR detection. Only these reads that have all of the donor events will count as HDR. Tolerates 'donor_mismatch' level of noise, but no indels are allowed. Use this when your reads should span over the whole window of the donor events. Might be more time consuming.

PRIMER_DIMER  (numeric) Value specifying buffer for PRIMER DIMER detection. For a given read it will be recognized as PRIMER DIMER when alignment will introduce gap of size bigger than:

\[ \text{length of amplicon} - (\text{lengths of PRIMERS} + \text{PRIMER_DIMER value}) \]

event_filter  (logical) Whether detection of offtarget reads, should be enabled.

cut_buffer  The number of bases by which extend expected cut sites (specified as UPPER case letters in the amplicon) in 5’ and 3’ directions.

promiscuous_consensus  (boolean) Whether rules of amplicanConsensus should be promiscuous. When promiscuous, we allow indels that have no confirmation on the other strand.

normalize  (character vector) If column 'Control' in config table has all FALSE/0 values then normalization is skipped. Otherwise, normalization is strict, which means events that are found in 'Control' TRUE group will be removed in 'Control' FALSE group. This parameter by default uses columns 'guideRNA' and 'Group' to impose additional restrictions on normalized events eg. only events created by the same 'guideRNA' in the same 'Group' will be normalized.

min_freq  (numeric) All events below this frequency are treated as sequencing errors and rejected. This parameter is used during normalization through amplicanNormalize.

continue  (boolean) Default TRUE, decides whether to continue failed ampliCan runs. In case of FALSE, all contents in 'results' folder will be removed.

Details

result_folder and also knit prepared reports into 'reports' folder.

Value

(invisible) results_folder path

See Also

Other analysis steps: amplicanAlign(), amplicanConsensus(), amplicanFilter(), amplicanMap(), amplicanNormalize(), amplicanOverlap(), amplicanPipeline(), amplicanReport(), amplicanSummarize()
amplicanReport

Prepare reports as .Rmd files.

Description

amplicanReport takes a configuration file, fastq reads and output directory to prepare summaries as an editable .Rmd file. You can specify whether you want to make summaries based on ID, Barcode, Group or even guideRNA and Amplicon. This function automatically knits all reports after creation. If you want to postpone knitting and edit reports, use .Rmd templates to create your own version of reports instead of this function.

Usage

```r
amplicanReport(
  results_folder,
  levels = c("id", "barcode", "group", "guide", "amplicon", "summary"),
                   "amplicon_report", "index"),
  cut_buffer = 5,
  xlab_spacing = 4,
  top = 5,
  knit_reports = TRUE
)
```

Arguments

- **results_folder** (string) Folder containing results from the `amplicanAlign` function, do not change names of the files.
- **levels** (vector) Possible values are: "id", "barcode", "group", "guide", "amplicon", "summary". You can also input more than one value eg. c("id", "barcode") will create two separate reports for each level.
- **report_files** (vector) You can supply your own names of the files. For each of the levels there has to be one file name. Files are created in current working directory by default.
- **cut_buffer** (numeric) Default 5. A number of bases that is used around the specified cut site.
- **xlab_spacing** (numeric) Default is 4. Spacing of the ticks on the x axis of plots.
- **top** (numeric) Default is 5. How many of the top most frequent unassigned reads to report? It is only relevant when you used forward and reverse reads. We align them to each other as we could not specify correct amplicon.
- **knit_reports** (boolean) Whether to knit reports automatically.

Value

- (string) Path to the folder with results.
See Also

Other analysis steps: `amplicanAlign()`, `amplicanConsensus()`, `amplicanFilter()`, `amplicanMap()`, `amplicanNormalize()`, `amplicanOverlap()`, `amplicanPipeline()`, `amplicanPipelineConservative()`, `amplicanSummarize()`

Examples

```r
results_folder <- tempdir()
amplicanReport(results_folder, report_files = file.path(results_folder, 
  c("id_report", 
    "barcode_report", 
    "group_report", 
    "guide_report", 
    "amplicon_report", 
    "index")),

  knit_reports = FALSE)

amplicanSummarize(aln, cfgT)
```

Description

Before using this function make sure events are filtered to represent consensus with `amplicanConsensus`, if you use both forward and reverse reads. If you want to calculate metrics over expected cut site, filter events using `amplicanOverlap`.

Usage

```r
amplicanSummarize(aln, cfgT)
```

Arguments

- `aln` (data.frame) Contains events from the alignments.
- `cfgT` (data.frame) Config file with the experiments details.

Details

Adds columns to `cfgT`:

- `ReadsCut` Count of reads with deletions overlapping expected cut site.
- `Reads_Frameshifted` Count of reads with frameshift overlapping expected cut site.

Value

(data.frame) As `cfgT`, but with extra columns.
amplican_print_reads

Pretty print forward and reverse reads aligned to each other.

Description
Usefull and needed for barcode reports.

Usage
amplican_print_reads(forward, reverse)

Arguments
forward (character or vector of characters) Forward reads.
reverse (character or vector of characters) Will be reverse complemented before alignment.

Value
Vector with alignments ready to be printed.

Examples
# load example data
unassigned_file <- system.file('extdata', 'results', 'alignments',
    'unassigned_reads.csv', package = 'amplican')
unassigned <- data.table::setDF(data.table::fread(unassigned_file))
# sort by frequency
unassigned <- unassigned[order(unassigned$BarcodeFrequency,
    decreasing = TRUE),]
# print alignment of most frequent unassigned reads
```r
cat(amplican_print_reads(unassigned[1, 'Forward'],
                        unassigned[1, 'Reverse']),
    sep = "\n")
```

---

## assignedCount

*Get count of assigned reads.*

### Description

Get count of assigned reads.

### Usage

```r
assignedCount(x)
```

### Arguments

- `x` *(AlignmentsExperimentSet)*

### Value

*(numeric)*

### Examples

```r
file_path <- system.file("extdata", "results", "alignments",
                        "AlignmentsExperimentSet.rds", package = "amplican")
aln <- readRDS(file_path)
writeAlignments(aln, file.path(tempdir(), "aln.txt"))
```

---

## barcodeData

*Barcode data.*

### Description

Get barcode data.frame with information on the barcode level.

### Usage

```r
barcodeData(x)
```

### Arguments

- `x` *(AlignmentsExperimentSet)*
Value

(data.tableOrNULL)

Examples

```r
file_path <- system.file("extdata", "results", "alignments", 
    "AlignmentsExperimentSet.rds", package = "amplican")
aln <- readRDS(file_path)
barcodeData(aln)
```

Description

Set barcode data.frame with information on the barcode level.

Usage

`barcodeData(x) <- value`

Arguments

- `x` (AlignmentsExperimentSet)
- `value` (data.frame)

Value

(AlignmentsExperimentSet)

Examples

```r
file_path <- system.file("extdata", "results", "alignments", 
    "AlignmentsExperimentSet.rds", package = "amplican")
aln <- readRDS(file_path)
barcodeData(aln) <- barcodeData(aln) # replace with the same values as before
```
checkConfigFile

Pre-process a config file and checks that everything is in order.

Description

Its takes care of the following: No IDs are duplicated. Every combination of barcode, forward primer and reverse primer is unique. Each barcode has unique forward reads file and reverse read files. Checks that the read files exist with read access.

Usage

checkConfigFile(configTable, fastq_folder)

Arguments

configTable (data.frame) Config file.
fastq_folder (string) Path to fastq folder.

Value

(boolean) TRUE, If anything goes wrong stops and prints error.

checkFileWriteAccess

Checks if the given directory exist and can be written to.

Description

Checks if the given directory exist and can be written to.

Usage

checkFileWriteAccess(filePath)

Arguments

filePath (string) A string the path to the file.

Value

(invisible) TRUE, Stop if no access.
checkPrimers

Checks if the forward and reverse primer are in the amplicon and where they are located.

Description
Checks if the forward and reverse primer are in the amplicon and where they are located.

Usage
checkPrimers(configTable, fastqfiles)

Arguments
- configTable (data.frame) A data frame of config file.
- fastqfiles (numeric) Which primers are important.

Value
configTable (data.frame) A data frame of config file with additional fields for start locations of the primers

checkTarget

Checks if the guideRNA is in the amplicon.

Description
Checks if the guideRNA is in the amplicon.

Usage
checkTarget(configTable)

Arguments
- configTable (data.frame) data frame of config file

Value
(boolean vector) Prints warning when some guides can’t be found.
cigarsToEvents  

Transform extended CIGAR strings into GRanges.

**Description**

Transform extended CIGAR strings into GRanges representation with events of deletions, insertions and mismatches.

**Usage**

```r
cigarsToEvents(
  cigars,
  aln_pos_start,
  query_seq,
  ref,
  read_id,
  mapq,
  seqnames,
  strands,
  counts
)
```

**Arguments**

- **cigars** (character) Extended CIGARS.
- **aln_pos_start** (integer) Pos of CIGARS.
- **query_seq** (character) Aligned query sequences.
- **ref** (character) Reference sequences used for alignment.
- **read_id** (numeric) Read id for assignment for each of the CIGARS.
- **mapq** (numeric) Maping scores.
- **seqnames** (character) Names of the sequences, potentially ids of the reference sequences.
- **strands** (character) Strands to assign.
- **counts** (integer) Vector of cigar counts, if data collapsed.

**Value**

(\texttt{GRanges}) Same as events.
### comb_along

*Generate all combinations along string exchanging m characters at a time with dictionary letters.*

**Description**

Generate all combinations along string `seq` swapping `m` characters at a time with letters defined in dictionary `letters`. Allows, for instance, to create a list of possible primers with two mismatches.

**Usage**

```r
comb_along(seq, m = 2, letters = c("A", "C", "T", "G"))
```

**Arguments**

- `seq` (character): input character to permute
- `m` (integer): number of elements to permute at each step
- `letters` (character vector): dictionary source for combinations of elements

**Value**

(character vector) all unique combinations of permutated string

**Examples**

- `comb_along("AC")`
- `comb_along("AAA", 1)`
- `comb_along("AAA")`
- `comb_along("AAA", 3)`
- `comb_along("AAAAAAAAAAA")`

### cumsumw

*Cumulative sum to calculate shift*

**Description**

Cumulative sum to calculate shift

**Usage**

```r
cumsumw(x)
```

**Arguments**

- `x` ([IRanges](https://bioconductor.org/packages/IRanges))
Value

(numeric vector)

decode

Get codons for given string - translate

Description

Get codons for given string - translate

Usage

decode(x)

Arguments

x (string)

Value

(string) codons

defGR

Helper to construct GRanges with additional metadata columns.

Description

Helper to construct GRanges with additional metadata columns.

Usage

defGR(
    x,
    ID,
    score,
    strand_info = "+",
    type = "deletion",
    originally = "",
    replacement = ""
)
experimentData

Arguments

x (IRanges) names(x) indicating read_id
ID (string)
score (numeric) scores from the alignments
strand_info (string) Either '+', '-'
type (string)
originally (string) Base pairs on the amplicon.
replacement (string) Base pairs on the read.

Value

(GRanges) Object with meta-data

experimentData

Experiment data.

Description

Get experiment data.frame with information on the experiment level.

Usage

experimentData(x)

Arguments

x (AlignmentsExperimentSet)

Value

(data.frameOrNULL)

Examples

file_path <- system.file("extdata", "results", "alignments",
                         "AlignmentsExperimentSet.rds", package = "amplican")
aln <- readRDS(file_path)
experimentData(aln)
experimentData<- Experiment data.

Description

Set experiment data.frame with information on the experiment level.

Usage

experimentData(x) <- value

Arguments

  x        (AlignmentsExperimentSet)
  value    (data.frame)

Value

  (AlignmentsExperimentSet)

Examples

file_path <- system.file("extdata", "results", "alignments",
"AlignmentsExperimentSet.rds", package = "amplican")
aln <- readRDS(file_path)
experimentData(aln) <- experimentData(aln) # replace with the same values

extractEvents Extract AlignmentsExperimentSet events into data.frame.

Description

Extracts events (insertions, deletions, mismatches) from alignments into data.frame. Can use multiple cores as process is quite slow. All events are relative towards forward strand. "-" in strand column indicates which events were from reverse reads.

Usage

extractEvents(object, use_parallel = FALSE)

Arguments

  object      (AlignmentsExperimentSet)
  use_parallel (boolean) Set to TRUE, if you have registered multicore back-end with register.
**findEOP**

**Value**

(data.frame) Compatible with GRanges style.

**Examples**

```r
file_path <- system.file("extdata", "results", "alignments",
"AlignmentsExperimentSet.rds", package = "amplican")
aln <- readRDS(file_path)
extractEvents(aln)
```

---

**findEOP**

Find Events Overlapping Primers.

**Description**

Very often alignments return deletions that are not real deletions, but rather artifact of incomplete reads eg.:

```
ACTGAAAA------ <- this "deletion" should be filtered
ACTG----ACTGACTG
```

**Usage**

```r
findEOP(aln, cfgT)
```

**Arguments**

- **aln** (data.frame) Should contain events from alignments in GRanges style with columns eg. seqnames, width, start, end.
- **cfgT** (data.frame) Needs columns Forward_Primer, ReversePrimer and Amplicon.

**Value**

(logical vector) where TRUE indicates events that are overlapping primers

**See Also**

findPD findLQR

Other filters: findLQR(), findPD()
findLQR

Find Off-targets and Fragmented alignments from reads.

Description
Will try to detect off-targets and low quality alignments (outliers). It tries k-means clustering on normalized number of events per read and read alignment score. If there are 3 clusters (decided based on silhouette criterion) cluster with high event count and low alignment score will be marked for filtering. When there is less than 1000 scores in \texttt{aln} it will filter nothing.

Usage

\begin{verbatim}
findLQR(aln)
\end{verbatim}

Arguments

- \texttt{aln} (data.frame) Should contain events from alignments in GRanges style with columns eg. seqnames, width, start, end, score.

Value

(logical vector) where TRUE indicates events that are potential off-targets or low quality alignments.

See Also

\texttt{findPD} \texttt{findEOP}

Other filters: \texttt{findEOP()}, \texttt{findPD()}

Examples

\begin{verbatim}
file_path <- system.file("extdata", "results", "alignments",
  "raw_events.csv", package = "amplican")
aln <- data.table::fread(file_path)
cfgT <- data.table::fread(
  system.file("extdata", "results", "config_summary.csv",
              package = "amplican"))
findEOP(aln, cfgT)
\end{verbatim}

\begin{verbatim}
findLQR
\end{verbatim}
findPD

Find PRIMER DIMER reads.

Description

Use to filter reads that are most likely PRIMER DIMERS.

Usage

findPD(aln, cfgT, PRIMER_DIMER = 30)

Arguments

aln (data.frame) Should contain events from alignments in GRanges style with columns eg. seqnames, width, start, end.

cfgT (data.frame) Needs columns Forward_Primer, ReversePrimer and Amplicon.

PRIMER_DIMER (numeric) Value specifying buffer for PRIMER DIMER detection. For a given read it will be recognized as PRIMER DIMER when alignment will introduce gap of size bigger than:

length of amplicon - (lengths of PRIMERS + PRIMER_DIMER value)

Value

(logical) Where TRUE indicates event classified as PRIMER DIMER

See Also

findEOP findLQR

Other filters: findEOP(), findLQR()

Examples

file_path <- system.file("extdata", "results", "alignments",
"raw_events.csv", package = "amplican")
aln <- data.table::fread(file_path)
cfgT <- data.table::fread(
  system.file("extdata", "results", "config_summary.csv",
  package = "amplican"))
findPD(aln, cfgT)
flipRanges

Reverse complement events that have amplicons with direction 1.

Description
Reverse complement events that have amplicons with direction 1.

Usage
flipRanges(idR, cfgT)

Arguments
idR (data.frame) Loaded events.
cfgT (data.frame) Loaded configuration file.

Value
(data.frame) Returns input idR, but events for amplicons with direction 1 reverse complemented, "+" and "-" swapped.

fwdReads

Alignments for forward reads.

Description
Get alignments for forward reads.

Usage
fwdReads(x)

Arguments
x (AlignmentsExperimentSet)

Value
(listOrNULL) list with objects of PairwiseAlignmentsSingleSubject

Examples
file_path <- system.file("extdata", "results", "alignments", "AlignmentsExperimentSet.rds", package = "amplican")
aln <- readRDS(file_path)
fwdReads(aln)
fwdReads<- *Alignments for forward reads.*

**Description**

Set alignments for forward reads.

**Usage**

```r
defwdReads(x) <- value
```

**Arguments**

- `x` (AlignmentsExperimentSet)
- `value` (list) Named (experiment IDs) list with elements of `PairwiseAlignmentsSingleSubject` class.

**Examples**

```r
file_path <- system.file("extdata", "results", "alignments", "AlignmentsExperimentSet.rds", package = "amplican")
aln <- readRDS(file_path)
fwdReads(aln) <- fwdReads(aln)  # replace with the same values
```

---

fwdReadsType *Type of forward reads.*

**Description**

Get type of forward reads.

**Usage**

```r
fwdReadsType(x)
```

**Arguments**

- `x` (AlignmentsExperimentSet)

**Value**

(listOrNULL) list with objects of `PairwiseAlignmentsSingleSubject`
Examples

```r
file_path <- system.file("extdata", "results", "alignments",
  "AlignmentsExperimentSet.rds", package = "amplican")
aln <- readRDS(file_path)
fwdReadsType(aln)
```

```r
fwdReadsType<-
```

**Description**

Set read type for forward reads.

**Usage**

```r
fwdReadsType(x) <- value
```

**Arguments**

- `x` (AlignmentsExperimentSet)
- `value` (list) Named (experiment IDs) list with elements of

**Value**

(AlignmentsExperimentSet) **PairwiseAlignmentsSingleSubject** class.

**Examples**

```r
file_path <- system.file("extdata", "results", "alignments",
  "AlignmentsExperimentSet.rds", package = "amplican")
aln <- readRDS(file_path)
fwdReadsType(aln) <- fwdReadsType(aln) # replace with the same values
```

**geom_bezier**

*Create quadratic or cubic bezier curves [copied from ggforce]*

**Description**

This set of functionality is copied from ggforce package due to dependency issues on Bioconductor and is used internally (not exported) only. This set of geoms makes it possible to connect points creating either quadratic or cubic beziers. bezier works by calculating points along the bezier and connecting these to draw the curve.
Usage

stat_bezier(
  mapping = NULL,
  data = NULL,
  geom = "path",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  n = 100,
  inherit.aes = TRUE,
  ...
)

geom_bezier(
  mapping = NULL,
  data = NULL,
  stat = "bezier",
  position = "identity",
  arrow = NULL,
  lineend = "butt",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  n = 100,
  ...
)

Arguments

mapping     Set of aesthetic mappings created by aes(). If specified and inherit.aes =
             TRUE (the default), it is combined with the default mapping at the top level of
             the plot. You must supply mapping if there is no plot mapping.

data         The data to be displayed in this layer. There are three options:
             If NULL, the default, the data is inherited from the plot data as specified in the
             call to ggplot().
             A data.frame, or other object, will override the plot data. All objects will be
             fortified to produce a data frame. See fortify() for which variables will be
             created.
             A function will be called with a single argument, the plot data. The return
             value must be a data.frame, and will be used as the layer data. A function
             can be created from a formula (e.g. ~ head(.x, 10)).

geom         The geometric object to use to display the data for this layer. When using a
             stat_*() function to construct a layer, the geom argument can be used to over-
             ride the default coupling between stats and geoms. The geom argument accepts
             the following:
             • A Geom ggproto subclass, for example GeomPoint.
geom_bezier

• A string naming the geom. To give the geom as a string, strip the function name of the geom_prefix. For example, to use geom_point(), give the geom as "point".
• For more information and other ways to specify the geom, see the layer geom documentation.

position A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following:
• The result of calling a position function, such as position_jitter(). This method allows for passing extra arguments to the position.
• A string naming the position adjustment. To give the position as a string, strip the function name of the position_prefix. For example, to use position_jitter(), give the position as "jitter".
• For more information and other ways to specify the position, see the layer position documentation.

na.rm If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.

show.legend logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display.

n The number of points to create for each segment

inherit.aes If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn’t inherit behaviour from the default plot specification, e.g. borders().

Other arguments passed on to layer()’s params argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the position argument, or aesthetics that are required can not be passed through .... Unknown arguments that are not part of the 4 categories below are ignored.

• Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, colour = "red" or linewidth = 3. The geom’s documentation has an Aesthetics section that lists the available options. The ‘required’ aesthetics cannot be passed on to the params. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.
• When constructing a layer using a stat_*() function, the ... argument can be used to pass on parameters to the geom part of the layer. An example of this is stat_density(geom = "area", outline.type = "both"). The geom’s documentation lists which parameters it can accept.
• Inversely, when constructing a layer using a geom_*() function, the ... argument can be used to pass on parameters to the stat part of the layer. An example of this is geom_area(stat = "density", adjust = 0.5). The stat’s documentation lists which parameters it can accept.
• The key_glyph argument of layer() may also be passed on through .... This can be one of the functions described as key glyphs, to change the display of the layer in the legend.
stat

The statistical transformation to use on the data for this layer. When using a geom_*() function to construct a layer, the stat argument can be used to override the default coupling between geoms and stats. The stat argument accepts the following:

- A Stat ggproto subclass, for example StatCount.
- A string naming the stat. To give the stat as a string, strip the function name of the stat_ prefix. For example, to use `stat_count()`, give the stat as "count".
- For more information and other ways to specify the stat, see the layer stat documentation.

arrow

Arrow specification, as created by `grid::arrow()`.

lineend

Line end style (round, butt, square).

Details

Input data is understood as a sequence of data points the first being the start point, then followed by one or two control points and then the end point. More than 4 and less than 3 points per group will throw an error.

Aesthetics

`geom_link`, `geom_link2` and `geom_lin0` understand the following aesthetics (required aesthetics are in bold):

- **x** - **y** - color - linwidth - linetype - alpha - lineend

Computed variables

- x, y The interpolated point coordinates
- index The progression along the interpolation mapped between 0 and 1

Author(s)

Thomas Lin Pedersen

Examples

```r
beziers <- data.frame(
  x = c(1, 2, 3, 4, 4, 6, 6),
  y = c(0, 2, 0, 0, 2, 2, 0),
  type = rep(c('cubic', 'quadratic'), c(3, 4)),
  point = c('end', 'control', 'end', 'end', 'control', 'control', 'end')
)
help_lines <- data.frame(
  x = c(1, 3, 4, 6),
  xend = c(2, 2, 4, 6),
  y = 0,
  yend = 2
)
```
getEvents

Transform aligned strings into GRanges representation of events.

Description

Transforms aligned strings into GRanges representation with events of deletions, insertions and mismatches. Subject should come from one amplicon sequence, after alignment to many sequences (patterns).

getEventInfo

This function takes alignments and gives back the events coordinates.

Description

This function takes alignments and gives back the events coordinates.

Usage

getEventInfo(align, ID, ampl_shift, strand_info = "+")

Arguments

align  (PairwiseAlignmentsSingleSubject)
ID  (string)
ampl_shift  (numeric vector) Shift events additionally by this value. PairwiseAlignmentsSingleSubject returns truncated alignments.
strand_info  (string) Either '+' or '-' or default '*'
ampl_len  (numeric) Length of the amplicon (subject)

Value

(GRanges) Object with meta-data for insertion, deletion, mismatch
**get_left_primer**

**Usage**

`getEvents(
  pattern,
  subject,
  scores,
  ID = "NA",
  ampl_shift = 1L,
  ampl_start = 1L,
  strand_info = "+"
)`

**Arguments**

- `pattern` (character) Aligned pattern.
- `subject` (character) Aligned subject.
- `scores` (integer) Alignment scores of the pattern and subject.
- `ID` (character) Will be used as seqnames of output GRanges.
- `ampl_shift` (numeric) Possible shift of the amplicons.
- `ampl_start` (numeric) Real amplicon starts. *pairwiseAlignment* clips alignments, therefore to output GRanges relative to the amplicon sequence (subject) ranges have to be shifted.
- `strand_info` (character) Strands to assign.

**Value**

- `GRanges` (GRanges) Same as events.

---

<table>
<thead>
<tr>
<th><code>get_left_primer</code></th>
<th><code>left primer sequence</code></th>
</tr>
</thead>
</table>

**Description**

left primer sequence

**Usage**

`get_left_primer(config, id)`

**Arguments**

- `config` (data.frame) config table
- `id` (vector) a vector of id's

**Value**

- (character) left primer sequence
get_right_primer    right primer sequence

Description
right primer sequence

Usage
get_right_primer(config, id)

Arguments
config      (data.frame) config table
id         (vector) a vector of id’s

Value
(character) right primer sequence

get_seq          amplicon sequence, reverse complemented when needed

Description
amplicon sequence, reverse complemented when needed

Usage
get_seq(config, id, column = "Amplicon")

Arguments
config      (data.frame) config table
id         (vector) a vector of id’s

Value
(character) amplicon sequence, reverse complemented if Direction 1
**goodAvgQuality**  
*This filters out sequences which have bad average quality readings.*

**Description**

This filters out sequences which have bad average quality readings.

**Usage**

```r
goodAvgQuality(reads, avg = 30, batch_size = 1e+07)
```

**Arguments**

- `reads` (ShortRead object) Loaded reads from fastq.
- `avg` (numeric) This is what the average score of the quality of sequence should be. For example, if we have a sequence with nucleotides which have quality 70-70-70, the average would be 70. If set the average to 70 or less the sequence will pass. If we set the average to 71 the sequence will not pass.
- `batch_size` (numeric) How many reads to process at a time.

**Value**

(boolean) Logical vector with the valid rows as TRUE.

---

**goodBaseQuality**  
*Filters out sequences which have bad base quality readings.*

**Description**

Filters out sequences which have bad base quality readings.

**Usage**

```r
goodBaseQuality(reads, min = 20, batch_size = 1e+07)
```

**Arguments**

- `reads` (ShortRead object) Loaded reads from fastq.
- `min` (numeric) This is the minimum quality that we accept for every nucleotide. For example, if we have a sequence with nucleotides which have quality 50-50-50-50-10, and we set the minimum to 30, the whole sequence will be a bad sequence.
- `batch_size` (numeric) How many reads to process at a time.

**Value**

(boolean) Logical vector with the valid rows as TRUE.
### is hdr strict

Figure out which reads conform to the HDR using the donor.

**Description**

This is strict detection as compared to `is_hdr` which was designed to be less specific and allow for all kinds of donors. This method requires that you have exactly the same events (mismatches, insertions, deletions) as the difference between amplicon and donor sequences. It ignores everything else, so other mismatches and small indels etc. as noise are allowed here for valid HDR.

**Usage**

```r
is_hdr_strict(aln, cfgT, scoring_matrix, gap_opening = 25, gap_extension = 0)
```

**Arguments**

- `aln` (data.table) This are events that contain already consensus column, they are also shifted and normalized.
- `cfgT` (data.table) Config data.table with columns for amplicon and donor.
- `scoring_matrix` (scoring matrix)
- `gap_opening` (integer)
- `gap_extension` (integer)

**Value**

(aln) same as aln on entry, but readType is updated to TRUE when read is recognized as HDR

### lookupAlignment

Show alignment in human readable format.

**Description**

Prints alignments in blast-like style for human examination.

**Usage**

```r
lookupAlignment(x, ID, read_id = 1)
```

**Arguments**

- `x` (AlignmentsExperimentSet)
- `ID` (string) Experiment Identifier
- `read_id` (numeric) Read Identifier. Reads are sorted by frequency. Defaults to 1, most abundant read.
Value

(print to view)

Examples

# load example object
file_path <- system.file("extdata", "results", "alignments", "AlignmentsExperimentSet.rds", package = "amplican")
aln <- readRDS(file_path)
# look at most frequent reads aligned from experiment ID_1
lookupAlignment(aln, "ID_1")

makeAlignment  Make alignments helper.

Description

Aligning reads to the amplicons for each ID in this barcode, constructing amplicanAlignment. Assume that all IDs here belong to the same barcode.

Usage

makeAlignment(
  cfgT,
  average_quality,
  min_quality,
  filter_n,
  batch_size,
  scoring_matrix,
  gap_opening,
  gap_extension,
  fastqfiles,
  primer_mismatch,
  donor_mismatch,
  donor_strict
)

Arguments

cfgT config file as data table
average_quality (numeric) The FASTQ file have a quality for each nucleotide, depending on sequencing technology there exist many formats. This package uses readFastq to parse the reads. If the average quality of the reads fall below value of average_quality then sequence is filtered. Default is 0.
min_quality (numeric) Similar as in average_quality, but depicts the minimum quality for ALL nucleotides in given read. If one of nucleotides has quality BELOW min_quality, then the sequence is filtered. Default is 20.

filter_n (boolean) Whether to filter out reads containing N base.

batch_size (numeric) How many reads to analyze at a time? Needed for filtering of large fastq files.

scoring_matrix (matrix) Default is 'NUC44'. Pass desired matrix using nucleotideSubstitutionMatrix.

gap_opening (numeric) The opening gap score.

gap_extension (numeric) The gap extension score.

fastqfiles (numeric) Normally you want to use both FASTQ files. But in some special cases, you may want to use only the forward file, or only the reverse file. Possible options:

- 0 Use both FASTQ files.
- 0.5 Use both FASTQ files, but only for one of the reads (forward or reverse) is required to have primer perfectly matched to sequence - eg. use when reverse reads are trimmed of primers, but forward reads have forward primer in the sequence.
- 1 Use only the forward FASTQ file.
- 2 Use only the reverse FASTQ file.

primer_mismatch (numeric) Decide how many mismatches are allowed during primer matching of the reads, that groups reads by experiments. When primer_mismatch = 0 no mismatches are allowed, which can increase number of unassigned read.

donor_mismatch (numeric) How many events of length 1 (mismatches, deletions and insertions of length 1) are allowed when aligning toward the donor template. This parameter is only used when donor template is specified. The higher the parameter the less strict will be algorithm accepting read as HDR. Set to 0 if only perfect alignments to the donor template marked as HDR, unadvised due to error rate of the sequencers.

donor.strict (logical) Applies more strict algorithm for HDR detection. Only these reads that have all of the donor events will count as HDR. Tolerates ‘donor_mismatch’ level of noise, but no indels are allowed. Use this when your reads should span over the whole window of the donor events. Might be more time consuming.

Value

amplicanAlignment object for this barcode experiments
**metaplot_deletions**

*MetaPlots deletions using ggplot2.*

**Description**

This function plots deletions in relation to the amplicons for given selection vector that groups values by given config group. All reads should already be converted to their relative position to their respective amplicon using `amplicanMap`. Top plot is for the forward reads and bottom plot is for reverse reads.

**Usage**

```r
metaplot_deletions(alnmt, config, group, selection, over = "overlaps")
```

**Arguments**

- **alnmt** (data.frame): Loaded alignment information from `events_filtered_shifted_normalized.csv` file.
- **config** (data.frame): Loaded table from `config_summary.csv` file.
- **group** (string): Name of the column from the config file to use for grouping. Events are subselected based on this column and values from selection.
- **selection** (string or vector of strings): Values from config column specified in group argument.
- **over** (string): Specify which column contains overlaps with expected cut sites generated by `amplicanOverlap`

**Value**

(deletions metaplot) ggplot2 object of deletions metaplot

**See Also**

Other specialized plots: `metaplot_insertions()`, `metaplot_mismatches()`, `plot_cuts()`, `plot_deletions()`, `plot_heterogeneity()`, `plot_insertions()`, `plot_mismatches()`, `plot_variants()`

**Examples**

```r
# example config
config <- read.csv(system.file("extdata", "results", "config_summary.csv", package = "amplican"))

# example alignments results
alignments_file <- system.file("extdata", "results", "alignments", "events_filtered_shifted_normalized.csv", package = "amplican")
alignments <- read.csv(alignments_file)
metaplot_deletions(alignments[alignments$consensus, ], config, "Group", "Betty")
```
metaplot_insertions  MetaPlots insertions using ggplot2.

Description

This function plots insertions in relation to the amplicons for given selection vector that groups values by given config group. All reads should already be converted to their relative position to their respective amplicon using `amplicanMap`. Top plot is for the forward reads and bottom plot is for reverse reads.

Usage

```
metaplot_insertions(alnmt, config, group, selection)
```

Arguments

- `alnmt` (data.frame): Loaded alignment information from alignments_events.csv file.
- `config` (data.frame): Loaded table from config_summary.csv file.
- `group` (string): Name of the column from the config file to use for grouping. Events are subselected based on this column and values from selection.
- `selection` (string or vector of strings): Values from config column specified in group argument.

Value

(insertions metaplot) ggplot2 object of insertions metaplot

See Also

Other specialized plots: `metaplot_deletions()`, `metaplot_mismatches()`, `plot_cuts()`, `plot_deletions()`, `plot_heterogeneity()`, `plot_insertions()`, `plot_mismatches()`, `plot_variants()`

Examples

```
#example config
cfg <- read.csv(system.file("extdata", "results", "config_summary.csv",
    package = "amplican")

#example alignments results
alignments_file <- system.file("extdata", "results", "alignments",
    "events_filtered_shifted_normalized.csv",
    package = "amplican")
alignments <- read.csv(alignments_file)
metaplot_insertions(alignments[alignments$consensus, ], config,
    "Group", "Betty")
```
**metaplot_mismatches**

MetaPlots mismatches using ggplot2.

**Description**

Plots mismatches in relation to the amplicons for given selection vector that groups values by given config group. All reads should already be converted to their relative position to their respective amplicon using `amplicanMap`. Zero position on new coordinates is the most left UPPER case letter of the respective amplicon. This function filters out all alignment events that have amplicons without UPPER case defined. Top plot is for the forward reads and bottom plot is for reverse reads.

**Usage**

```r
metaplot_mismatches(alnmt, config, group, selection)
```

**Arguments**

- `alnmt` (data.frame): Loaded alignment information from alignments_events.csv file.
- `config` (data.frame): Loaded table from config_summary.csv file.
- `group` (string): Name of the column from the config file to use for grouping. Events are subselected based on this column and values from selection.
- `selection` (string or vector of strings): Values from config column specified in group argument.

**Value**

(mismatches metaplot) ggplot2 object of mismatches metaplot

**See Also**

Other specialized plots: `metaplot_deletions()`, `metaplot_insertions()`, `plot_cuts()`, `plot_deletions()`, `plot_heterogeneity()`, `plot_insertions()`, `plot_mismatches()`, `plot_variants()`

**Examples**

```r
#example config
cfg <- read.csv(system.file("extdata", "results", "config_summary.csv", package = "amplican"))

#example alignments results
alignments_file <- system.file("extdata", "results", "alignments", "events_filtered_shifted_normalized.csv", package = "amplican")
alignments <- read.csv(alignments_file)
metaplot_mismatches(alignments, config, "Group", "Betty")
```
pairToEvents

Read "pair" format of EMBOSS needle into GRanges as events.

Description

Parse EMBOSS needle (or needleall) "pair" format into GRanges representation with events of deletions, insertions and mismatches. Make sure that each file corresponds to single subject (single amplicon). Assumes that bottom sequence "-bsequence" corresponds to the "subject" and full sequence alignment is returned.

Usage

```
pairToEvents(file, ID = "NA", strand_info = "+")
```

Arguments

- **file** (character) File path.
- **ID** (character) ID of the experiment, will be used as seqnames of the returned ranges.
- **strand_info** (character) Strand to assign.

Value

(GRanges) Same as events.

plot_amplicon

Plots amplicon sequence using ggplot2.

Description

Plots amplicon sequence using ggplot2.

Usage

```
plot_amplicon(amplicon, from, to)
```

Arguments

- **amplicon** (character) Sequence of the amplicon to plot.
- **from** (number) Minimum on x axis - start of the amplicon
- **to** (number) Maximum on x axis - not necessarily end of the amplicon

Value

(amplicon plot) ggplot2 object of amplicon plot
**plot_cuts**

Plots cuts using ggplot2.

**Description**

This function plots cuts in relation to the amplicon with distinction for each ID.

**Usage**

```r
plot_cuts(alignments, config, id, cut_buffer = 5, xlab_spacing = 4)
```

**Arguments**

- `alignments` (data.frame) Loaded alignment information from alignments_events.csv file.
- `config` (data.frame) Loaded table from config_summary.csv file.
- `id` (string or vector of strings) Name of the ID column from config file or name of multiple IDs if it is possible to group them. First amplicon will be used as the basis for plot.
- `cut_buffer` (numeric) Default is 5, you should specify the same as used in the analysis.
- `xlab_spacing` (numeric) Spacing of the x axis labels. Default is 4.

**Value**

(cuts plot) gtable object of cuts plot

**See Also**

Other specialized plots: `metaplot_deletions()`, `metaplot_insertions()`, `metaplot_mismatches()`, `plot_deletions()`, `plot_heterogeneity()`, `plot_insertions()`, `plot_mismatches()`, `plot_variants()`

**Examples**

```r
# example config
config <- read.csv(system.file("extdata", "results", "config_summary.csv", package = "amplican"))

# example alignments results
alignments_file <- system.file("extdata", "results", "alignments", "events_filtered_shifted_normalized.csv", package = "amplican")
alignments <- read.csv(alignments_file)
plot_cuts(alignments[alignments$consensus & alignments$overlaps, ], config, c("ID_1", "ID_3"))
```
plot_deletions

Plots deletions using ggplot2.

Description

This function plots deletions in relation to the amplicon, assumes events are relative to the expected cut site. Top plot is for the forward reads, middle one shows amplicon sequence, and bottom plot is for reverse reads.

Usage

plot_deletions(
  alignments,
  config,
  id,
  cut_buffer = 5,
  xlab_spacing = 4,
  over = "overlaps"
)

Arguments

alignments (data.frame) Loaded alignment information from alignments.csv file.
config (data.frame) Loaded table from config_summary.csv file.
id (string or vector of strings) Name of the ID column from config file or name of multiple IDs if it is possible to group them. First amplicon will be used as the basis for plot.
cut_buffer (numeric) Default is 5, you should specify the same as used in the analysis.
xlab_spacing (numeric) Spacing of the x axis labels. Default is 4.
over (string) Specify which columns contains overlaps with expected cut sites generated by amplicanOverlap

Value

(deletions plot) gtable object of deletions plot

See Also

Other specialized plots: metaplot_deletions(), metaplot_insertions(), metaplot_mismatches(), plot_cuts(), plot_heterogeneity(), plot_insertions(), plot_mismatches(), plot_variants()
Examples

# example config
config <- read.csv(system.file("extdata", "results", "config_summary.csv", package = "amplican"))

# example alignments results
alignments_file <- system.file("extdata", "results", "alignments", "events_filtered_shifted_normalized.csv", package = "amplican")
alignments <- read.csv(alignments_file)
p <- plot_deletions(alignments[alignments$consensus, ],
                    config, c('ID_1', 'ID_3'))

plot_height(x)

Description

Helper function to calculate figure height based on number of elements to plot for automating sizes of figures in knitted reports.

Usage

plot_height(x)

Arguments

x (numeric) number of elements to fit onto height axis

Value

(numeric) In inches

Examples

plot_height(20)
plot_heterogeneity

Plots heterogeneity of the reads using ggplot2.

Description

This function creates stacked barplot explaining reads heterogeneity. It groups reads by user defined levels and measures how unique are reads in this level. Uniqueness of reads is simplified to the bins and colored according to the color gradient. Default color black indicates very high heterogeneity of the reads. The more yellow (default) the more similar are reads and less heterogeneous.

Usage

plot_heterogeneity(
  alignments,        # (data.frame) Loaded alignment information from alignments_events.csv file.
  config,            # (data.frame) Loaded table from config_summary.csv file.
  level = "ID",      # (string) Name of the column from config file specifying levels to group by.
  colors = c("#000000", "#F0E442"), # (html colors vector) Two colours for gradient, eg. c('#000000', '#F0E442').
  bins = c(0, 5, seq(10, 100, 10))  # (numeric vector) Numeric vector from 0 to 100 specifying bins eg. c(0, 5, seq(10, 100, 10)).
)

Arguments

alignments (data.frame) Loaded alignment information from alignments_events.csv file.
config (data.frame) Loaded table from config_summary.csv file.
level (string) Name of the column from config file specifying levels to group by.
colors (html colors vector) Two colours for gradient, eg. c('#000000', '#F0E442').
bins (numeric vector) Numeric vector from 0 to 100 specifying bins eg. c(0, 5, seq(10, 100, 10)).

Value

(heterogeneity plot) ggplot2 object of heterogeneity plot

See Also

Other specialized plots: metaplot_deletions(), metaplot_insertions(), metaplot_mismatches(), plot_cuts(), plot_deletions(), plot_insertions(), plot_mismatches(), plot_variants()

Examples

#example config
cfg <- read.csv(system.file("extdata", "results", "config_summary.csv", 
  package = "amplican"))

#example alignments results
alignments_file <- system.file("extdata", "results", "alignments", 
  "events_filtered_shifted_normalized.csv", 
  package = "amplican")
**plot_insertions**  

Plots insertions using ggplot2.

**Description**

This function plots insertions in relation to the amplicon. Top plot is for the forward reads, middle one shows amplicon sequence, and bottom plot is for reverse reads.

**Usage**

`plot_insertions(alignments, config, id, cut_buffer = 5, xlab_spacing = 4)`

**Arguments**

- `alignments` (data.frame): Loaded alignment information from alignments_events.csv file.
- `config` (data.frame): Loaded table from config_summary.csv file.
- `id` (string or vector of strings): Name of the ID column from config file or name of multiple IDs if it is possible to group them. First amplicon will be used as the basis for plot.
- `cut_buffer` (numeric): Default is 5, you should specify the same as used in the analysis.
- `xlab_spacing` (numeric): Spacing of the x axis labels. Default is 4.

**Value**

(insertions plot) gtable object of insertions plot

**See Also**

Other specialized plots: `metaplot_deletions()`, `metaplot_insertions()`, `metaplot_mismatches()`, `plot_cuts()`, `plot_deletions()`, `plot_heterogeneity()`, `plot_mismatches()`, `plot_variants()`

**Examples**

```r
# example config
config <- read.csv(system.file("extdata", "results", "config_summary.csv", package = "amplican"))

# example alignments results
alignments_file <- system.file("extdata", "results", "alignments", "events_filtered_shifted_normalized.csv", package = "amplican")
alignments <- read.csv(alignments_file)
p <- plot_insertions(alignments, config, c('ID_1', 'ID_3'))
```
**plot_mismatches**  
Plots mismatches using ggplot2.

**Description**
Plots mismatches in relation to the amplicon, assumes your reads are relative to the respective amplicon sequences predicted cut sites. Top plot is for the forward reads, middle one shows amplicon sequence, and bottom plot is for reverse reads.

**Usage**

```r
plot_mismatches(alignments, config, id, cut_buffer = 5, xlab_spacing = 4)
```

**Arguments**

- `alignments` (data.frame): Loaded alignment information from alignments_events.csv file.
- `config` (data.frame): Loaded table from config_summary.csv file.
- `id` (string or vector of strings): Name of the ID column from config file or name of multiple IDs, if it is possible to group them. They have to have the same amplicon, amplicons on the reverse strand will be reverse complemented to match forward strand amplicons.
- `cut_buffer` (numeric): Default is 5, you should specify the same as used in the analysis.
- `xlab_spacing` (numeric): Spacing of the x axis labels. Default is 4.

**Value**
(mismatches plot) gtable object of mismatches plot

**See Also**
Other specialized plots: `metaplot_deletions()`, `metaplot_insertions()`, `metaplot_mismatches()`, `plot_cuts()`, `plot_deletions()`, `plot_heterogeneity()`, `plot_insertions()`, `plot_variants()`

**Examples**

```r
# example config
config <- read.csv(system.file("extdata", "results", "config_summary.csv",
package = "amplican"))
# example alignments results
alignments_file <- system.file("extdata", "results", "alignments",
"events_filtered_shifted_normalized.csv",
package = "amplican")
alignments <- read.csv(alignments_file)
id <- c('ID_1', 'ID_3'); cut_buffer = 5; xlab_spacing = 4;
p <- plot_mismatches(alignments, config, c('ID_1', 'ID_3'))
ggplot2::ggsave("~/test.png", p, width = 25, units = "in")
```
**plot_variants**

Plots most frequent variants using ggplot2.

**Description**

This function plots variants in relation to the amplicon. Shows sequences of top mutants without aggregating on deletions, insertions and mismatches.

**Usage**

```r
plot_variants(
    alignments,  # (data.frame) Loaded alignment information from alignments_events.csv file.
    config,      # (data.frame) Loaded table from config_summary.csv file.
    id,          # (string or vector of strings) Name of the ID column from config file or name of multiple IDs if it is possible to group them. First amplicon will be used as the basis for plot. If Donor is available we will try to add the first donor and mark it on the plot.
    cut_buffer = 5,  # (numeric) Default is 5, you should specify the same as used in the analysis.
    top = 10,       # (numeric) Specify number of most frequent reads to plot. By default it is 10. Check `plot_heterogeneity` to see how many reads will be enough to give good overview of your variants.
    annot = if (amplican:::get_seq(config, id, "Donor") == ") "cov" else NA,  # ("codon" or "cov" or NA) What to display for annotation top plot. When NA will not display anything, also not display total summary. Codon plot is all reading frames for a given window, and the default "cov" is coverage of all indels and mismatches over a given window.
    summary_plot = amplican:::get_seq(config, id, "Donor") == "",  # (boolean) Whether small summary plot in the upper right corner should be displayed. Top bar summarizes total reads with frameshift (F), reads with Edits without Frameshift (Edits) and reads without Edits (Match).
    frameshift = amplican:::get_seq(config, id, "Donor") == ""  # (boolean) Whether to include Frameshift column in the table.
)
```

**Arguments**

- **alignments** (data.frame): Loaded alignment information from alignments_events.csv file.
- **config** (data.frame): Loaded table from config_summary.csv file.
- **id** (string or vector of strings): Name of the ID column from config file or name of multiple IDs if it is possible to group them. First amplicon will be used as the basis for plot. If Donor is available we will try to add the first donor and mark it on the plot.
- **cut_buffer** (numeric): Default is 5, you should specify the same as used in the analysis.
- **top** (numeric): Specify number of most frequent reads to plot. By default it is 10. Check `plot_heterogeneity` to see how many reads will be enough to give good overview of your variants.
- **annot** ("codon" or "cov" or NA): What to display for annotation top plot. When NA will not display anything, also not display total summary. Codon plot is all reading frames for a given window, and the default "cov" is coverage of all indels and mismatches over a given window.
- **summary_plot** (boolean): Whether small summary plot in the upper right corner should be displayed. Top bar summarizes total reads with frameshift (F), reads with Edits without Frameshift (Edits) and reads without Edits (Match).
- **frameshift** (boolean): Whether to include Frameshift column in the table.
Details

Top plot shows all six possible frames for given amplicon. Amino acids are colored as follows:

- Small nonpolar: G, A, S, T (Orange)
- Hydrophobic: C, V, I, L, P, F, Y, M, W (Green)
- Polar: N, Q, H (Magenta)
- Negatively charged: D, E (Red)
- Positively charged: K, R (Blue)
- Other: eg. *, U, + (Grey)

Variant plot shows amplicon reference. UPPER letters which were the basis for window selection are highlighted with dashed white box (guideRNA). Black triangles are reflecting insertion points. Dashed letters indicate deletions. Table associated with variant plot represents:

- Freq: Frequency of given read in experiment. Variants are ordered by frequency value.
- Count: Represents raw count of this variant reads in experiment.
- F: Sum of deletion and insertion widths of events overlapping presented window. Green background indicates frameshift.

Value

(variant plot) gtable object of variants plot

Note

This function is inspired by `plotAlignments`.

See Also

Other specialized plots: `metaplot_deletions()`, `metaplot_insertions()`, `metaplot_mismatches()`, `plot_cuts()`, `plot_deletions()`, `plot_heterogeneity()`, `plot_insertions()`, `plot_mismatches()`

Examples

```r
#example config
cfg <- read.csv(system.file("extdata", "results", "config_summary.csv", package = "amplican"))
#example alignments results
alignments_file <- system.file("extdata", "results", "alignments", "events_filtered_shifted_normalized.csv", package = "amplican")
alignments <- read.csv(alignments_file)

alignments <- alignments[alignments$consensus & alignments$overlaps,]
p <- plot_variants(alignments[alignments$consensus & alignments$overlaps,],
  config, c("ID_1","ID_3"))
# with Donor we dont plot summary and the annot, summary plot and frameshift
p <- plot_variants(alignments[alignments$consensus & alignments$overlaps,],
  config, c("ID_5"))
```
**readCounts**

Alignments for forward reads.

**Description**
Set alignments for forward reads.

**Usage**

```r
readCounts(x)
```

**Arguments**

- `x` (AlignmentsExperimentSet)

**Value**

(listOrNULL)

**Examples**

```r
define file_path: system.file("extdata", "results", "alignments", "AlignmentsExperimentSet.rds", package = "amplican")
define aln: readRDS(file_path)
define readCounts(aln)
```
Examples

```r
file_path <- system.file("extdata", "results", "alignments", "AlignmentsExperimentSet.rds", package = "amplican")
aln <- readRDS(file_path)
readCounts(aln) <- readCounts(aln) # replace with the same values
```

---

### revComp

<table>
<thead>
<tr>
<th>revComp</th>
<th>Reverse and complement given string or list of strings</th>
</tr>
</thead>
</table>

**Description**
Reverse and complement given string or list of strings

**Usage**

```r
revComp(x)
```

**Arguments**

- **x** (string or vector of strings)

**Value**
(string or vector of strings) reverse complemented input

---

### rveReads

<table>
<thead>
<tr>
<th>rveReads</th>
<th>Alignments for reverse reads.</th>
</tr>
</thead>
</table>

**Description**
Get alignments for reverse reads.

**Usage**

```r
rveReads(x)
```

**Arguments**

- **x** (AlignmentsExperimentSet)

**Value**
(listOrNULL) list with objects of PairwiseAlignmentsSingleSubject
Examples

```r
file_path <- system.file("extdata", "results", "alignments", "AlignmentsExperimentSet.rds", package = "amplican")
aln <- readRDS(file_path)
rveReads(aln)
```

Description

Alignments for forward reads.

Usage

```
rveReads(x) <- value
```

Arguments

- `x` (AlignmentsExperimentSet)
- `value` (list) Named (experiment IDs) list with elements of Value (AlignmentsExperimentSet) PairwiseAlignmentSSingleSubject class.

Examples

```r
file_path <- system.file("extdata", "results", "alignments", "AlignmentsExperimentSet.rds", package = "amplican")
aln <- readRDS(file_path)
rveReads(aln) <- rveReads(aln) # replace with the same values
```

rveReadsType Type of reverse reads.

Description

Get type of reverse reads.

Usage

```
rveReadsType(x)
```

Arguments

- `x` (AlignmentsExperimentSet)
Value

(listOrNULL) list with objects of PairwiseAlignmentsSingleSubject

Examples

file_path <- system.file("extdata", "results", "alignments", "AlignmentsExperimentSet.rds", package = "amplican")
aln <- readRDS(file_path)
rveReadsType(aln)

---

rveReadsType<- Read type for reverse reads.

Description

Set read type for reverse reads.

Usage

rveReadsType(x) <- value

Arguments

x (AlignmentsExperimentSet)
value (list) Named (experiment IDs) list with elements of

Value

(AlignmentsExperimentSet) PairwiseAlignmentsSingleSubject class.

Examples

file_path <- system.file("extdata", "results", "alignments", "AlignmentsExperimentSet.rds", package = "amplican")
aln <- readRDS(file_path)
rveReadsType(aln) <- rveReadsType(aln) # replace with the same values
unassignedCount

Get count of unassigned reads.

Description
Get count of unassigned reads.

Usage
unassignedCount(x)

Arguments
x (AlignmentsExperimentSet)

Value
(numeric)

Examples
file_path <- system.file("extdata", "results", "alignments", "AlignmentsExperimentSet.rds", package = "amplican")
aln <- readRDS(file_path)
unassignedCount(aln)

unassignedData
Unassigned read information.

Description
Get unassigned reads and their characteristics.

Usage
unassignedData(x)

Arguments
x (AlignmentsExperimentSet)

Value
(data.frameOrNULL)
Examples

```r
file_path <- system.file("extdata", "results", "alignments",
"AlignmentsExperimentSet.rds", package = "amplican")
aln <- readRDS(file_path)
unassignedData(aln)
```

```
unassignedData<- Alignments for forward reads.
```

Description

Set alignments for forward reads.

Usage

```
unassignedData(x) <- value
```

Arguments

- `x`  
  (AlignmentsExperimentSet)
- `value`  
  (list) Named (experiment IDs) list with elements of

Value

(AlignmentsExperimentSet) `PairwiseAlignmentsSingleSubject` class.

Examples

```r
file_path <- system.file("extdata", "results", "alignments",
"AlignmentsExperimentSet.rds", package = "amplican")
aln <- readRDS(file_path)
unassignedData(aln) <- unassignedData(aln) # replace with the same values
```

```
upperGroups                Detect uppercase as ranges object.
```

Description

For a given string, detect how many groups of uppercases is inside, where are they, and how long they are.

Usage

```
upperGroups(candidate)
```
writeAlignments

Arguments

candidate (string) A string with the nucleotide sequence.

Details

For example: asdkfaAGASDGAsjaeureAFDSfasfjaeiorAuaoeursfasdhsfTTSfajeiasjfa
Has 4 groups of uppercases of length 7, 4, 1 and 3.

Value

(IRanges) A IRanges object with uppercases groups for given candidate string

writeAlignments Write alignments to file.

Description

Saves alignments into txt or fasta file.

Usage

writeAlignments(x, file = '', aln_format = 'txt')

Arguments

x (AlignmentsExperimentSet)
file (connection or string) Destination file. When empty, defaults to standard output.
aln_format ("txt" or "fasta") Specifies format of the file.

Value

(invisible)

Examples

file_path <- system.file("extdata", "results", "alignments",
"AlignmentsExperimentSet.rds", package = "amplican")
aln <- readRDS(file_path)
writeAlignments(aln, file.path(tempdir(), "aln.txt"))
Index

* analysis steps
  - amplicanAlign, 8
  - amplicanConsensus, 11
  - amplicanFilter, 12
  - amplicanMap, 13
  - amplicanNormalize, 14
  - amplicanOverlap, 15
  - amplicanPipeline, 16
  - amplicanPipelineConservative, 19
  - amplicanReport, 23
  - amplicanSummarize, 24

* filters
  - findEOP, 35
  - findLQR, 36
  - findPD, 37

* internal
  - AlignmentsExperimentSet-class, 4
  - alphabetQuality, 7
  - assignedCount, 26
  - barcodeData, 26
  - barcodeData<-, 27
  - checkConfigFile, 28
  - checkFileWriteAccess, 28
  - checkPrimers, 29
  - checkTarget, 29
  - cumsumw, 31
  - decode, 32
  - defGR, 32
  - experimentData, 33
  - experimentData<-, 34
  - extractEvents, 34
  - flipRanges, 38
  - fwdReads, 38
  - fwdReads<-, 39
  - fwdReadsType, 39
  - fwdReadsType<-, 40
  - get_left_primer, 45
  - get_right_primer, 46
  - get_seq, 46
  - getEventInfo, 44
  - goodAvgQuality, 47
  - goodBaseQuality, 47
  - lookupAlignment, 48
  - makeAlignment, 49
  - plot_amplicon, 54
  - readCounts, 63
  - readCounts<-, 63
  - revComp, 64
  - rveReads, 64
  - rveReads<-, 65
  - rveReadsType, 65
  - rveReadsType<-, 66
  - unassignedCount, 67
  - unassignedData, 67
  - unassignedData<-, 68
  - upperGroups, 68
  - writeAlignments, 69

* specialized plots
  - metaplot_deletions, 51
  - metaplot_insertions, 52
  - metaplot_mismatches, 53
  - plot_cuts, 55
  - plot_deletions, 56
  - plot_heterogeneity, 58
  - plot_insertions, 59
  - plot_mismatches, 60
  - plot_variants, 61

[,AlignmentsExperimentSet,numeric,missing,missing-method
  (AlignmentsExperimentSet-class), 4

$,AlignmentsExperimentSet-method
  (AlignmentsExperimentSet-class), 4

aes(), 41
AlignmentsExperimentSet, 8, 10, 17, 20
AlignmentsExperimentSet
  (AlignmentsExperimentSet-class), 4
goodBaseQuality, 47
GRanges, 7, 13, 30, 33, 35, 37, 44, 45, 54
grid::arrow(), 43
IRanges, 31, 33, 69
is_hdr_strict, 48
key glyphs, 42
layer geom, 42
layer position, 42
layer stat, 43
layer(), 42
length, AlignmentsExperimentSet-method
  (AlignmentsExperimentSet-class), 4
lookupAlignment, 10, 48
lookupAlignment, AlignmentsExperimentSet-method
  (AlignmentsExperimentSet-class), 4
makeAlignment, 49
metaplot_deletions, 51, 52, 53, 55, 56, 58–60, 62
metaplot_insertions, 51, 52, 53, 55, 56, 58–60, 62
metaplot_mismatches, 51, 52, 53, 55, 56, 58–60, 62
names, AlignmentsExperimentSet-method
  (AlignmentsExperimentSet-class), 4
nucleotideSubstitutionMatrix, 9, 18, 21, 50
pairToEvents, 54
pairwiseAlignment, 45
PairwiseAlignmentsSingleSubject, 6, 39, 40, 44, 63, 65, 66, 68
plot_amplicon, 54
plot_cuts, 51–53, 55, 56, 58–60, 62
plot_deletions, 51–53, 55, 56, 58–60, 62
plot_height, 57
plot_heterogeneity, 51–53, 55, 56, 58, 59–62
plot_insertions, 51–53, 55, 56, 58, 59, 60, 62
plot_mismatches, 51–53, 55, 56, 58, 59, 60, 62
plot_variants, 51–53, 55, 56, 58–60, 61
plotAlignments, 62
readCounts, 63
readCounts, AlignmentsExperimentSet-method
  (AlignmentsExperimentSet-class), 4
readCounts<-, 63
readCounts<-, AlignmentsExperimentSet-method
  (AlignmentsExperimentSet-class), 4
readFastq, 9, 17, 21, 49
register, 6, 34
revComp, 64
rveReads, 64
rveReads, AlignmentsExperimentSet-method
  (AlignmentsExperimentSet-class), 4
rveReads<-, 65
rveReads<-, AlignmentsExperimentSet-method
  (AlignmentsExperimentSet-class), 4
rveReadsType, 65
rveReadsType, AlignmentsExperimentSet-method
  (AlignmentsExperimentSet-class), 4
rveReadsType<-, 66
rveReadsType<-, AlignmentsExperimentSet-method
  (AlignmentsExperimentSet-class), 4
stat_bezier (geom_bezier), 40
unassignedCount, 67
unassignedCount, AlignmentsExperimentSet-method
  (AlignmentsExperimentSet-class), 4
unassignedData, 67
unassignedData, AlignmentsExperimentSet-method
  (AlignmentsExperimentSet-class), 4
unassignedData<-, 68
unassignedData<-, AlignmentsExperimentSet-method
  (AlignmentsExperimentSet-class), 4
upperGroups, 68
writeAlignments, 69
writeAlignments, AlignmentsExperimentSet-method
  (AlignmentsExperimentSet-class), 4