Package 'posDemux'

November 12, 2026

Type Package

Title Positional combinatorial sequence demultiplexer

Encoding UTF-8

Version 1.0.0

Date 2025-11-05

Description Demultiplexing and filtering utilities intended for reads with combinatorial barcodes (i.e. PETRI-seq and SPLiT-seq). The demultiplexer algorithm uses the position of the segments to extract and compare the barcodes with the reference (whitelist).
A Shiny application is provided to interactively select cutoffs for which barcode combinations to keep.

License AGPL (>= 3)

Depends R (>= 4.6.0)

Imports Biostrings, ggplot2, methods, assertthat, glue, magrittr, dplyr, rlang, ShortRead, readr, shiny, purrr

LinkingTo Rcpp, Biostrings, IRanges, S4Vectors, XVector

biocViews SequenceMatching, Sequencing, Software, RNASeq

LazyData FALSE

URL https://github.com/yaccos/posDemux, https://yaccos.github.io/posDemux/

BugReports https://github.com/yaccos/posDemux/issues

RoxygenNote 7.3.3

Roxygen list(markdown = TRUE)

Suggests testthat, devtools, DNABarcodes, knitr, rmarkdown, tibble, tidyr, BiocStyle, RefManageR, sessioninfo, DBI, chunked, RSQLite, dbplyr

Config/testthat/edition 3

VignetteBuilder knitr

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

git_branch RELEASE_3_23

git_last_commit 053127f

git_last_commit_date 2026-04-28

Repository Bioconductor 3.23

Date/Publication 2026-11-11

Author Jakob Peder Pettersen [aut, cre] (ORCID:
<https://orcid.org/0000-0002-3485-1634>),
Centre for new antibacterial strategies (CANS) [fnd]

Maintainer Jakob Peder Pettersen <jakobpeder.pettersen@gmail.com>

Contents

bc_to_freq_cutoff 2
combinatorial_demultiplex 4
create_freq_table 5
create_summary_res 7
filter_demultiplex_res 9
freq_plot 11
interactive_bc_cutoff 13
posDemux 14
row_match 14
streaming_callbacks 15
streaming_demultiplex 17

Index **20**

`bc_to_freq_cutoff` *Convert between cutoff types*

Description

There are at least two ways to specify the cutoff to use when selecting barcode combinations (cells) for further analysis. One way is to specify the number of barcode combinations to keep, effectively keeping a given number of barcode combinations with the highest frequencies. The other way is to specify the frequency cutoff directly without regard to the number of barcode combination to keep. In the former case, `bc_to_freq_cutoff()` is used to find the corresponding frequency cutoff, whereas in the latter case `freq_to_bc_cutoff()` is used to find the corresponding barcode cutoff.

Usage

```
bc_to_freq_cutoff(freq_table, cutoff)

freq_to_bc_cutoff(freq_table, cutoff)
```

Arguments

freq_table	The frequency table from create_freq_table(). In case the table is derived from another source, it must be sorted in descending order of frequency.
cutoff	Integer vector, the cutoff values to be converted.

Details

In the edge case of the barcode threshold being zero, the frequency cutoff is set to the maximum frequency in the table plus one. This feature makes sure that the cutoff line is visible in the frequency plot.

Value

Integer, the converted cutoff values.

Examples

```
library(purrr)
library(Biostrings)
input_fastq <- system.file(
    "extdata", "PETRI-seq_forward_reads.fq.gz",
    package = "posDemux")
reads <- readDNAStringSet(input_fastq, format = "fastq")
barcode_files <- system.file(
    "extdata/PETRI-seq_barcodes", c(bc1 = "bc1.fa", bc2 = "bc2.fa",
    bc3 = "bc3.fa"), package = "posDemux")
names(barcode_files) <- paste0("bc", 1L:3L)
barcode_index <- map(barcode_files, readDNAStringSet)
barcodes <- barcode_index[c("bc3", "bc2", "bc1")]
sequence_annotation <- c(UMI = "P", "B", "A", "B", "A", "B", "A")
segment_lengths <- c(7L, 7L, 15L, 7L, 14L, 7L, NA_integer_)
demultiplex_res <- posDemux::combinatorial_demultiplex(
    reads, barcodes = barcodes, segments = sequence_annotation,
    segment_lengths = segment_lengths)
filtered_res <- filter_demultiplex_res(demultiplex_res, allowed_mismatches = 1L)
freq_table <- create_freq_table(filtered_res$demultiplex_res$assigned_barcodes)

bc_cutoff <- c(100L, 204L, 50L, 655L)
freq_cutoff <- bc_to_freq_cutoff(freq_table, bc_cutoff)
# Note: The reconstructed barcode cutoff is not equal to
# the original due to ties in the frequency table
reconstructed_bc_cutoff <- freq_to_bc_cutoff(freq_table, freq_cutoff)
# The frequency cutoff is still preserved through these conversions
reconstruced_freq_cutoff <- bc_to_freq_cutoff(
    freq_table, reconstructed_bc_cutoff)
```

combinatorial_demultiplex *Combinatorial demultiplexer*

Description

This function performs segmenting of sequences and combinatorial demultiplexing and segmenting of sequences. The sequences have a structure as defined by the argument `segments` with corresponding lengths in `segment_lengths`. As segmentation and extraction can take place from either end, a single middle segment can be variadic in length. There are three types of segments: Adapter, Barcode and Payload. The adapter is trimmed and ignored, the barcode is used for demultiplexing, and the payload is kept after segmenting and demultiplexing and returned from the function. If there are multiple payload segments, then each segment constitutes its own segment in a list. For type stability reasons, such a list is returned also when there is zero or one payload segments. The barcodes can be positioned at either end of the sequences, but no barcode can (for obvious reasons) be variadic in length.

Usage

```
combinatorial_demultiplex(sequences, barcodes, segments, segment_lengths)
```

Arguments

`sequences`	A `XStringSet` object, the sequences to be demultiplexed.
`barcodes`	A list of `XStringSet` objects in the same order they appear in the `sequences`, the barcodes to be used for demultiplexing. All of the barcodes in each `XStringSet` must have the same length as specified by the `segment_lengths` argument and be named. For computational reasons, the maximum possible length of an individual barcode is 127.
`segments`	Character vector showing the segments of the sequences from 5' end to 3' end. The code applied is as follows: • `'A'`: Adapter (often referred to as linker), is trimmed and ignored • `'B'`: Barcode, used for demultiplexing • `'P'`: Payload, sequence to be kept after trimming and demultiplexing (e.g. cDNA or UMI). If this vector is named, this will determine the names of the payload sets. Names of the barcode sets will be determined by the names of the argument `barcodes` (if any).
`segment_lengths`	Integer vector with the same length as `segments`, lengths of the segments provided in the same order as in `segments`. Up to one of the non-barcode segments can have its length set to `NA` which means it is considered a variadic length segment.

Details

If there are two barcodes both having the minimum number of mismatches the first one will be selected. It is therefore important to choose the error tolerance to be equal or less than the redundancy of the barcodes. All sequences are assumed to be long enough for all segments to be extracted. Otherwise, an error is raised.

Value

A list with the following elements:

- `assigned_barcodes`: A `character` matrix with the names of the assigned barcodes as elements. The rows correspond to the sequences and the columns to the barcode segments.
- `mismatches`: An `integer` matrix with the number of mismatches between the assigned barcodes and the sequences. The rows correspond to the sequences and the columns to the barcode segments.
- `payload`: A list of `XStringSet` objects, each containing the results for a payload segment.
- `barcodes`: The `barcodes` argument passed into the function. It is included in order to ease downstream processing.

Examples

```
library(purrr)
library(Biostrings)
sequence_annotation <- c(UMI = "P", "B", "A", "B", "A", "B", "A")
segment_lengths <- c(7L, 7L, 15L, 7L, 14L, 7L, NA_integer_)
barcode_files <- system.file(
    "extdata/PETRI-seq_barcodes",
    c(bc1 = "bc1.fa", bc2 = "bc2.fa", bc3 = "bc3.fa"),
    package = "posDemux"
    )
names(barcode_files) <- paste0("bc", 1L:3L)
barcode_index <- map(barcode_files, readDNAStringSet)

barcodes <- barcode_index[c("bc3", "bc2", "bc1")]
input_fastq <- system.file(
    "extdata", "PETRI-seq_forward_reads.fq.gz", package = "posDemux")
reads <- readDNAStringSet(input_fastq, format = "fastq")
demultiplex_res <- combinatorial_demultiplex(
    reads, barcodes = barcodes, segments = sequence_annotation,
    segment_lengths = segment_lengths
    )
```

`create_freq_table` *Frequency table*

Description

Creates a sorted frequency table of each of the observed barcode combinations. This function is indended to be used after running `filter_demultiplex_res()` and before creating frequency plots, knee plots, or selecting the number of barcodes to include.

Usage

```
create_freq_table(assigned_barcodes)
```

Arguments

`assigned_barcodes`
: A character or integer matrix, corresponding to the field `assigned_barcodes` from `combinatorial_demultiplex()` or the field `demultiplex_res$assigned_barcodes` from `filter_demultiplex_res()`.

Value

A data frame where each row corresponds to a unique observed barcode combination. The rows are sorted in descending order of frequency. The first columns specify the barcode assignment (e.g `bc3`, `bc2`, `bc1`) and the last columns were the following:

- `frequency`: The number of reads with the barcode combination.
- `cumulative_frequency`: The cumulative frequency of the barcode combination counted from the top.
- `fraction`: The fraction of reads with the barcode combination.
- `cumulative_fraction`: The cumulative fraction of the barcode combination counted from the top.

Examples

```
library(purrr)
library(Biostrings)
input_fastq <- system.file(
    "extdata", "PETRI-seq_forward_reads.fq.gz", package = "posDemux")
reads <- readDNAStringSet(input_fastq, format = "fastq")
barcode_files <- system.file(
    "extdata/PETRI-seq_barcodes",
    c(bc1 = "bc1.fa", bc2 = "bc2.fa", bc3 = "bc3.fa"),
    package = "posDemux"
    )
names(barcode_files) <- paste0("bc", 1L:3L)
barcode_index <- map(barcode_files, readDNAStringSet)
barcodes <- barcode_index[c("bc3", "bc2", "bc1")]
sequence_annotation <- c(UMI = "P", "B", "A", "B", "A", "B", "A")
segment_lengths <- c(7L, 7L, 15L, 7L, 14L, 7L, NA_integer_)
demultiplex_res <- posDemux::combinatorial_demultiplex(
    reads, barcodes = barcodes, segments = sequence_annotation,
    segment_lengths = segment_lengths
    )
```

```
filtered_res <- filter_demultiplex_res(demultiplex_res, allowed_mismatches = 1L)
freq_table <- create_freq_table(filtered_res$demultiplex_res$assigned_barcodes)
```

`create_summary_res` *Create a summary of match filtering*

Description

`create_summary_res()` is a helper function in order to create a summary of the demultiplexing and following match filtering. It is not designed to be invoked directly, but its results will be returned automatically from `filter_demultiplex_res()`. This returned object has it own method for printing the result in a user-friendly manner.

Usage

```
create_summary_res(
  retained,
  barcodes,
  assigned_barcodes,
  allowed_mismatches,
  mismatches
)

## S3 method for class 'demultiplex_filter_summary'
print(x, ...)
```

Arguments

`retained`	Logical vector with the same length as the number of reads in the input to the demultiplexer. `TRUE` if the corresponding read is retained. Corresponds to the field `retained` of the output of `filter_demultiplex_res()`.
`barcodes`	A list of `XStringSet` objects, the barcodes which were used for demultiplexing.
`assigned_barcodes`	Character matrix of the assigned barcodes only including the onces within the mismatch threshold. Corresponds to of the field `demultiplex_res$assigned_barcodes` of `filter_demultiplex_res()`.
`allowed_mismatches`	Integer vector of length one or the same length as the number of barcode segments; the threshold Hamming distance. All reads having a number of mismatches above this number in any of the barcodes will be filtered away.
`mismatches`	Integer matrix of the number of mismatches of each assigned barcode. Corresponds to the field `mismatches` of `combinatorial_demultiplex()`.
`x`	An object of class `demultiplex_filter_summary` from `create_summary_res()`.
`...`	Ignored

Details

Following a uniform distribution of barcodes, the expected number of barcode collisions (observed barcodes combinations being composed of two or more features) is given by

$$N\left(1 - e^{-\lambda} - \lambda e^{-\lambda}\right),$$

where N is the number of possible barcode combinations and λ is in this summary referred to as the collision lambda:

$$\lambda = \frac{n}{N},$$

where n is the number of features. However, n is unknown as we cannot know how many features there were originally due to potential collisions. Utilizing the fact that the expected observed number of barcodes is given by

$$N\left(1 - e^{-\lambda}\right),$$

we can correct the estimate for λ from the known value of the observed barcode combinations, and thus estimate the number of features and barcode collisions.

While each unique feature can be conceptually thought of as single cell with its transcripts, realistic datasets have many features with relatively small numbers of reads which are artifacts and unlikely to correspond to true cells.

Value

`create_summary_res()` returns a list of S3 class `demultiplex_filter_summary` providing diagnostics for the filtering process. It contains the the following fields:

- `n_reads`: The total number of reads in the dataset before filtering.
- `n_removed`: The number of reads removed because demultiplexing failed.
- `n_barcode_sets`: The number of barcode sets.
- `n_barcode_combinations`: The possible number of barcode combinations.
- `n_unique_barcodes`: The number of observed unique barcode combinations (i.e. features which may be cells) detected after filtering mismatches.
- `n_estimated_features`: The estimated number of features having a detected combination of barcodes. This number will always be greater or equal than `n_unique_barcodes` due to barcode collisions.
- `observed_collision_lambda`: The ratio of observed barcode combinations divided by the total number of possible barcode combinations.
- `corrected_collision_lambda`: The ratio of estimated number of features to the total number of possible barcode combinations.
- `expected_collisions`: The statistically expected number of barcode collisions or more precicely the expected number of observed barcodes which correspond to two or more features.
- `barcode_summary`: A list containing a summary for each barcode set. Each element contains the following:
  - `width`: The width (number of nucleotides) of the barcode set.
  - `n_barcodes`: Number of query barcodes.
  - `n_allowed_mismatches`: Number of allowed mismatches for the barcode set.

- n_removed: Number of reads having too many mismatches for this barcode set.
- mismatch_frame: A data.frame with the two columns, n_mismatches and frequency showing the number of reads for each of the allowed number of mismatches for the given barcode set.

The print() method returns its output invisibly.

Examples

```
library(purrr)
library(Biostrings)
input_fastq <- system.file(
    "extdata", "PETRI-seq_forward_reads.fq.gz", package = "posDemux")
reads <- readDNAStringSet(input_fastq, format = "fastq")
barcode_files <- system.file(
    "extdata/PETRI-seq_barcodes",
    c(bc1 = "bc1.fa", bc2 = "bc2.fa", bc3 = "bc3.fa"), package = "posDemux"
    )
names(barcode_files) <- paste0("bc", 1L:3L)
barcode_index <- map(barcode_files, readDNAStringSet)
barcodes <- barcode_index[c("bc3", "bc2", "bc1")]
sequence_annotation <- c(UMI = "P", "B", "A", "B", "A", "B", "A")
segment_lengths <- c(7L, 7L, 15L, 7L, 14L, 7L, NA_integer_)
demultiplex_res <- posDemux::combinatorial_demultiplex(
    reads, barcodes = barcodes, segments = sequence_annotation,
    segment_lengths = segment_lengths
    )
filtered_res <- filter_demultiplex_res(demultiplex_res, allowed_mismatches = 1L)
freq_table <- create_freq_table(filtered_res$demultiplex_res$assigned_barcodes)
print(filtered_res$summary_res)

# This also works, but is usually not necessary to call directly
alternative_summary_res <- create_summary_res(
    retained = filtered_res$retained, barcodes = barcodes,
    assigned_barcodes = filtered_res$demultiplex_res$assigned_barcodes,
    allowed_mismatches = 1L, mismatches = demultiplex_res$mismatches
    )
```

---

filter_demultiplex_res

Filter demultiplexed reads

Description

Filters the demultiplexed reads from combinatorial_demultiplex() such that any read exceeding the number of allowed mismatches for any of the barcodes is removed. The function gives diagnostic information on the number of reads removed per barcode and the total number of reads removed.

Usage

```
filter_demultiplex_res(demultiplex_res, allowed_mismatches)
```

Arguments

demultiplex_res	Unprocessed output from combinatorial_demultiplex().
allowed_mismatches	Integer vector of length one or the same length as the number of barcode segments; the threshold Hamming distance. All reads having a number of mismatches above this number in any of the barcodes will be filtered away.

Details

The value of n_removed does not in general equal the sum of n_removed_per_barcode since a read can have too many mismatches with multiple barcodes.

Value

A list with the following elements:

- demultiplex_res: The contents of the input argument demultiplex_res with the sequences filtered.
- retained: Logical vector with the same length as the number of reads in the input. TRUE if the corresponding read is retained. Useful for future filtering of paired-end reads.
- summary_res: Result of create_summary_res() called on the results of filtering.

See Also

create_summary_res()

Examples

```
library(purrr)
library(Biostrings)
input_fastq <- system.file(
    "extdata", "PETRI-seq_forward_reads.fq.gz", package = "posDemux")
reads <- readDNAStringSet(input_fastq, format = "fastq")
barcode_files <- system.file(
    "extdata/PETRI-seq_barcodes",
    c(bc1 = "bc1.fa", bc2 = "bc2.fa", bc3 = "bc3.fa"), package = "posDemux"
    )
names(barcode_files) <- paste0("bc", 1L:3L)
barcode_index <- map(barcode_files, readDNAStringSet)
barcodes <- barcode_index[c("bc3", "bc2", "bc1")]
sequence_annotation <- c(UMI = "P", "B", "A", "B", "A", "B", "A")
segment_lengths <- c(7L, 7L, 15L, 7L, 14L, 7L, NA_integer_)
demultiplex_res <- posDemux::combinatorial_demultiplex(
    reads, barcodes = barcodes, segments = sequence_annotation,
    segment_lengths = segment_lengths
```

```
    )
filtered_res <- filter_demultiplex_res(demultiplex_res, allowed_mismatches = 1L)
freq_table <- create_freq_table(filtered_res$demultiplex_res$assigned_barcodes)
print(filtered_res$summary_res)

# This also works, but is usually not necessary to call directly
alternative_summary_res <- create_summary_res(
    retained = filtered_res$retained, barcodes = barcodes,
    assigned_barcodes = filtered_res$demultiplex_res$assigned_barcodes,
    allowed_mismatches = 1L, mismatches = demultiplex_res$mismatches
    )
```

`freq_plot` *Diagnostic plots from demultiplexing*

Description

Diagnostic plots for determining the effect of the barcode cutoff. `freq_plot()` shows a histogram or distribution plot of the number of reads for each barcode combination, whereas `knee_plot()` shows the cumulative fraction of reads ranked by the frequency of the barcode combinations in descending order.

Usage

```
freq_plot(
  freq_table,
  cutoff = NULL,
  type = "histogram",
  log_scale_x = TRUE,
  log_scale_y = FALSE,
  scale_by_reads = FALSE
)

knee_plot(freq_table, cutoff = NULL)
```

Arguments

`freq_table`	The frequency table from `create_freq_table()`.
`cutoff`	Optional scalar numeric, the x-coordinate for drawing a vertical dashed line in the plots in order to indicate the cutoff. Please note that this argument is interpreted literally, meaning that in order to correctly display the same cutoff on both type of plots, the cutoff value has to be transformed. In order to safely convert between the two types of cutoffs, use the functions `bc_to_freq_cutoff()` and `freq_to_bc_cutoff()`.
`type`	The type of frequency plot to make, either `'histogram'` or `'density'`.
`log_scale_x`	Logical: Should a log scale be applied to the x-axis of the frequency plot?
`log_scale_y`	Logical: Should a log scale be applied to the y-axis of the frequency plot?

scale_by_reads Logical: Should the y-axis of the plot be scaled by the number of reads on the x-axis?

Value

A ggplot object which can be displayed immediately or further modified.

See Also

bc_to_freq_cutoff() freq_to_bc_cutoff()

Examples

```
library(purrr)
library(Biostrings)
input_fastq <- system.file(
    "extdata", "PETRI-seq_forward_reads.fq.gz", package = "posDemux")
reads <- readDNAStringSet(input_fastq, format = "fastq")
barcode_files <- system.file(
    "extdata/PETRI-seq_barcodes",
    c(bc1 = "bc1.fa", bc2 = "bc2.fa", bc3 = "bc3.fa"),
    package = "posDemux"
    )
names(barcode_files) <- paste0("bc", 1L:3L)
barcode_index <- map(barcode_files, readDNAStringSet)
barcodes <- barcode_index[c("bc3", "bc2", "bc1")]
sequence_annotation <- c(UMI = "P", "B", "A", "B", "A", "B", "A")
segment_lengths <- c(7L, 7L, 15L, 7L, 14L, 7L, NA_integer_)
demultiplex_res <- posDemux::combinatorial_demultiplex(
    reads, barcodes = barcodes, segments = sequence_annotation,
    segment_lengths = segment_lengths)
filtered_res <- filter_demultiplex_res(demultiplex_res, allowed_mismatches = 1L)
freq_table <- create_freq_table(filtered_res$demultiplex_res$assigned_barcodes)

bc_cutoff <- 500L
# Notice the bend (knee) of the curve
knee_plot(freq_table, cutoff = bc_cutoff)

# Note that we must convert the cutoff when constructing the frequency plot
freq_cutoff <- bc_to_freq_cutoff(freq_table, bc_cutoff)

# This is the most basic type of frequency plot which can be made,
# but it is a bit hard to interpret whether a selected cutoff is sensible
freq_plot(
    freq_table, cutoff = freq_cutoff, type = "histogram",
    log_scale_x = FALSE, log_scale_y = FALSE, scale_by_reads = FALSE)

# For most practical purposes, this is the most informative version
freq_plot(freq_table, cutoff = freq_cutoff, type = "density",
    log_scale_x = TRUE, log_scale_y = FALSE,
    scale_by_reads = TRUE)
```

`interactive_bc_cutoff` *Interactive cutoff selection*

Description

Returns an interactive Shiny application for determining cutoff from knee plot and barcode frequency plot. The user will select the appropriate number of barcode combinations (i.e. distinct cells) to keep for further analysis. Usually, this is done by aiming for the 'knee' of the knee plot in order to keep most reads, while at the same time removing barcode combinations which are either artifacts or broken cells.

Usage

```
interactive_bc_cutoff(freq_table)
```

Arguments

`freq_table`	The frequency table from `create_freq_table()`.

Value

A `shiny.appobj` which launches when printed and returns the last selected cutoff (invisibly) when it stops.

Examples

```
if (interactive()) {
    library(purrr)
    library(shiny)
    library(Biostrings)
    input_fastq <- system.file(
        "extdata", "PETRI-seq_forward_reads.fq.gz", package = "posDemux")
    reads <- readDNAStringSet(input_fastq, format = "fastq")
    barcode_files <- system.file(
        "extdata/PETRI-seq_barcodes",
        c(bc1 = "bc1.fa", bc2 = "bc2.fa", bc3 = "bc3.fa"),
        package = "posDemux"
        )
    names(barcode_files) <- paste0("bc", 1L:3L)
    barcode_index <- map(barcode_files, readDNAStringSet)
    barcodes <- barcode_index[c("bc3", "bc2", "bc1")]
    sequence_annotation <- c(UMI = "P", "B", "A", "B", "A", "B", "A")
    segment_lengths <- c(7L, 7L, 15L, 7L, 14L, 7L, NA_integer_)
    demultiplex_res <- posDemux::combinatorial_demultiplex(
        reads, barcodes = barcodes, segments = sequence_annotation,
        segment_lengths = segment_lengths
        )
    filtered_res <- filter_demultiplex_res(
        demultiplex_res, allowed_mismatches = 1L)
```

```
    freq_table <- create_freq_table(
        filtered_res$demultiplex_res$assigned_barcodes)
    selection_app <- interactive_bc_cutoff(freq_table)
    selected_bc_cutoff <- runApp(selection_app)
}
```

---

posDemux *Utilities for segmented sequences with combinatorial barcoding*

Description

Provides tools for handling reads with combinatorial barcodes, and multiple adapter regions. This includes utilities for demultiplexing, sequence segmenting and filtering. The package is intended to work with single-cell RNA-seq data with multiple barcoding (e.g. SPLiT-seq and PETRI-seq).

Author(s)

Jakob Peder Pettersen jakobpeder.pettersen@gmail.com

---

row_match *Row matching of tables*

Description

This function extends the functionality of `%in%` for finding which rows in the first argument exist in the second.

Usage

```
row_match(x, table)
```

Arguments

x	A matrix or data frame which rows to be matched. Typically, this will be a matrix of assigned barcodes for each read.
table	A matrix or data frame with the rows to be matched against. Typically, this will be the top portion of a frequency table.

Details

As this function is intended to be used for data frames containing more than just the barcodes, the intersection of the column names is used for matching. As opposed to `base::match()`, this function is implemented more efficiently by converting each row into a numeric encoding before matching.

For technical reasons, it is not permitted for the product of the number of the unique values of the columns in `table` to exceed $2^{32} - 1 \approx 2.1 \cdot 10^9$.

Value

Logical vector, for each row in `x`, is the same row found in `table`?

See Also

`create_freq_table()` for how frequency tables are constructed, `combinatorial_demultiplex()` for more information on the matrix of assigned barcodes, and `dplyr::inner_join()` for a function with similar functionality.

Examples

```
barcode_table <- data.frame(
    read = c("seq_1", "seq_2", "seq_3", "seq_4"),
    bc1 = c("A", "B", "C", "B"),
    bc2 = c("A", "C", "A", "A")
    )

freq_table <- data.frame(
    bc1 = c("B", "B", "C", "A"),
    bc2 = c("A", "C", "A", "A"),
    frequency = c(200L, 100L, 50L, 10L)
    )

freq_cutoff <- 100L

selected_freq_table <- freq_table[freq_table$frequency >= freq_cutoff, ]

selected_rows <- row_match(barcode_table, selected_freq_table)
selected_barcode_table <- barcode_table[selected_rows, ]
```

`streaming_callbacks` *Suggested setup for FASTQ streaming*

Description

Even though the user can define the arguments `state_init`, `loader`, and `archiver` for `streaming_demultiplex()`, this approach is only recommended for advanced users. This functions defines a premade combinations of these three arguments which should be suitable in most cases. The loader streams a FASTQ file (having multiple files is also supported) and the archiver outputs a data frame to file consisting of the read name (`read`), the sequences of all payloads (e.g. `UMI`), and barcode assignments (`c('bc3','bc2','bc1')`).

Usage

```
streaming_callbacks(
  input_file,
  output_table_file,
  chunk_size = 1e+06,
```

```
  verbose = TRUE,
  min_width = NULL
)
```

Arguments

input_file	A character vector containing the paths to the FASTQ files to be used for demultiplexing. Often this is only one file, but multiple files are supported such that demultiplexing data from multiple lanes does not require merging the lanes first.
output_table_file	The path to which the output barcode table will be written.
chunk_size	Integer, the number of reads to process in each chunk.
verbose	Logical scalar: Should the progress be displayed?
min_width	Optional integer scalar: Minimum width of the sequences to keep. For reads which are shorter than this, a warning is emitted and the reads are removed and ignored and thus not appear in any statistics. The data loader is **not** supposed to be used as a length filter, so this option is more like an escape hatch for being able to deal with sequences which have not been properly filtered beforehand.

Details

If the read names have any spaces in them, the loader will only keep the portion of the read name preceding the first space. This is due to the Illumina platform's behavior of encoding the sequencing direction (forward or reverse) past the space. Keeping the read names with the space is usually not desirable as it makes the resulting barcode table more confusing and makes it more difficult to group the forward and reverse reads together afterwards.

Value

A list with the following elements, all of which are intended to be used as the corresponding arguments to `streaming_demultiplex()`:

- `state_init`
- `loader`
- `archiver`

See Also

`ShortRead::FastqStreamer()` which is used as a backend for the data loader.

Examples

```
library(purrr)
library(Biostrings)
input_fastq <- system.file(
    "extdata", "PETRI-seq_forward_reads.fq.gz", package = "posDemux")
output_barcode_table <- tempfile(pattern = "barcode_table", fileext = ".txt")
```

```
callbacks <- streaming_callbacks(
    input_file = input_fastq, output_table_file = output_barcode_table,
    chunk_size = 10000, verbose = TRUE)
barcode_files <- system.file(
    "extdata/PETRI-seq_barcodes",
    c(bc1 = "bc1.fa", bc2 = "bc2.fa", bc3 = "bc3.fa"),
    package = "posDemux"
    )
names(barcode_files) <- paste0("bc", 1L:3L)
barcode_index <- map(barcode_files, readDNAStringSet)
barcodes <- barcode_index[c("bc3", "bc2", "bc1")]
sequence_annotation <- c(UMI = "P", "B", "A", "B", "A", "B", "A")
segment_lengths <- c(7L, 7L, 15L, 7L, 14L, 7L, NA_integer_)
streaming_summary_res <- streaming_demultiplex(
    state_init = callbacks$state_init, loader = callbacks$loader,
    archiver = callbacks$archiver, barcodes = barcodes, allowed_mismatches = 1L,
    segments = sequence_annotation, segment_lengths = segment_lengths
    )
```

`streaming_demultiplex` *Demultiplexing with streaming*

Description

This function provides an interface to `combinatorial_demultiplex()` and `filter_demultiplex_res()` such that reads are streamed in chunks instead having to load everything at once, hence reducing memory consumption. It accepts two functions which are called once per chunk: A data loader function for producing the sequences of the chunk and an archiver writing the results to file.

Usage

```
streaming_demultiplex(
  state_init,
  loader,
  archiver,
  barcodes,
  allowed_mismatches,
  segments,
  segment_lengths
)
```

Arguments

`state_init`	The initial state to pass into `loader`.
`loader`	Function loading the reads. It has the signature `f(state)`, where `state` is a user-defined object which is initialized to be `state_init` and for the subsequent iterations taken as the `state` field of the output of `archiver`. Its return value is a list with the following fields:

- `state`: The state to be passed into `archiver`.
- `sequences`: A `XStringSet` object, the sequences to be demultiplexed in the current chunk.
- `should_terminate`: A scalar logical. If `TRUE`, the demultiplexing process terminates and the final results are returned. Notice that this termination happens before the sequences of the final call to `loader` are demultiplexed.

`archiver` Function taking care of archiving the demultiplexed results. Its arguments are:

- `state`: The state of the process returned by `loader`.
- `filtered_res`: The output from running `combinatorial_demultiplex()` and `filter_demultiplex_res()` on the data expect that the field `summary_res` is missing.

Its output is a state object fed into the next call to `loader`.

`barcodes` A list of `XStringSet` objects in the same order they appear in the `sequences`, the barcodes to be used for demultiplexing. All of the barcodes in each `XStringSet` must have the same length as specified by the `segment_lengths` argument and be named. For computational reasons, the maximum possible length of an individual barcode is 127.

`allowed_mismatches`

Integer vector of length one or the same length as the number of barcode segments; the threshold Hamming distance. All reads having a number of mismatches above this number in any of the barcodes will be filtered away.

`segments` Character vector showing the segments of the sequences from 5' end to 3' end. The code applied is as follows:

- `'A'`: Adapter (often referred to as linker), is trimmed and ignored
- `'B'`: Barcode, used for demultiplexing
- `'P'`: Payload, sequence to be kept after trimming and demultiplexing (e.g. cDNA or UMI).

If this vector is named, this will determine the names of the payload sets. Names of the barcode sets will be determined by the names of the argument `barcodes` (if any).

`segment_lengths`

Integer vector with the same length as `segments`, lengths of the segments provided in the same order as in `segments`. Up to one of the non-barcode segments can have its length set to `NA` which means it is considered a variadic length segment.

Details

The data loader decides the size of each chunk. While this framework does not provide any restriction on the `state` object, the loader and archiver must be written such that the state objects they return are compatible. Since the data loader alone decides when to terminate, bad terminations crieria can cause a runaway loop. Usually, it will be useful to have a progress tracker of how many reads are demultiplexed. The framework itself does not implement this, so it is typically implemented into the archiver or loader.

For technical reasons, it is not possible to do streaming when the number of possible barcode combinations exceeds $2^{32} - 1 \approx 2.1 \cdot 10^9$.

Value

A list with three elements:

- `freq_table`: The frequency table for all reads, akin to the output of `create_freq_table()`.
- `summary_res`: The summary result of match filtering of all reads per `create_summary_res()`.
- `state_final`: The final state object returned from `loader`.

See Also

`filter_demultiplex_res()`, `combinatorial_demultiplex()`, `create_freq_table()`, and `create_summary_res()` for the underlying processing.

Examples

```
library(purrr)
library(Biostrings)
input_fastq <- system.file(
    "extdata", "PETRI-seq_forward_reads.fq.gz", package = "posDemux")
output_barcode_table <- tempfile(pattern = "barcode_table", fileext = ".txt")

callbacks <- streaming_callbacks(
    input_file = input_fastq, output_table_file = output_barcode_table,
    chunk_size = 10000, verbose = TRUE)
barcode_files <- system.file(
    "extdata/PETRI-seq_barcodes",
    c(bc1 = "bc1.fa", bc2 = "bc2.fa", bc3 = "bc3.fa"),
    package = "posDemux"
    )
names(barcode_files) <- paste0("bc", 1L:3L)
barcode_index <- map(barcode_files, readDNAStringSet)
barcodes <- barcode_index[c("bc3", "bc2", "bc1")]
sequence_annotation <- c(UMI = "P", "B", "A", "B", "A", "B", "A")
segment_lengths <- c(7L, 7L, 15L, 7L, 14L, 7L, NA_integer_)
streaming_summary_res <- streaming_demultiplex(
    state_init = callbacks$state_init, loader = callbacks$loader,
    archiver = callbacks$archiver, barcodes = barcodes, allowed_mismatches = 1L,
    segments = sequence_annotation, segment_lengths = segment_lengths
    )
```

Index

∗ **internal**
    posDemux, 14
%in%, *14*

base::match(), *14*
bc_to_freq_cutoff, 2
bc_to_freq_cutoff(), *11, 12*

combinatorial_demultiplex, 4
combinatorial_demultiplex(), *6, 7*, *9, 10*, *15*, *17–19*
create_freq_table, 5
create_freq_table(), *3*, *11*, *13*, *15*, *19*
create_summary_res, 7
create_summary_res(), *10*, *19*

dplyr::inner_join(), *15*

filter_demultiplex_res, 9
filter_demultiplex_res(), *6, 7*, *17–19*
freq_plot, 11
freq_to_bc_cutoff *(*bc_to_freq_cutoff*)*, 2
freq_to_bc_cutoff(), *11, 12*

ggplot, *12*

interactive_bc_cutoff, 13

knee_plot *(*freq_plot*)*, 11

posDemux, 14
posDemux-package *(*posDemux*)*, 14
print.demultiplex_filter_summary *(*create_summary_res*)*, 7

row_match, 14

shiny.appobj, *13*
ShortRead::FastqStreamer(), *16*
streaming_callbacks, 15
streaming_demultiplex, 17
streaming_demultiplex(), *15, 16*

XStringSet, *4, 5*, *7*, *18*