

Package: seqpac (via r-universe)

October 4, 2026

Type Package

Title Seqpac: A Framework for smallRNA analysis in R using
Sequence-Based Counts

Version 1.12.0

Description Seqpac provides functions and workflows for analysis of short sequenced reads. It was originally developed for small RNA analysis, but can be implemented on any sequencing raw data (provided as a fastq-file), where the unit of measurement is counts of unique sequences. The core of the seqpac workflow is the generation and subsequence analysis/visualization of a standardized object called PAC. Using an innovative targeting system, Seqpac process, analyze and visualize sample or sequence group differences using the PAC object. A PAC object in its most basic form is a list containing three types of data frames. - Phenotype table (P): Sample names (rows) with associated metadata (columns) e.g. treatment. - Annotation table (A): Unique sequences (rows) with annotation (columns), eg. reference alignments. - Counts table (C): Counts of unique sequences (rows) for each sample (columns). The PAC-object follows the rule: - Row names in P must be identical with column names in C. - Row names in A must be identical with row names in C. Thus P and A describes the columns and rows in C, respectively. The targeting system, will either target specific samples in P (pheno_target) or sequences in A (anno_target) and group them according to a target column in P and A, respectively (see vignettes for more details).

License GPL-3

Encoding UTF-8

LazyData false

RoxygenNote 7.3.3

URL <https://github.com/Oestlab/seqpac>

BugReports <https://github.com/OestLab/seqpac/issues>

biocViews Workflow, BasicWorkflow, GeneExpressionWorkflow,
EpigeneticsWorkflow, AnnotationWorkflow

Depends R (>= 4.5.0)

Imports Biostrings (>= 2.46.0), foreach (>= 1.5.1), GenomicRanges (>= 1.30.3), Rbowtie (>= 1.18.0), ShortRead (>= 1.36.1), tibble (>= 3.1.2), BiocParallel (>= 1.12.0), cowplot (>= 0.9.4), data.table (>= 1.14.0), digest (>= 0.6.27), doParallel (>= 1.0.16), dplyr (>= 1.0.6), factoextra (>= 1.0.7), FactoMineR (>= 1.41), ggplot2 (>= 4.0.2), IRanges (>= 2.12.0), parallel (>= 3.4.4), reshape2 (>= 1.4.4), rtracklayer (>= 1.38.3), stringr (>= 1.4.0), stats (>= 3.4.4), methods, S4Vectors, readr

Suggests benchmarkme (>= 0.6.0), DESeq2 (>= 1.18.1), GenomeInfoDb (>= 1.14.0), gginnards (>= 0.0.2), qqman (>= 0.1.8), rmarkdown, BiocStyle, knitr, testthat, UpSetR (>= 1.4.0), venneuler, R.utils, bigreadr, vroom

VignetteBuilder knitr

Config/pak/sysreqs cmake libfreetype6-dev make libbz2-dev libicu-dev libjpeg-dev liblzma-dev libpng-dev libuv1-dev libxml2-dev libssl-dev libx11-dev xz-utils zlib1g-dev

Repository Bioconductor 3.23

Date/Publication 2026-04-28 12:44:54 UTC

RemoteUrl <https://github.com/bioc/seqpac>

RemoteRef RELEASE_3_23

RemoteSha 32327358db6b9bfba00447e145fd26309278e7ba

Contents

add_reanno	3
anno	8
as.PAC	9
as.reanno	10
colnames,PAC-method	12
counts	13
filtsep_bin	14
full	15
import_reanno	17
length,PAC-method	19
make_conv	20
make_counts	22
make_cutadapt	26
make_PAC	28
make_pheno	30
make_reanno	32
make_trim	35
map_rangetype	39
map_reanno	41

merge_lanes	46
names,PAC-method	47
ncol,PAC-method	49
norm	50
nrow,PAC-method	51
overview	52
PAC-class	54
PAC_analyze	55
PAC_annotate	56
PAC_check	58
PAC_covplot	59
PAC_create	62
PAC_deseq	63
PAC_filter	65
PAC_filtsep	68
PAC_gtf	69
PAC_jitter	72
PAC_mapper	74
PAC_nbias	76
PAC_norm	78
PAC_pca	80
PAC_pie	82
PAC_saturation	84
PAC_sizedist	85
PAC_stackbar	87
PAC_summary	89
PAC_trna	91
pheno	96
reanno-class	97
rownames,PAC-method	99
show.PAC	100
simplify_reanno	101
summary	104
tRNA_class	105

Index**108**

add_reanno	<i>Generates biotype annotation from reannotation object</i>
------------	--

Description

add_reanno adds biotypes from reannotation to PAC.

Usage

```
add_reanno(
  reanno,
  mismatches = 0,
  type = "genome",
  bio_search,
  bio_perfect = FALSE,
  genome_max = 10,
  merge_pac = NULL
)
```

Arguments

reanno	A reannotation list object generated by make_reanno , with an Overview data.frame and a Full_anno list.
mismatches	Integer indicating the number of mismatches that should be reported. Can never have a higher number than was originally specified with the map_reanno function. While default=0, reporting the maximum number is recommended. Note that the mismatch information is only added to the report. Further classification can be generated with the simplify_reanno function.
type	Character indicating what type of mapping that has been performed. If type="genome", the reanno object is expected to have been mapped against a genome or larger fasta references like chromosomes. This may include coordinates as controlled by map_reanno and the import_reanno functions. See genome_max on how to control how coordinates are reported. If coordinates are missing, only hit or no hit will be reported. If type="biotype", then coordinates are not expected. Reference hits will instead be classified according to the biotype search terms in bio_search.
bio_search	List of character vectors indicating search terms targeting feature names in the original reference fasta files. These search terms will be parsed to grepl as regular expressions. List names must match names of the references in the reanno object. Classifications will be reported with reference name + search term (e.g. "Ensembl_trna").
bio_perfect	Logical whether the function should allow that not all hits against the references must have a unique bio_search term. If perfect=FALSE (default) all references hits that was not caught by a search term will be annotated as reference + other (e.g. "Ensembl_other"). In case perfect=TRUE, the function will throw an error if the search terms do not catch all reference hits. Can be used to ensure that all hits receive a classification.
genome_max	Integer or character indicating the number of maximum coordinates to be reported when type="genome". If the number of hits exceeds genome_max it will be indicated in the classification and only the first hits up to max_hits will be reported. This is useful to handling sequences that multimap. If genome_max="all", all coordinates will be reported which may dramatically affect performance. (default=10)

<code>merge_pac</code>	PAC object. If a PAC object is provided in <code>merge_pac</code> , then the function will automatically merge the Anno table of the PAC object with the newly generated annotations. As default, <code>merge_pac=NULL</code> will return a tibble data.frame instead.
------------------------	--

Details

Given a reanno object generated by `make_reanno` this function will either extract genome coordinates or classify biotypes using a list of search terms (see details). Information will be compiled into a data frame with the same order of sequences as in the original PAC master file.

Please rename the column names of your previously added columns if running `map_reanno` more than two times using the same type input ("genome" or "biotype").

Value

Data frame with mismatch and coordinate or biotype information (search term hits) that can directly be added to an Anno data.frame of a PAC object, given that the PAC object was used in the reannotation workflow. Annotations can be further simplified using the `simplify_reanno` function.

See Also

<http://bowtie-bio.sourceforge.net/index.shtml> for information about Bowtie and <https://github.com/OestLab/segpac> for updates on the current package.

Other PAC reannotation: `PAC_annotate()`, `as.reanno()`, `import_reanno()`, `make_conv()`, `make_reanno()`, `map_reanno()`, `simplify_reanno()`

Examples

[illegible]

```

rrna_dir <- dirname(rrna_file)

if(!sum(stringr::str_count(list.files(rrna_dir), ".ebwt")) ==6){
  Rbowtie::bowtie_build(rrna_file,
                        outdir=rrna_dir,
                        prefix="rRNA", force=TRUE)
}

## Then load a PAC-object and remove previous mapping from anno:
load(system.file("extdata", "drosophila_sRNA_pac_filt_anno.Rdata",
                package = "seqpac", mustWork = TRUE))
anno(pac) <- anno(pac)[,1, drop = FALSE]

ref_paths <- list(trna= trna_file, rrna= rrna_file)

## You may add an output path of your choice, but here we use a temp folder:
output <- paste0(tempdir(),"/seqpac/test")

## Then map the PAC-object against the fasta references.
# Warning: if you use your own data, you may want to use override=FALSE, to avoid
# deleting previous mapping by mistake.

map_reanno(pac, input=ref_paths, output=output,
           type="internal", mismatches=0, import="biotype",
           threads=2, keep_temp=FALSE, override=TRUE)

## Then import and generate a reanno-object of the temporary bowtie-files
reanno_biotype <- make_reanno(output, PAC=pac, mis_fasta_check = TRUE)

## Now make some search terms against reference names to create shorter names
# Theses can be used to create factors in downstream analysis
# One search hit (regular expressions) gives one new short name

bio_search <- list(
  rrna=c("5S", "5.8S", "12S", "16S", "18S", "28S", "pre_S45"),
  trna =c("_tRNA", "mt:tRNA"))

## You can merge directly with your PAC-object by adding your original
# PAC-object, that you used with map_reanno, to merge_pac option.

pac <- add_reanno(reanno_biotype, bio_search=bio_search,
                 type="biotype", bio_perfect=FALSE,
                 mismatches = 0, merge_pac=pac)

table(anno(pac)$mis0_bio)

#####
## Similarly, you can use Bowtie indexed genome fasta references

```

```

## But don't forget to use import="genome" for coordinate import
#
# ref_paths <- list(genome="<path_to_bowtie_indexed_fasta>")
# output_genome <- "<your_path_to_output_folder>"
#
## We can actually map against several genomes simultaneously:
# (to illustrate the principle we run the mycoplasma genome twice)
# ref_paths <- list(genome1=mycoplasma_file, genome2=mycoplasma_file)
# ### Run map_reanno
# map_reanno(PAC=pac, input=ref_paths, output=output,
#             type="internal", mismatches=0, import="genome",
#             threads=2, keep_temp=TRUE, override=TRUE)
# reanno_genome <- make_reanno(output, PAC=pac, mis_fasta_check = TRUE)
#####
#### The trick to succeed with bio_perfect=TRUE

## Run add_reanno with bio_perfect="FALSE" (look where "Other=XX" occurs)
# Use the bio
#' ## Add a search terms that catches the other rrna

anno <- add_reanno(reanno_biotype, bio_search=bio_search,
                  type="biotype", bio_perfect=FALSE, mismatches = 0)

## Find sequences that has been classified as other
other_seqs <- anno[grepl("other", anno$mis0),]$seq
tab <- full(reanno_biotype)$mis0$strna
tab[tab$seq %in% other_seqs,]          #No other hit in trna

tab <- full(reanno_biotype)$mis0$rrna
tab[tab$seq %in% other_seqs,]          #A few in rrna

## Add a search terms that catches the other rrna
bio_search <- list(
  rrna=c("5S", "5.8S", "12S", "16S",
         "18S", "28S", "pre_S45", "Other"),
  trna =c("_tRNA", "mt:tRNA"))

anno <- add_reanno(reanno_biotype, bio_search=bio_search,
                  type="biotype", bio_perfect=FALSE, mismatches = 0)

table(anno$mis0)

## Repeat search until no "Other" appear when running add_reanno,
## then run bio_perfect=TRUE:

anno <- add_reanno(reanno_biotype, bio_search=bio_search,
                  type="biotype", bio_perfect=TRUE, mismatches = 0)

```

anno

anno(PAC) Access the annotation table in S4 PAC

Description

anno(PAC)

Access the annotation table in S4 PAC

Usage

anno(object)

```
## S4 method for signature 'PAC'
```

```
anno(object)
```

```
## S4 replacement method for signature 'PAC,data.frame'
```

```
anno(object) <- value
```

Arguments

object A S4 PAC-object containing Pheno, Anno and Counts tables. Can be generated by the 'PAC generation' family of functions, for example see ?make_counts.

value Data.frame having the same row names as row names in Counts (sequences).

Value

The Anno data.frame table from a PAC-object.

Updated S4 PAC-object.

See Also

Other PAC methods: [counts\(\)](#), [names,PAC-method](#), [norm\(\)](#), [pheno\(\)](#), [show.PAC\(\)](#), [summary\(\)](#)

Examples

```
load(system.file("extdata", "drosophila_sRNA_pac_filt_anno.Rdata",
  package = "seqpac", mustWork = TRUE))
```

```
# extra fuctionality with s4 PAC-object:
```

```
names(pac)
```

```
length(pac)
```

```
nrow(pac)
```

```
ncol(pac)
```

```
rownames(pac)
```

```
colnames(pac)
```

```
pheno(pac)
```

```
head(anno(pac))
```

```
head(counts(pac))
```

```
head(norm(pac)$cpm)
```

as.PAC

Converts an S3 PAC into a S4 PAC

Description

as.PAC Converts an S3 PAC object (list) to an S4 PAC object

Usage

```
as.PAC(from)
```

Arguments

from S3 PAC-object containing at least a Pheno table with samples as row names, Anno table with sequences as row names and a Count table with raw counts (columns=samples, rows=sequences).

Details

Seqpac comes with two versions of the PAC object, either S4 or S3. The S3 PAC is simply a list where each object can be received using the \$-sign. The S4 version is a newer type of R object with predefined slots, that is received using the @-sign. This function converts an S3 PAC object (list) to an S4 PAC object. You can also use the S4 coercion method "as" to turn an S4 PAC into an S3. See examples below.

Value

An S4 PAC object.

See Also

<https://github.com/OestLab/seqpac> for updates on the current package.

Other PAC analysis: `PAC_covplot()`, `PAC_deseq()`, `PAC_filter()`, `PAC_filtsep()`, `PAC_gtf()`, `PAC_jitter()`, `PAC_mapper()`, `PAC_nbias()`, `PAC_norm()`, `PAC_pca()`, `PAC_pie()`, `PAC_saturation()`, `PAC_sizedist()`, `PAC_stackbar()`, `PAC_summary()`, `PAC_trna()`, `filtsep_bin()`, `map_rangetype()`, `trna_class()`

Examples

```
load(system.file("extdata", "drosophila_sRNA_pac_filt_anno.Rdata",
  package = "seqpac", mustWork = TRUE))

# check type
class(pac)
isS4(pac)
```

```
# Turns S4 PAC object into a S3
pac_s3 <- as(pac, "list")

# Turns S3 PAC object into a S4
pac_s4 <- as.PAC(pac_s3)
```

as.reanno

Converts an S3 reanno into a S4 reanno

Description

as.reanno Converts an S3 reanno object (list) to an S4 reanno object

Usage

```
as.reanno(from)
```

Arguments

from S3 reanno object.

Details

Seqpac comes with two versions of the reanno object, either S4 or S3. The S3 is simply a list where each object can be received using the \$-sign. The S4 version is a newer type of R object with predefined slots, that is received using the @-sign. This function converts an S3 reanno object (list) to an S4 object. You can also use the S4 coercion method "as" to turn an S4 into an S3. See examples below.

Value

An S4 reanno object.

See Also

<https://github.com/OestLab/seqpac> for updates on the current package.

Other PAC reannotation: [PAC_annotate\(\)](#), [add_reanno\(\)](#), [import_reanno\(\)](#), [make_conv\(\)](#), [make_reanno\(\)](#), [map_reanno\(\)](#), [simplify_reanno\(\)](#)

Examples

```
##### Create an reanno object

## First load a PAC- object

load(system.file("extdata", "drosophila_sRNA_pac_filt_anno.Rdata",
                 package = "seqpac", mustWork = TRUE))

## Then specify paths to fasta references
# If you are having problem see the vignette small RNA guide for more info.

trna_file <- system.file("extdata/trna", "tRNA.fa",
                        package = "seqpac", mustWork = TRUE)
trna_dir <- dirname(trna_file)

if(!sum(stringr::str_count(list.files(trna_dir), ".ebwt")) == 6){
  Rbowtie::bowtie_build(trna_file,
                      outdir=trna_dir,
                      prefix="tRNA", force=TRUE)
}

ref_paths <- list(trna= trna_file)

## Add output path of your choice.
# Here we use the R temporary folder depending on platform
output <- paste0(tempdir(), "/seqpac/test")

## Make sure it is empty (otherwise you will be prompted for a question)
out_fls <- list.files(output, recursive=TRUE)
closeAllConnections()
suppressWarnings(file.remove(paste(output, out_fls, sep="/")))

## Then map your PAC-object against the fasta references
map_reanno(pac, input=ref_paths, output=output,
           type="internal", mismatches=0, import="biotype",
           threads=2, keep_temp=FALSE, override=TRUE)

## Then import and generate a reanno-object of the temporary bowtie-files
reanno <- make_reanno(output, PAC=pac, mis_fasta_check = TRUE)

## Now make some search terms against reference names to create shorter names
# Theses can be used to create factors in downstream analysis
# One search hit (regular expressions) gives one new short name
bio_search <- list(trna =c("_tRNA", "mt:tRNA"))

## You can merge directly with your PAC-object by adding your original
# PAC-object, that you used with map_reanno, to merge_pac option.
pac <- add_reanno(reanno, bio_search=bio_search,
                 type="biotype", bio_perfect=FALSE,
                 mismatches = 0, merge_pac=pac)
```

```
## Turn your S3 list to an S4 reanno-object
isS4(reanno)
names(reanno)
table(overview(reanno)$trna)

reanno_s3 <- as(reanno, "list")
isS4(reanno_s3)

# Turns S3 reanno object into a S4
reanno_s4 <- as.reanno(reanno_s3)
isS4(reanno_s4)
```

colnames,PAC-method	<i>sample names</i>
---------------------	---------------------

Description

sample names

Usage

```
## S4 method for signature 'PAC'
colnames(x)
```

Arguments

x A S4 object of class PAC.

Value

The sample names held by the Counts table (column names) and the Pheno table (row names) of a PAC-object.

Examples

```
load(system.file("extdata", "drosophila_sRNA_pac_filt_anno.Rdata",
                 package = "seqpac", mustWork = TRUE))

# extra fuctionality with s4 PAC-object:
names(pac)
length(pac)
nrow(pac)
ncol(pac)
rownames(pac)
colnames(pac)
pheno(pac)
head(anno(pac))
head(counts(pac))
```

```
head(norm(pac)$cpm)
```

counts	<i>counts(PAC)</i>
--------	--------------------

Description

Access the raw counts table in S4 PAC

Usage

```
counts(object)

## S4 method for signature 'PAC'
counts(object)

## S4 replacement method for signature 'PAC,data.frame'
counts(object) <- value
```

Arguments

object	A S4 PAC-object containing Pheno, Anno and Counts tables. Can be generated by the 'PAC generation' family of functions, for example see <code>?make_counts</code> .
value	Data.frame having the same column names as row names in Pheno (sample ID) and row names as row names in Anno (sequences).

Value

The Counts data frame table from a PAC-object.
Updated S4 PAC-object.

See Also

Other PAC methods: [anno\(\)](#), [names,PAC-method](#), [norm\(\)](#), [pheno\(\)](#), [show.PAC\(\)](#), [summary\(\)](#)

Examples

```
load(system.file("extdata", "drosophila_sRNA_pac_filt_anno.Rdata",
  package = "seqpac", mustWork = TRUE))

# extra fuctionality with s4 PAC-object:
names(pac)
length(pac)
nrow(pac)
ncol(pac)
rownames(pac)
colnames(pac)
```

```
pheno(pac)
head(anno(pac))
head(counts(pac))
head(norm(pac)$cpm)
```

filtsep_bin

Binary converter for PAC_filtsep

Description

`filtsep_bin` Converts PAC_filtsep data.frame output into a binary (hit-no-hit) data.frame

Usage

```
filtsep_bin(filtsep_out)
```

Arguments

`filtsep_out` PAC_filtsep output data.frame, where each column contains the names of sequences that passed the filter for a specific group specified by a `pheno_target` object.

Details

Given a PAC_filtsep output data.frame, where each column contains the sequences that passed the filter for a specific group specified in `pheno_target`, `filtsep_bin` converts this into a data.frame where sequences are reported as hit (=1) or no hit (=0). Such binary coded group occurrence can for example be used by `UpSetR::upset` to generate visualization of overlaps using UpSet plots.

Value

data.frame where each unique sequence (row names) in `filtsep_out` are reported as hit (=1) or no hit (=0) across samples or sample groups (column names).

See Also

<https://github.com/OestLab/seqpac> for updates on the current package.

Other PAC analysis: `PAC_covplot()`, `PAC_deseq()`, `PAC_filter()`, `PAC_filtsep()`, `PAC_gtf()`, `PAC_jitter()`, `PAC_mapper()`, `PAC_nbias()`, `PAC_norm()`, `PAC_pca()`, `PAC_pie()`, `PAC_saturation()`, `PAC_sizedist()`, `PAC_stackbar()`, `PAC_summary()`, `PAC_trna()`, `as.PAC()`, `map_rangetype()`, `trna_class()`

Examples

```
load(system.file("extdata", "drosophila_sRNA_pac_filt_anno.Rdata",
  package = "seqpac", mustWork = TRUE))

## Keep sequences with 5 counts (threshold) in 100% (coverage) of
## samples in a group:
# Use PAC_filtsep to find sequences
filtsep <- PAC_filtsep(pac, norm="counts", threshold=5,
  coverage=100, pheno_target= list("stage"))

# Filter by unique sequences passing filtsep
filtsep <- unique(do.call("c", as.list(filtsep)))
pac_filt <- PAC_filter(pac, subset_only = TRUE, anno_target= filtsep)

# Find overlap
olap <- reshape2::melt(filtsep,
  measure.vars = c("Stage1", "Stage3", "Stage5"),
  na.rm=TRUE)

## Upset plot using the UpSetR package
# (when output="binary" PAC_filtsep uses filtsep_bin for binary conversion
# Use PAC_filtsep with binary output
filtsep_bin <- PAC_filtsep(pac, norm="counts", threshold=5,
  coverage=100, pheno_target= list("stage"),
  output="binary")

# Plot Venn diagram or UpSetR
#
# plot(venneuler::venneuler(data.frame(olap[,2], olap[,1])))
#
# UpSetR::upset(filtsep_bin, sets = colnames(filtsep_bin),
#   mb.ratio = c(0.55, 0.45), order.by = "freq", keep.order=TRUE)
```

full

full(reanno)

Description

Access the multilevel list of imported data.frames in a S4 reanno object

Usage

```
full(object)

## S4 method for signature 'reanno'
full(object)

## S4 replacement method for signature 'reanno,list'
full(object) <- value
```

Arguments

object	A S4 reanno-object containing an Full_anno list.
value	List of tibble data.frames with the same property as the Full_anno list of tibble data.frames in the reanno object, e.g. having the same names in seq column as row names (sequences) in the original PAC-object used to generate the reanno object. See examples and ?make_reanno for details.

Value

All the results that were imported into the reanno-object returned as a list of tibble data.frames (class tbl_df, tbl, data.frame).

Updated S4 reanno-object.

See Also

Other reanno methods: [overview\(\)](#)

Examples

```
##### Create a reanno object

## First load a PAC- object

load(system.file("extdata", "drosophila_sRNA_pac_filt_anno.Rdata",
                 package = "seqpac", mustWork = TRUE))
anno(pac) <- anno(pac)[,1, drop = FALSE]

## Then specify paths to fasta references
# If you are having problem see the vignette small RNA guide for more info.

trna_path <- system.file("extdata/trna", "tRNA.fa",
                        package = "seqpac", mustWork = TRUE)
rrna_path <- system.file("extdata/rrna", "rRNA.fa",
                        package = "seqpac", mustWork = TRUE)

ref_paths <- list(trna= trna_path, rrna= rrna_path)

## Add output path of your choice.
# Here we use the R temporary folder depending on platform
if(grepl("windows", .Platform$OS.type)){
  output <- paste0(tempdir(), "\\seqpac\\test")
}else{
  output <- paste0(tempdir(), "/seqpac/test")}

## Make sure it is empty (otherwise you will be prompted for a question)
out_fls <- list.files(output, recursive=TRUE)
closeAllConnections()
suppressWarnings(file.remove(paste(output, out_fls, sep="/")))
```

```
## Then map your PAC-object against the fasta references
map_reanno(pac, input=ref_paths, output=output,
            type="internal", mismatches=2, import="biotype",
            threads=2, keep_temp=FALSE)

## Then generate a reanno-object of the temporary bowtie-files
reanno_object <- make_reanno(output, PAC=pac, mis_fasta_check = TRUE)

## Accessing content and S4/S3 conversion:
names(reanno_object)
overview(reanno_object)
full(reanno_object)
rownames(reanno_object)
length(reanno_object)
nrow(reanno_object)
reanno_s3 <- as(reanno_object, "list")
reanno_s4 <- as.reanno(reanno_s3)
```

import_reanno	<i>Imports annotation from reannotation mapping</i>
---------------	---

Description

This function imports bowtie output and summarizes the content.

Usage

```
import_reanno(
  bowtie_path,
  threads = 1,
  coord = FALSE,
  report = "minimum",
  reduce = NULL
)
```

Arguments

bowtie_path	Path to a directory where bowtie output files can be found.
threads	Integer stating the number of parallel jobs.
coord	Logical whether or not mapping coordinates should be reported when report="full".
report	Character vector indicating what to report "minimum" or "full" (default="minimum").
reduce	Character indicating a reference name (without file type extension) that should be exempted from report="full" and instead will generate a minimum report.

Details

Given the path to alignment outputs from bowtie, `import_reanno` will attempt to read these files into R and generate a list of unsorted data.frames where each row summarizes all annotations for a given sequence. It is called by `map_reanno` to generate summarized data from bowtie output saved as .Rdata files. These files can then be re-imported into R where information is organized using `make_reanno` and finally added to the annotation table (PAC\$Anno) of the original PAC-list object using `add_reanno`.

Value

List of data frames with additional information from reannotation files generated with bowtie. If `report="minimum"`, the function will report each hit only as the number of mismatches for a given reference file. If `report="full"` the full name reported in the fasta file used as reference in the bowtie reannotation will be reported. If a reference name is specified in `reduce`, this reference is excerpted from `report="full"` and is instead reported as `report="minimum"`.

Caution: Large references with lots of redundancy (such as piRBase in some species) will cause massive character strings if `report="full"` with no restrictions. Specifying such references in `reduce=<reference_names>` will circumvent this problem.

Important

Re-annotation must have been done using Bowtie default output settings, where output files specifically have been named '.txt'. The function will not work with other formats such as SAM/BAM formats.

The basenames of the bowtie output will also be used to annotate. Example: bowtie -v 3 -a -f '<piRBase.fasta>' '<master_anno.fasta>' piRNA.txt Will annotate sequences as piRNA since the txt-file output was named 'piRNA'

See Also

<http://bowtie-bio.sourceforge.net/index.shtml> for information about Bowtie and for Rbowtie: <https://www.bioconductor.org/packages/release/bioc/html/Rbowtie.html>. <https://github.com/OestLab/seqpac> for updates on the current package.

Other PAC reannotation: `PAC_annotate()`, `add_reanno()`, `as_reanno()`, `make_conv()`, `make_reanno()`, `map_reanno()`, `simplify_reanno()`

Examples

```
#####
##### Test import_reanno

### First, if you haven't already generated Bowtie indexes for the included
# fasta reference you need to do so. If you are having problem see the small
# RNA guide (vignette) for more info.

## tRNA reference:
trna_file <- system.file("extdata/trna", "tRNA.fa",
                        package = "seqpac", mustWork = TRUE)
trna_dir<- gsub("tRNA.fa", "", trna_file)
```

```

if(!sum(stringr::str_count(list.files(trna_dir), ".ebwt")) ==6){
  Rbowtie::bowtie_build(trna_file,
                        outdir=trna_dir,
                        prefix="tRNA", force=TRUE)
}
## Then load a PAC-object and remove previous mapping from anno:
load(system.file("extdata", "drosophila_sRNA_pac_filt_anno.Rdata",
                 package = "seqpac", mustWork = TRUE))

ref_paths <- list(trna= trna_file)

## You may add an output path of your choice, but here we use a temp folder:
output <- paste0(tempdir(),"/seqpac/test")

## Then map the PAC-object against the fasta references. Warning: if you use
# your own data, you may want to use override=FALSE, to avoid deleting previous
# mapping by mistake. keep_temp=TRUE can be used to run import_reanno
# independently.

map_reanno(pac, input=ref_paths, output=output,
           type="internal", mismatches=2, import="biotype",
           threads=2, keep_temp=TRUE, override=TRUE)

reanno1 <- import_reanno(output, report="Biotype", threads=1)

```

length,PAC-method	<i>number of objects in PAC</i>
-------------------	---------------------------------

Description

number of objects in PAC
length of reanno object

Usage

```

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

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

```

Arguments

x A S4 object of class PAC.

Value

The number of items in a PAC-object.

Number of items in the reanno-object.

Examples

```
load(system.file("extdata", "drosophila_sRNA_pac_filt_anno.Rdata",
  package = "seqpac", mustWork = TRUE))
```

```
# extra functionality with s4 PAC-object:
```

```
names(pac)
length(pac)
nrow(pac)
ncol(pac)
rownames(pac)
colnames(pac)
pheno(pac)
head(anno(pac))
head(counts(pac))
head(norm(pac)$cpm)
```

make_conv

Create reference genome conversion table

Description

Generates a table that can be used for converting seqnames across referens genomes.

Usage

```
make_conv(
  reference_list = NULL,
  ref_path_A = NULL,
  ref_path_B = NULL,
  ref_path_C = NULL,
  output = "tibble",
  skip_after = NULL
)
```

Arguments

reference_list List containing 2-3 file paths (character strings) to references in fasta format. The names of the list items will be imported into the final conversion table. The function may also take each reference path as single character strings (ref_path_A, ref_path_B, ref_path_C), but then any user-defined reference name will be lost (see example below).

ref_path_A Character string with path to 1st fasta reference (leading)

ref_path_B	Character string with path to 2nd fasta reference
ref_path_C	Character string with path to 3rd fasta reference
output	Character indicating what type of output to provide. As default output="tibble" will result in a table in the tibble format as described in tibble/tidyverse packages. Anything else will result in a data frame.
skip_after	Character or named list with character strings. If a character string, only the part of the seqnames prior to that string will be returned. Thus, if skip_after=" ", seqnames will be trimmed after first white space. If a list of character strings, then each string will be applied to each provided reference. Thus, if skip_after=list(ensembl=" ", ucsc=" ", ncbi=";"), then seqnames will be trimmed at white space for the reference named "ensembl", and at ";" for the reference named "ucsc", etc. If skip_after=NULL then the full seqnames are returned.

Details

Given the file paths to 2-3 fasta reference genomes, this function will quickly match the sequences between these reference genomes and return seqnames of each matching sequence. Thus, given the paths to the fasta for the Ensembl, UCSC and NCBI assemblies of a given genome version (e.g. hg38 or dm6), will result in a name conversion table between these databases.

Only perfect matches or no matches will be reported. Thus, in case an entire sequence is missing (e.g. a sex chromosomes), a table will be returned with a warning, but if a sequence is partly missing (e.g. only half a chromosome), then an error is returned.

Sequence matching is done using md5 hashes, which dramatically increases the speed for perfect matches. Matching will always be done from reference A against the other references. Thus, if reference A (=1st reference in reference_list) contains less sequences than the other references, only the reference A sequences will be reported in the output, having the same order as in reference A.

Value

A name conversion table either as a tibble or a data frame (see output)

See Also

<https://github.com/OestLab/seqpac> for updates.

Other PAC reannotation: `PAC_annotate()`, `add_reanno()`, `as.reanno()`, `import_reanno()`, `make_reanno()`, `map_reanno()`, `simplify_reanno()`

Examples

```
## Only for testing:
fasta_path <- system.file("extdata/trna", "tRNA.fa",
                          package = "seqpac", mustWork = TRUE)
ref1 <- Biostrings::readDNAStringSet(fasta_path)
ref1 <- ref1[1:295]
sqnames <- do.call("rbind", (strsplit(names(ref1), "\\d chr")))[,2]
names(ref1) <- do.call("rbind", (strsplit(sqnames, " \\(")))[,1]
logi_dup <- duplicated(do.call("rbind", strsplit(names(ref1), "\\:"))[,1])
```

```

ref1 <- ref1[!logi_dup]
ref2 <- ref1
names(ref2) <- paste0("chr", names(ref2))
# Save new reference in temporary folder
if(grepl("windows", .Platform$OS.type)){
  tmpdr <- paste0(tempdir(), "\\seqpac")
}else{
  tmpdr <- paste0(tempdir(), "/seqpac")}
dir.create(tmpdr, showWarnings=FALSE)
ref_path1 <- paste0(tmpdr, "/ref1.fa")
ref_path2 <- paste0(tmpdr, "/ref2.fa")
Biostrings::writeXStringSet(ref1, filepath=ref_path1, format="fasta")
Biostrings::writeXStringSet(ref2, filepath=ref_path2, format="fasta")
ref_list <- list(ensembl=ref_path1, ucsc=ref_path2)
conv_table <- make_conv(reference_list=ref_list)
conv_table

## The principles:
#
#ref_path_A <- "/some/path/to/ensembl.fa"
#ref_path_B <- "/some/path/to/ucsc.fa"
#ref_path_C <- "/some/path/to/refseq.fa"
#
#ref_list <- list(ensembl=ref_path_A, UCSC=ref_path_B, NCBI=ref_path_C)
#
## Best (user defined names):
#conv_table <- make_conv(reference_list=ref_list)
#
## But also (no names)
#conv_table <- make_conv(ref_path_A, ref_path_B, ref_path_C)
#conv_table <- make_conv(ref_path_A, ref_path_C)
#
#
## Make short names (skip everything after white space)
#conv_table <- make_conv(reference_list=reference_list, skip_after=" ")

```

make_counts

Make a count table

Description

make_counts uses parallel processing to generate a count table.

Usage

```

make_counts(
  input,
  trimming = NULL,

```

```

threads = 1,
save_temp = FALSE,
plot = TRUE,
parse = "default_illumina",
evidence = c(experiment = 2, sample = 1),
optimize = list(on_disk = FALSE, chunk_size = NULL, rm_wild = FALSE, seq_min = NULL)
)

```

Arguments

input	A path to a directory containing input fastq-files. The script will recursively search this directory for the .fastq.fastq.gz extension.
trimming	Character indicating what type of trimming tool that should be used. If trimming="seqpac", fastq files will be sent to the make_trim function in seqpac, while if trimming="cutadapt" fastq files will be sent to the make_cutadapt function. Note that trimming="seqpac" runs independently of external software, while trimming="cutadapt" is dependent on an externally installed version of cutadapt and fastq_quality_filter. Trimmed fastq files are stored temporarily in the systems default temporary folder. Please, run make_trim and make_cutadapt separately for permanent storage options, or set save_temp=TRUE to avoid that make_counts will delete all temporary files. As default trimming=NULL, which indicates that input fastq files has already been trimmed.
threads	Integer stating the number of parallel jobs. Note, that reading multiple fastq files drains memory fast, using up to 10Gb per fastq file. To avoid crashing the system due to memory shortage, make sure that each thread on the machine have at least 10 Gb of memory available, unless your fastq files are very small. Use <code>parallel::detectcores()</code> to see available threads on the machine.
save_temp	Logical whether temporary files (including trimmed fastq files) should be saved or not. Note, the function will print the path to the temporary folder in the console.
plot	Logical whether evidence plots should be printed and saved in the returned list (default=TRUE).
parse	Character strings defining the command that should be parsed to make_trim or make_cutadapt . This will allow you to customize your trimming according to 3' adapter sequence and platform standards etc. Please see examples below and the manuals for make_trim and make_cutadapt for more details. For convenience, parse also have two default mode for sRNA trimming, using Illumina and New England Biotype (neb) type small RNA adapters. make_counts will automatically print the exact setting for each default mode. Briefly, both modes involves polyG (NextSeq/NovaSeq) trimming and 3' adapter trimming, with a 0.1 tolerance for mismatch/indels. If parse="default_illumina", then the "TG-GAATTCTCGGGTGCCAAGGAAGTCCAGTCAC" 3' adapter is trimmed and untrimmed sequences are removed. If parse="default_neb", then "AGATCG-GAAGAGCACACGTCTGAACTCCA" is trimmed and untrimmed sequences are removed. Removing untrimmed sequences is recommended for sRNA sequencing.

evidence	Character vector with two inputs named 'experiment' and 'sample' that controls the low-level evidence filter. Users may already at this point reduce the level of noise in the counts table by specifying the number of independent evidence that a specific sequence must have to be included. As default, <code>evidence=c(experiment=2, sample=1)</code> will include all sequences that have ≥ 1 count in at least 2 independent fastq files. Thus 'experiment' controls the number of independent fastq evidence needed across the whole experiment. Note, however, that 'sample' does not control the number of counts needed in each sample. The evidence filter will always use ≥ 1 count in X number of fastq files. Instead 'sample' controls when a sequence should be included despite not reaching the 'experiment' threshold. Thus if <code>evidence=c(experiment=2, sample=10)</code> , sequences that reach 10 counts in a single sample will also be included. If <code>evidence=NULL</code> all unique sequences will be saved. See 'examples' below for more examples.
optimize	List that controls the behavior of <code>make_counts</code>, possibly enabling analysis on low-end computers on the expense of processing time. Has four items named: <code>list(on_disk=, chunk_size=, rm_wild=, seq_min=)</code>. 1. <code>on_disk</code>: Logical whether or not some processes should be handled on-disk instead of in-memory. Important, this option will generate large temporary files. For a max fastq size of 1 GB (.gz compressed), keep at least 20 GB of free disk space. (default=FALSE). 2. <code>chunk_size</code>: Integer, if set, determines whether reading and processing should be handled in chunks and how big those should be. If NULL, samples are not handled in chunks. If handling a challenging dataset on low-end computers, a <code>chunk_size</code> of 50000000 is a good starting point. (default=NULL). 3. <code>rm_wild</code>. Logical (panic filter). If evidence filter fails while <code>on_disk=TRUE</code> then <code>rm_wild=TRUE</code> will remove all sequences containing wild-card characters (N), either across the experiment or for individual samples. (default=FALSE). 4. <code>seq_min</code>. Integer (panic filter). If evidence filter fails while <code>on_disk=TRUE</code> then <code>seq_min</code> will remove sequences shorter than <code>seq_min</code>, either across the experiment or for individual samples. (default=NULL) Note, Panic filters (2. and 3.) are applied at two places in the script as last resort if <code>on_disk</code> processing fails. First this may happen when unique sequences (UNI) fulfilling the evidence filter are extracted across the whole experiment. Here panic filters will be applied to all samples. The second place is when read sequences fulfilling the evidence filter are extracted from the trimmed fastq. Thus, the panic filter is applied to individual samples. To alert users, warnings are given in addition to notes in the <code>panic_type</code> column of the progress report. Here, "none" means no panic, "filt" means that panic filters were applied, and "chunk" means that loading of filtered fastq was forced into chunks. The last does not involve panic filters but helps loading very large fastqs on the expense of loading time.

Details

Given a path to fastq files this function performs low-level evidence filtering, generating a counts table of sequences passing the filter and plots summary statistics.

Value

A list containing three objects:

1. counts (data frame) = count table.
2. progress_report = progress report from trimming and evidence filter.
3. evidence_plots = bar graphs showing the impact of evidence filter.

See Also

<https://github.com/OestLab/seqpac> for updates on the current package.

Other PAC generation: `PAC_check()`, `PAC_create()`, `make_PAC()`, `make_cutadapt()`, `make_pheno()`, `make_trim()`, `merge_lanes()`

Examples

```
#####
### Seqpac fastq trimming with the make_counts function
### using default settings for NEBNext small RNA adaptor

# Seqpac includes strongly down-sampled smallRNA fastq.
sys_path = system.file("extdata", package = "seqpac", mustWork = TRUE)
input <- list.files(path = sys_path, pattern = "fastq", all.files = FALSE,
                    full.names = TRUE)

# Now we can run make_counts
# Notice that make_counts will generate another temp folder, that will
# be emptied on finalization. By setting save_temp=TRUE you may save the
# content. You may also use your own trimming settings by using the parse
# option. See ?make_trim for more details.

counts <- make_counts(input, threads=1, parse="default_neb",
                      trimming="seqpac", plot=TRUE,
                      evidence=c(experiment=2, sample=1))

head(counts$counts)
head(counts$progress_report)

# Notice that that there are fewer unique sequences than number of reads
# passing the filter. In normal, large, fastq we keep 95-99% of all reads
# while filtering out 30-40% of the sequences that only appear in 1 fastq
# Thus, the evidence filter may gain you performance in later analysis by
# avoiding nonsense sequences.

#####
### Lets change the evidence filter
###

# 2 evidence over two indepenent samples, saving single sample
# sequences reaching 3 counts:

test <- make_counts(input=input, trimming="seqpac",
                    parse="default_neb",
```

```

        evidence=c(experiment=2, sample=3))
extras <- apply(test$counts, 1, function(x){sum(!x==0)})
test$counts[extras==1,] # 6 single sample sequences reached 3 counts

# 2 evidence over two independent samples, saving single sample
# sequences reaching 2 counts

test <- make_counts(input=input, trimming="seqpac",
                    parse="default_neb",
                    evidence=c(experiment=2, sample=2))
extras <- apply(test$counts, 1, function(x){sum(!x==0)})
test$counts[extras==1,] # 120 single sample sequences reached 2 counts

```

make_cutadapt	<i>Trims/filter fastq using external cutadapt/fastq_quality_filter</i>
---------------	--

Description

make_cutadapt cutadapt/fastq_quality_filter

Usage

```
make_cutadapt(input, output, parse = NULL, threads = 1)
```

Arguments

input	Character path to a directory containing input fastq-files. The script will recursively search this directory for the .fastq.fastq.gz extension.
output	Character path to the output directory where trimmed fastq files will be stored and temporary files will be generated.
parse	List with two character string expressions. The first will be parsed to cutadapt while the other is be parsed to fastq_quality_filter. If any is NULL, then the function will not pass the command and the trimming or filtering will not be applied. Thus, if parse = list(cutadapt=NULL, fastq_quality_filter="-q 20 -p 80"), then only the quality filter will be applied.
threads	Integer stating the number of parallel jobs. Note, that reading multiple fastq files drains memory fast, using up to 10Gb per fastq file. To avoid crashing the system due to memory shortage, make sure that each thread on the machine have at least 10 Gb of memory available, unless your fastq files are very small. Use <code>parallel::detectcores()</code> to see available threads on the machine.

Details

Given a path to sequence files in fastq format this function will trim adaptor and remove sequences with low quality.

Value

Externally the function will generate trimmed and/or quality filtered fastq files in the output folder. Internally, a list of logs that can be used to generate a progress report is returned.

See Also

<https://cutadapt.readthedocs.io/en/stable/> for download and documentation on cutadapt. http://hannonlab.cshl.edu/fastx_toolkit/commandline.html for download and documentation on fastq_quality_filter. <https://github.com/OestLab/seqpac> for updates on seqpac.

Other PAC generation: [PAC_check\(\)](#), [PAC_create\(\)](#), [make_PAC\(\)](#), [make_counts\(\)](#), [make_pheno\(\)](#), [make_trim\(\)](#), [merge_lanes\(\)](#)

Examples

```
#####
### Principle of trimming using the make_cutadapt function
### (Important: Need external installations of cutadapt
### and fastq_quality_filter to work)
#
# input = "/some/path/to/input/folder"
# output = "/some/path/to/output/folder"
#
## Parse for make_cutadapt is a list of 2 character string expressions.
## The first is parsed to cutadapt and the other to fastq_quality_filter
## For parallel processes '-j 1' is recommended since seqpac will
## parallelize across samples and not within.
## Run system2("cutadapt -h", stdout=TRUE) and
## system("fastq_quality_filter -h", stdout=TRUE)
## for more options.
#
## String to parse to cutadapt:
# cut_prs <- paste0("-j 1 -a AGATCGGAAGAGCACACGTCTGAACTCCAGTCACAT",
#                  " --discard-untrimmed --nextseq-trim=20",
#                  " -O 10 -m 7 -M 70")
#
## Add string to parse to fastq_quality_filter:
# parse = list(
#   cutadapt=cut_prs,
#   fastq_quality_filter="-q 20 -p 80")
#
# logs <- make_cutadapt(input, output, threads=8, parse=parse)

#' # Clean up temp
closeAllConnections()
fls_temp <- list.files(tempdir(), recursive=TRUE, full.names = TRUE)
file.remove(fls_temp, showWarnings=FALSE)
```

make_PAC

*Make PAC***Description**

make_PAC Compiles, order and checks the assumptions of a PAC file.

Usage

```
make_PAC(counts, pheno = NULL, anno = NULL, output = "S4")
```

Arguments

counts	Data.frame representing a counts table, with column names as sample names and sequence names as row names. Contains the counts for each sequence across the samples. Can be generated by the make_counts .
pheno	Data.frame with sample names as row names containing metadata about each sample. Can be generated by the make_pheno function.
anno	Data.frame with unique sequences as row names containing metadata about each sequence. If anno=NULL, a simple Anno data.frame will automatically be generated from the the sequence names in counts. Annotations for each sequence can be extended using for example the reanno workflow (see vignette).
output	Character indicating output format. If type="S4" (default), then the PAC object is returned as an S4 object. If type="S3", the PAC object will be returned as a list of data.frames

Details

Given a pheno, an anno and a counts data.frames the functions will safely generate a simple PAC list-object.

Value

Ordered PAC list object, checked for compatibility with downstream analysis in seqpac.

See Also

<https://github.com/OestLab/seqpac> for updates on the current package.

Other PAC generation: [PAC_check\(\)](#), [PAC_create\(\)](#), [make_counts\(\)](#), [make_cutadapt\(\)](#), [make_pheno\(\)](#), [make_trim\(\)](#), [merge_lanes\(\)](#)

Examples

```

### First make counts

# Seqpac includes strongly down-sampled smallRNA fastq.
sys_path = system.file("extdata", package = "seqpac", mustWork = TRUE)
input <- list.files(path = sys_path, pattern = "fastq", all.files = FALSE,
                    full.names = TRUE)

# Notice that make_counts will generate another temp folder, that will
# be emptied on finalization. By setting save_temp=TRUE you may save the
# content.

counts <- make_counts(input, threads=2, parse="default_neb",
                      trimming="seqpac", plot=TRUE,
                      evidence=c(experiment=2, sample=1))

colnames(counts$counts)

### Then generate a phenotype table with make_pheno

# Note: 'Sample_ID' column needs to be similar IDs as
#       colnames in the counts table. You may also
#       specify a path to a txt file.

Sample_ID <- colnames(counts$counts)

pheno <- data.frame(Sample_ID=Sample_ID,
                    Treatment=c(rep("heat", times=1),
                                rep("control", times=2)),
                    Batch=rep(c("1", "2", "3"), times=1))

pheno <- make_pheno(pheno=pheno, progress_report=counts$progress_report,
                   counts=counts$counts)

pheno

# Note that progress report from make_counts is added if you specify it

### Lastly combine into PAC

pac <- make_PAC(pheno=pheno, counts=counts$counts)

pac
names(pac)

# Note: a simple annotation table is added automatically.
head(anno(pac))

# Clean up temp
closeAllConnections()

```

```
fls_temp <- list.files(tempdir(), recursive=TRUE, full.names = TRUE)
file.remove(fls_temp, showWarnings=FALSE)
```

make_pheno	<i>Make Phenotype File</i>
------------	----------------------------

Description

make_pheno generates a Pheno data.frame object containing additional metadata for unique samples.

Usage

```
make_pheno(pheno, counts = NULL, progress_report = NULL)
```

Arguments

pheno	Character vector with the path to a .csv file or a data.frame with a column named "Sample_ID".
counts	Data.frame object with the same column names as in Sample_ID column of the pheno.
progress_report	Data.frame object with progress report.

Details

Given the path to a directory with a single csv file, the function will read the file. Alternatively, a data.frame with sample information can be provided. If provided with a count table, (see [make_counts](#)), it will attempt to organize the row names according to the column names in the counts table.

Value

A Pheno data.frame compatible with [make_PAC](#)

See Also

<https://github.com/OestLab/seqpac> for updates on the current package.

Other PAC generation: [PAC_check\(\)](#), [PAC_create\(\)](#), [make_PAC\(\)](#), [make_counts\(\)](#), [make_cutadapt\(\)](#), [make_trim\(\)](#), [merge_lanes\(\)](#)

Examples

```

### First make counts

# Seqpac includes strongly down-sampled smallRNA fastq.
sys_path = system.file("extdata", package = "seqpac", mustWork = TRUE)
input <- list.files(path = sys_path, pattern = "fastq", all.files = FALSE,
                    full.names = TRUE)

# Notice that make_counts will generate another temp folder, that will
# be emptied on finalization. By setting save_temp=TRUE you may save the
# content.

counts <- make_counts(input, threads=2, parse="default_neb",
                      trimming="seqpac", plot=TRUE,
                      evidence=c(experiment=2, sample=1))

colnames(counts$counts)

### Then generate a phenotype table with make_pheno

# Note: 'Sample_ID' column needs to be similar IDs as
#       colnames in the counts table. You may also
#       specify a path to a txt file.

Sample_ID <- colnames(counts$counts)

pheno <- data.frame(Sample_ID=Sample_ID,
                    Treatment=c(rep("heat", times=1),
                                rep("control", times=2)),
                    Batch=rep(c("1", "2", "3"), times=1))

pheno <- make_pheno(pheno=pheno, progress_report=counts$progress_report,
                   counts=counts$counts)

pheno

# Note that progress report from make_counts is added if you specify it

### Lastly combine into PAC

pac <- make_PAC(pheno=pheno, counts=counts$counts)

pac
names(pac)

# Note: a simple annotation table is added automatically.
head(anno(pac))

# Clean up temp
closeAllConnections()

```

```
fls_temp <- list.files(tempdir(), recursive=TRUE, full.names = TRUE)
file.remove(fl_s_temp, showWarnings=FALSE)
```

make_reanno

Makes annotation from R imported reannotation mapping

Description

make_reanno makes a reannotation (reanno) object.

Usage

```
make_reanno(reanno_path, PAC, mis_fasta_check = FALSE, output = "S4")
```

Arguments

reanno_path	Path to a directory where reannotation .Rdata files can be found.
PAC	PAC-list object containing an Anno data.frame with sequences as row names.
mis_fasta_check	Logical TRUE/FALSE if checking against anno_misX.fa should be done. The anno_misX.fa is the fasta file generated by the map_reanno function after completing the last mismatch cycle. This file contains all PAC sequences that failed to receive an alignment in any of fasta references and after all mismatch cycles. This file should be available in the same folder as the output files generated in the map_reanno function (=reanno_path).
output	Character indicating output format. If type="S4" (default), then the reanno object is returned as an S4 object. If type="list", the reanno object will be returned as a list of tables (tibbles).

Details

Given the path to the R reannotation files (reanno_mis0/1/2/3.Rdata) generated by [map_reanno](#), this function will summarize the reannotation files into one output that matches the order of sequences in a PAC object.

Value

Contains two slots/objects: 1. Overview: A table with a summary of the mapping. 2. Full_anno: Lists of tables holding the full imports per mismatch cycle (mis0, mis1 etc) and reference (mi0-ref1, mis0-ref2, mis1-ref1, mis1-ref2 etc). All table rows are ordered according to the order of the PAC originally used for the mapping. If *mis_fasta_check* is specified, the function will look for a *anno_misX.fa* (X = the file with the highest number) previously generated by the reannotation workflow. This file is used to double check so that no sequences are missing before and after reannotation.

See Also

<http://bowtie-bio.sourceforge.net/index.shtml> for information about Bowtie <https://github.com/OestLab/seqpac> for updates on the current package.

Other PAC reannotation: `PAC_annotate()`, `add_reanno()`, `as.reanno()`, `import_reanno()`, `make_conv()`, `map_reanno()`, `simplify_reanno()`

Examples

```
#####
#### Simple example for reference mapping
#### Please see manual for ?simply_reanno an ?add_reanno for more advanced
#### mapping
closeAllConnections()
#####
#### Create an reanno object

### First, if you haven't already generated Bowtie indexes for the included
# fasta references you need to do so. If you are having problem see the small
# RNA guide (vignette) for more info.

## tRNA:
trna_file <- system.file("extdata/trna", "tRNA.fa",
                        package = "seqpac", mustWork = TRUE)
trna_dir <- dirname(trna_file)

if(!sum(stringr::str_count(list.files(trna_dir), ".ebwt")) == 6){
  Rbowtie::bowtie_build(trna_file,
                      outdir=trna_dir,
                      prefix="tRNA", force=TRUE)
}

## rRNA:
rrna_file <- system.file("extdata/rrna", "rRNA.fa",
                        package = "seqpac", mustWork = TRUE)
rrna_dir <- dirname(rrna_file)

if(!sum(stringr::str_count(list.files(rrna_dir), ".ebwt")) == 6){
  Rbowtie::bowtie_build(rrna_file,
                      outdir=rrna_dir,
                      prefix="rRNA", force=TRUE)
}

## Then load a PAC-object and remove previous mapping from anno:
load(system.file("extdata", "drosophila_sRNA_pac_filt_anno.Rdata",
                package = "seqpac", mustWork = TRUE))
anno(pac) <- anno(pac)[,1, drop = FALSE]

ref_paths <- list(trna= trna_file, rrna= rrna_file)

## You may add an output path of your choice, but here we use a temp folder:
```



```

    }

    ref_paths <- list(genome1=mycoplasma_file, genome2=mycoplasma_file)
    map_reanno(PAC=pac, input=ref_paths, output=output,
               type="internal", mismatches=0, import="genome",
               threads=2, keep_temp=TRUE, override=TRUE)
    reanno_genome <- make_reanno(output, PAC=pac, mis_fasta_check = TRUE)

    table(overview(reanno_genome)$Any) # Low Mycoplasma contamination

#####

```

make_trim

*Make trimmed and quality filtered fastq files***Description**

make_trim uses parallel processing to generate trimmed reads.

Usage

```

make_trim(
  input,
  output,
  indels = TRUE,
  concat = 12,
  check_mem = FALSE,
  threads = 1,
  chunk_size = NULL,
  polyG = c(type = NULL, min = NULL, mismatch = NULL),
  adapt_3_set = c(type = "hard_rm", min = 10, mismatch = 0.1),
  adapt_3 = "AGATCGGAAGAGCACACGTCTGAACTCCAGTCACTA",
  adapt_5_set = c(type = NULL, min = NULL, mismatch = NULL),
  adapt_5 = NULL,
  nucleotide_range = c(min = NULL, max = NULL),
  quality = c(threshold = 20, percent = 0.8)
)

```

Arguments

input	Character indicating the path to a directory containing fastq formatted sequence files either compressed (.fastq.gz) or text (.fastq).
output	Character indicating the path to a directory where the compressed trimmed fastq files should be saved (as basename trim.fastq.gz).
indels	Logical whether indels should be allowed in the mismatch between read and adaptor sequences. Note that indel trimming will only be done on fragments that have failed to be trimmed by other means. Default=TRUE.

concat	Integer setting the threshold for trimming concatemer-like adaptors. Even when adaptor synthesis and ligation are strictly controlled, concatemer-like ("di-") adaptor sequences are formed. When concat is an integer, make_trim will search all trimmed sequences for additional adaptor sequence using a shorter version of the provided adaptor in adapt_3. The length of the short adaptor is controlled by adapt_3_set\$min. If an additional, shorter, adaptor sequence is found in a read, trimming will only occur if the resulting sequence is $\leq$ concat shorter in length than the first trimmed sequence. Thus, if concat=12, a read with a NNNNN-XXXXX-YYYYYYYYYY composition, where N is 'real' nucleotides and X/Y are two independent adaptor sequences, the trimming will result in NNNNN-XXXXX. If instead concat=5 then trimming will result in NNNNN. Note that setting concat too low will result in trimming of real nucleotides that just happen to share sequence with the adaptor. As default concat=12, which have been carefully evaluated in relation to the risk of trimming real sequence. If concat=NULL, concatemer-like adaptor trimming will not be done.
check_mem	Logical whether a memory check should be performed. A memory check estimates the approximate memory usage given the fastq file sizes and number of threads. The check happens before entering the parallel trimming loop, and will give an immediate warning given intermediary risky memory estimates, and an error if the risk is very high. Note, this option depends on the parallel and benchmarkme packages. (default=FALSE)
threads	Integer stating the number of parallel jobs. Note, that reading multiple fastq files drains memory, using up to 10Gb per fastq file. To avoid crashing the system due to memory shortage, make sure that each thread on the machine have at least 10 Gb of memory available, unless your fastq files are very small or very large. Use parallel::detectcores() to see available threads on the machine. Never use all threads! To run large fastq on low end computer use 'chunk_size'.
chunk_size	Integer indicating the number of reads to be read into memory in each chunk. When chunk_size=NULL (default), the whole fastq will be processed as one. Thus, running 5 threads will keep 5 full fastq files in memory simultaneously (fast but demanding). Setting chunk_size to an integer will instead read only a portion of each fastq at a time and then append the trimmed results on disc (slow but less demanding). Try using chunk_size=1000000 reads, less will be extremely slow. Rather decrease threads if you encounter memory problems.
polyG	Character vector with three inputs named 'type', 'min' and 'mismatch' that controls poly G trimming. This trimming might be necessary for fastq files from two channel illumina sequencers (e.g. NextSeq and NovaSeq) where no color signals are interpreted as 'G', but where read failure sometimes also results in long stretches of 'no-signal-Gs'. Works similar to adapt_3_set but instead of an adaptor sequence a poly 'G' string is constructed from the 'min' input option. Thus, if polyG=c(type="hard_rm", min=10, mismatch=0.1) then sequences containing 'GGGGGGGGGNNNNN...etc' with 10 percent mismatch will be removed from the output fastq file.
adapt_3_set	Character vector with three inputs named 'type', 'min' and 'mismatch' that controls 3' trimming. As default, adapt_3_set=c(type="hard_rm", min=10, mismatch=0.1). Options for each input are:

	type= 'hard_trim': Trims all up to the very last nucleotide. 'hard_rm': Same as 'hard_trim' but removes untrimmed sequences. 'hard_save': Same as 'hard_trim' but saves all untrimmed sequences in new fastq file. 'soft_trim': No trimming on the last 3' nucleotides equal to 1/2 of 'min'. 'soft_rm': Same as 'soft_trim' but removes untrimmed sequences. 'soft_save': Same as 'soft_trim' but saves all untrimmed sequences in new fastq file. min= Integer that controls the short adaptor/poly-G/concatamer trimming and soft trimming. When min=10, a short version of the adaptor is generated containing the 10 first nucleotides. The short version of the adaptor therefore controls the minimum number of overlaps between adaptor/poly-G. Short adaptor trimming will only occur on untrimmed reads that have failed trimming with full adaptor. The min option also controls how many terminal nucleotides that will escape trimming when type is set to 'soft' (1/2 of min). mismatch= Numeric controlling the percent mismatch. For instance, if min=10 and mismatch=0.1, then 1 mismatch is allowed in the minimum overlap.
adapt_3	Character string defining the 3' adaptor sequence. In most library preparation protocols, the 3' adaptor is the only adaptor needed trimming.
adapt_5_set	Currently not supported, but will be in future updates. Same as adapt_3_set but controls the behavior of 5' trimming when a sequence is provided in adapt_5.
adapt_5	Character string defining a 5' adaptor sequence. 5' adaptor sequences are only trimmed in specialized protocols or in very short paired-end sequencing.
nucleotide_range	Numeric vector with two inputs named 'min' and 'max', that controls the sequence size filter. For example, if nucleotide_range=c(min=15, max=50) the function will extract sequences in the range between 15-50 nucleotides after trimming. As default, nucleotide_range=c(min=NULL, max=NULL) and will retain all trimmed sequences.
quality	Numeric vector with two inputs named 'threshold' and 'percent' that controls the fastq quality filter. For example, if quality=c(threshold=20, percent=0.8) (default) the function will extract sequences that pass a fastq phred score >=20 in 80 percent of the nucleotides after trimming. If quality=c(threshold=NULL, percent=NULL) then all sequences will be retained (not default!).

Details

Given a file path to sequence files in fastq format (extension .fastq.gz or .fastq) this function will first trim the sequences from adaptor sequence, and then apply sequence size and phred score quality filters. On systems with low memory or very large fastq files, users may choose to handle each file in chunks on the expense of time performance.

Value

Exports a compressed trimmed fastq file to the output folder (file name: basename+trim.fastq.gz). If any type="save" has been set, an additional fastq file (file name includes 'REMOVED') will be exported. In addition, overview statistics (progress report) will be returned as a data.frame.

See Also

<https://github.com/OestLab/seqpac> for updates on the current package.

Other PAC generation: `PAC_check()`, `PAC_create()`, `make_PAC()`, `make_counts()`, `make_cutadapt()`, `make_pheno()`, `merge_lanes()`

Examples

```
#####
### Seqpac fastq trimming with the make_trim function
### (for more streamline options see the make_counts function)
#####
### Seqpac fastq trimming with the make_counts function
### using default settings for NEBNext small RNA adaptor

# Seqpac includes strongly down-sampled smallRNA fastq.
sys_path = system.file("extdata", package = "seqpac", mustWork = TRUE)
input <- list.files(path = sys_path, pattern = "fastq", all.files = FALSE,
                    full.names = TRUE)

# Run make_trim using NEB-next adaptor
# (Here we store the trimmed files in the input folder)
# (but you may choose where to store them using the output option)

input <- unique(dirname(input)) # mak_trim searches for fastq in input dir
output <- tempdir()             # save in tempdir

list.files(input, pattern = "fastq") #before

prog_report <- make_trim(
  input=input, output=output, threads=1,
  adapt_3_set=c(type="hard_rm", min=10, mismatch=0.1),
  adapt_3="AGATCGGAAGAGCACACGTCTGAACTCCAGTCACTA",
  polyG=c(type="hard_trim", min=20, mismatch=0.1),
  nucleotide_range=c(min=14, max=70),
  quality=c(threshold=20, percent=0.8))

list.files(path = output, pattern = "fastq") #after

# How did it go? Check progress report:
prog_report

## The principle:
# input <- "/some/path/to/untrimmed_fastq_folder"
# output <- "/some/path/to/output_folder_for_trimmed_fastq"
#
# prog_report <- make_trim(
#   input=input, output=output,
#   threads=<how_many_parallel>,
#   check_mem=<should_memory_be_checked?>,
#   adapt_3_set=<options_for_main_trimming,
```

```
# adapt_3=<sequence_to be trimmed,
# polyG=<Illumina_Nextseq_type_poly_G_trimming>,
# nucleotide_range=<what_sequence_range_to_save>,
# quality=<quality_filtering_options>))
```

map_rangetype	<i>Interval classification.</i>
---------------	---------------------------------

Description

map_rangetype Classifies sequences based on interval mapping against a reference.

Usage

```
map_rangetype(
  map,
  type = "percent",
  ss = NULL,
  min_loop_width = 4,
  intervals = list(start = 1:5, mid = 45:55, end = 95:100),
  N_include = FALSE
)
```

Arguments

map	PAC_map (generated by PAC_mapper) or a Reanno object (generated by map_reanno), containing mapping coordinates against a reference fasta file.
type	Character indicating what type of intervals that is provided. If type="nucleotides", then the interval list is given as ranges of nucleotide positions. For example, if interval=list(start=1:3, end=1:3) the function will classify sequences starting within the first three nucleotides of the reference as 'type_start_nuc' and sequences ending in within the last three nucleotides of the reference as 'type_end_nuc'. If type="percent", then intervals needs to be provided as percent ranges. For example, if intervals=list(start=1:5, mid=45:50, end=95:100) then the function will classify sequences starting within the 5 in the references as 'type_start_per', and sequences ending within the 5 last nucleotides of the references as "type_end_per". It will also, classify sequences starting within 45-50 "type_mid_start_per" and sequences ending within 45-50 as 'type_mid_end_per'. If type="ss", then intervals is obtained from an ss file, obtained for example from tRNAscan-SE (http://lowelab.ucsc.edu/tRNAscan-SE/) or at GtR-NAdb http://gtrnadb.ucsc.edu/. Importantly - the intervals list is name sensitive. If type="nuclotides", intervals can only contain two intervals named 'start' and 'end', while if type="percent" then intervals needs to contain three intervals named 'start', 'mid' and 'end'. Hint, for classifying 5' and 3' half tsRNA you need to run the function twice. First, classify each sequence as 5'-start or 3'-end tsRNA using type="nucleotides",

	and then rerun the the map object using type="percent" specifying the 'mid' region as the half interval.
ss	File path to ss file (character), readLines vector of ss file (character) or ss list. If character, the function will attempt to read a file from the path given in the character string. If this fails, the function assumes that the ss file has already been read using readLines, and will attempt to split that character vector into a list of unique sequences by splitting at the empty lines. Empty line normally delimits each sequence entry in the ss file. Such a list can also be parsed directly to the function, making it easy to change for example sequence names using lapply prior to running the function.
min_loop_width	Integer setting the minimum number of nucleotides for a loop. Only applicable when type="ss". Loops in ss-files are defined by ">" followed by x number of "." ending with "<". For example: ATCGGTGGTTTCAGTGGTAGAATGCTCGCCTCGCGGGCGGCCGGGTTTCGATTCCCGGCCGATG >»»...>»>.....<«<.>»»...<««....>»».....<««««« Here are three possible loops: "AGTGGTA", "CTC", "TTCGATT". If min_loop_width=3, the middle loop (">...<="CTC") will be classified as a loop. If min_loop_width=4 (default), the middle loop will not be classified as a loop because it is too short.
intervals	A named list with integer intervals.
N_include	Logical whether or not N "wild card" nucleotides should be counted in the terminals. This conveniently controls the N_up and N_down arguments in the PAC_mapper function. If N_include=FALSE (default), start and end of tRNA will be measured from the first and last canonical nucleotides (A, T, C, G). Thus, if fragments align to an NNN-terminal, it will receive a negative value. If N_include=TRUE, N wild-cards will be treated as any other nucleotide.

Details

Given a PAC_map object ([PAC_mapper](#)) and an interval list this function will attempt to classify mapped sequences based on where these sequences starts and ends in reference. This function can for example be used for 5' and 3' tRNA classification.

Value

Map list object containing reference sequence (Ref_seq) as Biostrings::DNASTringSet and the new classifications embedded with the alignments (Alignments) in a dataframe.

See Also

<https://github.com/OestLab/seqpac> for updates on the current package.

Other PAC analysis: [PAC_covplot\(\)](#), [PAC_deseq\(\)](#), [PAC_filter\(\)](#), [PAC_filtsep\(\)](#), [PAC_gtf\(\)](#), [PAC_jitter\(\)](#), [PAC_mapper\(\)](#), [PAC_nbias\(\)](#), [PAC_norm\(\)](#), [PAC_pca\(\)](#), [PAC_pie\(\)](#), [PAC_saturation\(\)](#), [PAC_sizedist\(\)](#), [PAC_stackbar\(\)](#), [PAC_summary\(\)](#), [PAC_trna\(\)](#), [as.PAC\(\)](#), [filtsep_bin\(\)](#), [trna_class\(\)](#)

Examples

```
#####
### test the map_rangetype function
# More complicated examples can be found in the vignette.
##-----

# First create an annotation blank PAC with group means
load(system.file("extdata", "drosophila_sRNA_pac_filt_anno.Rdata",
                 package = "seqpac", mustWork = TRUE))
anno(pac) <- anno(pac)[,1, drop=FALSE]
pac_trna <- PAC_summary(pac, norm = "cpm", type = "means",
                       pheno_target=list("stage"), merge_pac = TRUE)

# Then re-annotate only tRNA using the PAC_mapper function
ref <- system.file("extdata/trna", "tRNA.fa",
                   package = "seqpac", mustWork = TRUE)
map_object <- PAC_mapper(pac_trna, input=ref, N_up = "NNN", N_down = "NNN",
                        mismatches=0, threads=2, report_string=TRUE,
                        override=TRUE)

## Coverage plot of tRNA using PAC_covplot

# Single tRNA targeting a summary table
PAC_covplot(pac_trna, map=map_object,
            summary_target= list("cpmMeans_stage"),
            map_target="tRNA-Ala-AGC-1-1")

## Classify range types with map_rangetype (see vignette for examples
# on how to use ss-files for detailed tRNA loop structure).

# Classify fragments using percent intervals
map_object <- map_rangetype(map_object,
                           intervals = list(start = 1:5, mid = 45:55, end = 95:100))

names(map_object)
map_object[[1]]
```

Description

Aligns sequences in a PAC list object against specified references generating summarized report files in the destination folder.

Usage

```
map_reanno(
```

```

PAC,
type = "internal",
output,
input,
mismatches = 3,
threads = 1,
parse_external = "-a -f",
parse_internal = "a=TRUE, f=TRUE",
import = "genome",
keep_temp = FALSE,
override = FALSE
)

```

Arguments

PAC	PAC-list object containing an Anno data.frame with sequences as row names.
type	Character indicating if mapping should be performed by calling the internal bowtie function of the Rbowtie package (type="internal"), or if bowtie should be called externally through a system call for a locally installed bowtie (type="external").
output	Character indicating the path to the destination folder for the files generated by the function.
input	List of file paths (character) indicating the full paths to each fasta reference file. It is important to carefully name each reference path. The names will appear as they are named in the final annotation table. Thus, if <i>input=list(tRNA="<path_tRNA_ref>", miRNA="<path_miRNA_ref>")</i> mapping to these fasta references will appear as "tRNA" and "miRNA", respectively. Note: All reference fasta files must have bowtie indexes using Rbowtie::bowtie_build.
mismatches	Integer indicating the maximum number of mismatches allowed in the alignments. The function currently supports no more than 3 mismatches (botwie max).
threads	Integer indicating the number of parallel processes to be used.
parse_external	One character string specifying the additional options parsed to externally installed bowtie when type="external". The string follows similar rules as '...' parsing. More information on the formatting use: <code>system("bowtie --manual", intern=TRUE, ignore.stderr=FALSE)</code> . If this command fails you probably do not have bowtie correctly installed. As default parse_external= "-a -f".
parse_internal	One character string specifying the additional options parsed to the bowtie function in the Rbowtie package when type="internal". The string follows similar rules as '...' parsing. See Rbowtie::bowtie for details on the format. As default: parse_internal= "a=TRUE, f=TRUE".
import	Character or a list. If import="genome" mapping is done against a reference genome and genomic coordinates are acquired. If import="biotype", mapping is done against a specialized fasta reference (e.g. Ensembl_ncrna, pirBase etc), where genomic coordinates is not required because classification will be performed on a match-or-no-match basis. A list of exactly 3 objects, named "co-ord", "report" and "reduce" can also be provided. This list will be parsed to the

`import_reanno` function. When `import="genome"`, the list `import=list(coord=TRUE, report="full", reduce=NULL)` is automatically parsed, while when `import="biotype"` the list parsed is `import=list(coord=FALSE, report="full", reduce=NULL)`. Performance increases by setting `coord=FALSE`. See `import_reanno` for more information on how to set `report` and `reduce` for increased performance when extremely large and repetitive references are used, such as `pirBase` and `repeatMasker`.

<code>keep_temp</code>	Logical whether or not bowtie output files temporarily stored in the output path should be deleted. Note, this option is only used for troubleshooting. The bowtie output files are named as the reference files and are overwritten in each mismatch cycle. Thus, for safe saving of mismatch 0 bowtie output make sure that <code>mismatches=0</code> . If not, the mismatch 1 cycle will overwrite the bowtie files.
<code>override</code>	Logical whether or not the function should prompt you for a question if there are files in output. As default, <code>override=FALSE</code> will prevent deleting large files by accident, but requires an interactive R session. Setting <code>override=TRUE</code> may solve non-interactive problems.

Details

Given a PAC object this function will extract the read sequences and align them against single or multiple fasta reference file(s). Summarized output will be saved as `.Rdata` files in the destination folder. In its default setting the function will only report hit or no hit in up to 3 mismatches, but this can easily be changed between 0-3 mismatches, to include alignment coordinates and reference name extractions. Information from the `.Rdata` files (`Full_reanno_mis0/1/2/3.Rdata`) can be extracted and added to a PAC object using `make_reanno` and `add_reanno` functions. For increased compatibility across platforms, the function provides both R internal Bowtie parsing (through the `Rbowtie` package), as well as external parsing to a locally installed version of Bowtie.

Value

Will primarily generate `.Rdata` files in the destination folder (output) containing summarized information about the reference alignments. One file is generated for every mismatch specified in `mismatches`. The `make_reanno` function can then be used to extract and generate annotation tables for a PAC list object. Large temporary bowtie input and output files will also be generated in the destination folder, but are removed unless `temp_remove=FALSE`.

See Also

<http://bowtie-bio.sourceforge.net/index.shtml> for information about Bowtie and for `Rbowtie`:
<https://www.bioconductor.org/packages/release/bioc/html/Rbowtie.html>. <https://github.com/OestLab/seqpac> for updates on the current package.

Other PAC reannotation: `PAC_annotate()`, `add_reanno()`, `as_reanno()`, `import_reanno()`, `make_conv()`, `make_reanno()`, `simplify_reanno()`

Examples

```
#####
##### Simple example for reference mapping
##### Please see manual for ?simplify_reanno and ?add_reanno for more advanced
```

```
##### mapping
closeAllConnections()
#####
##### Create an reanno object

### First, if you haven't already generated Bowtie indexes for the included
# fasta references you need to do so. If you are having problem see the small
# RNA guide (vignette) for more info.

## tRNA:
trna_file <- system.file("extdata/trna", "tRNA.fa",
                        package = "seqpac", mustWork = TRUE)
trna_dir <- dirname(trna_file)

if(!sum(stringr::str_count(list.files(trna_dir), ".ebwt")) == 6){
  Rbowtie::bowtie_build(trna_file,
                      outdir=trna_dir,
                      prefix="tRNA", force=TRUE)
}

## rRNA:
rrna_file <- system.file("extdata/rrna", "rRNA.fa",
                        package = "seqpac", mustWork = TRUE)
rrna_dir <- dirname(rrna_file)

if(!sum(stringr::str_count(list.files(rrna_dir), ".ebwt")) == 6){
  Rbowtie::bowtie_build(rrna_file,
                      outdir=rrna_dir,
                      prefix="rRNA", force=TRUE)
}

## Then load a PAC-object and remove previous mapping from anno:
load(system.file("extdata", "drosophila_sRNA_pac_filt_anno.Rdata",
                package = "seqpac", mustWork = TRUE))
anno(pac) <- anno(pac)[,1, drop = FALSE]

input <- list(trna= trna_file, rrna= rrna_file)

## You may add an output path of your choice, but here we use a temp folder:
output <- paste0(tempdir(), "/seqpac/test")

## Then map the PAC-object against the fasta references.
# Warning: if you use your own data, you may want to use override=FALSE, to avoid
# deleting previous mapping by mistake.

map_reanno(pac, input=input, output=output,
           type="internal", mismatches=0, import="biotype",
           threads=2, keep_temp=FALSE, override=TRUE)

## Then import and generate a reanno-object of the temporary bowtie-files
reanno_biotype <- make_reanno(output, PAC=pac, mis_fasta_check = TRUE)
```

```

## Now make some search terms against reference names to create shorter names
# These can be used to create factors in downstream analysis
# One search hit (regular expressions) gives one new short name

bio_search <- list(
  rrna=c("5S", "5.8S", "12S", "16S", "18S", "28S", "pre_S45"),
  trna=c("_tRNA", "mt:tRNA"))

## You can merge directly with your PAC-object by adding your original
# PAC-object, that you used with map_reanno, to merge_pac option.

pac <- add_reanno(reanno_biotype, bio_search=bio_search,
  type="biotype", bio_perfect=FALSE,
  mismatches = 0, merge_pac=pac)

table(anno(pac)$mis0_bio)

#####
## Similarly, you can use Bowtie indexed genome fasta references
## But don't forget to use import="genome" for coordinate import
#
# input <- list(genome="<path_to_bowtie_indexed_fasta>")
# output_genome <- "<your_path_to_output_folder>"

## We can actually map against several genomes simultaneously:
# (to illustrate the principle we run the mycoplasma genome twice)

mycoplasma_file <- system.file("extdata/mycoplasma_genome",
  "mycoplasma.fa",
  package = "seqpac", mustWork = TRUE)
mycoplasma_dir<- gsub("mycoplasma.fa", "", mycoplasma_file)

## Don't forget to create bowtie indexes
if(!sum(stringr::str_count(list.files(mycoplasma_dir), ".ebwt")) ==6){
  Rbowtie::bowtie_build(mycoplasma_file,
    outdir=mycoplasma_dir,
    prefix="mycoplasma", force=TRUE)
}

input <- list(genome1=mycoplasma_file, genome2=mycoplasma_file)
map_reanno(PAC=pac, input=input, output=output,
  type="internal", mismatches=0, import="genome",
  threads=2, keep_temp=TRUE, override=TRUE)
reanno_genome <- make_reanno(output, PAC=pac, mis_fasta_check = TRUE)

table(overview(reanno_genome)$Any) # Low Mycoplasma contamination

#####

```

merge_lanes

*Merge fastq lane files***Description**

merge_lanes Identifies, reads and merges fastq lane files.

Usage

```
merge_lanes(input, output, threads = 1, nlanes = NULL)
```

Arguments

input	Character string with the path to lane files that should be merged. The names of the lane files should follow the convention: 'sample1_lane1.fastq.gz, sample1_lane2.fastq.gz, sample2_lane1.fastq.gz, etc', where the first part of the name indicate the sample while the second part indicate lanes. The function will automatically try to identify sample names based on that the second part is "_lane" or "_L00". The function only takes fastq.gz compressed files.
output	Character string with the path to destination folder for merged fastq.gz files.
threads	Integer indicating the number of parallel processes that should be used.
nlanes	Integer indicating the number of lanes that should be merged. This works as a safety measure to ensure correct lane merging if names are long or complicated. Default=NULL.

Details

Given an input path were flow cell lane files in fastq format (typically generated by for example Illumina sequencers) this function will try to identify unique samples among the lane files and merge all lane files for each sample. The function uses md5 checks to control that each lane file is unique (no accidental copies) and will compare expected outcomes given that all samples should have the same number of lanes (coming from the same experiment). If the automatic regex for commonly used lane naming does not work, try defining number of lanes to be merged by @param nlanes.

Value

Merged fastq files in destination folder.

See Also

<https://github.com/OestLab/seqpac> for updates on the current package.

Other PAC generation: [PAC_check\(\)](#), [PAC_create\(\)](#), [make_PAC\(\)](#), [make_counts\(\)](#), [make_cutadapt\(\)](#), [make_pheno\(\)](#), [make_trim\(\)](#)

Examples

```
## The simple principle:
# input <- "/some/path/to/lane/files/fastq.gz"
# output <- "/some/path/to/merged/files/"
# merge_lanes(input, output, threads=12)

## Real example
# First generate some correct file names (see above).
sys_path = system.file("extdata", package = "seqpac", mustWork = TRUE)
fq <- list.files(path = sys_path, pattern = "fastq", all.files = FALSE,
                 full.names = TRUE)

# Create an output folder
input <- paste0(tempdir(), "/lanes/")
output <- paste0(tempdir(), "/merged/")
dir.create(input, showWarnings=FALSE)
dir.create(output, showWarnings=FALSE)
# Fix compatible file names
file.copy(from = fq, to = input)
old_fls <- list.files(input, full.names=TRUE)
new_sample <- c(rep("sample1_", times=3), rep("sample2_", times=3))
new_lane <- rep(c("lane1", "lane2", "lane3"), times=2)
new_fls <- paste0(input, new_sample, new_lane, ".fastq.gz")
file.rename(from = old_fls, to = new_fls)

# Then merge the fastq files
merge_lanes(input, output, threads=2)

# You will find the files in:
input
output

##-----##
## Warning: Clean up temp folder      ##
# (Sometimes needed for automated examples)

closeAllConnections()
fls_temp <- tempdir()
fls_temp <- list.files(fls_temp, recursive=TRUE, full.names = TRUE)
suppressWarnings(file.remove(fls_temp))
```

Description

names of objects in reanno

Usage

```
## S4 method for signature 'PAC'
names(x)

## S4 method for signature 'reanno'
names(x)
```

Arguments

x A S4 object of class PAC.

Value

The names of the content in a PAC-object.
Names of the items in the reanno-object.

See Also

Other PAC methods: [anno\(\)](#), [counts\(\)](#), [norm\(\)](#), [pheno\(\)](#), [show.PAC\(\)](#), [summary\(\)](#)

Examples

```
load(system.file("extdata", "drosophila_sRNA_pac_filt_anno.Rdata",
                 package = "seqpac", mustWork = TRUE))

# extra fuctionality with s4 PAC-object:
names(pac)
length(pac)
nrow(pac)
ncol(pac)
rownames(pac)
colnames(pac)
pheno(pac)
head(anno(pac))
head(counts(pac))
head(norm(pac)$cpm)

#####
##### Create a reanno object

## First load a PAC- object

load(system.file("extdata", "drosophila_sRNA_pac_filt_anno.Rdata",
                 package = "seqpac", mustWork = TRUE))
anno(pac) <- anno(pac)[,1, drop = FALSE]

## Then specify paths to fasta references
# If you are having problem see the vignette small RNA guide for more info.

trna_path <- system.file("extdata/trna", "tRNA.fa",
```

```

                                package = "seqpac", mustWork = TRUE)
rrna_path <- system.file("extdata/rrna", "rRNA.fa",
                                package = "seqpac", mustWork = TRUE)

ref_paths <- list(trna= trna_path, rrna= rrna_path)

## Add output path of your choice.
# Here we use the R temporary folder depending on platform
if(grepl("windows", .Platform$OS.type)){
  output <- paste0(tempdir(), "\\seqpac\\test")
}else{
  output <- paste0(tempdir(), "/seqpac/test")}

## Make sure it is empty (otherwise you will be prompted for a question)
out_fls <- list.files(output, recursive=TRUE)
closeAllConnections()
suppressWarnings(file.remove(paste(output, out_fls, sep="/")))

## Then map your PAC-object against the fasta references
map_reanno(pac, input=ref_paths, output=output,
            type="internal", mismatches=2, import="biotype",
            threads=2, keep_temp=FALSE)

## Then generate a reanno-object of the temporary bowtie-files
reanno_object <- make_reanno(output, PAC=pac, mis_fasta_check = TRUE)

## Accessing content and S4/S3 conversion:
names(reanno_object)
overview(reanno_object)
full(reanno_object)
rownames(reanno_object)
length(reanno_object)
nrow(reanno_object)
reanno_s3 <- as(reanno_object, "list")
reanno_s4 <- as.reanno(reanno_s3)

```

ncol,PAC-method	<i>number of samples</i>
-----------------	--------------------------

Description

number of samples

Usage

```
## S4 method for signature 'PAC'
ncol(x)
```

Arguments

x A S4 object of class PAC.

Value

The number of samples in a PAC-object.

Examples

```
load(system.file("extdata", "drosophila_sRNA_pac_filt_anno.Rdata",
  package = "seqpac", mustWork = TRUE))

# extra fuctionality with s4 PAC-object:
names(pac)
length(pac)
nrow(pac)
ncol(pac)
rownames(pac)
colnames(pac)
pheno(pac)
head(anno(pac))
head(counts(pac))
head(norm(pac)$cpm)
```

<code>norm</code>	<code>norm(PAC)</code>
-------------------	------------------------

Description

Access the norm list with normalized table in S4 PAC

Usage

```
norm(object)

## S4 method for signature 'PAC'
norm(object)

## S4 replacement method for signature 'PAC,list'
norm(object) <- value
```

Arguments

<code>object</code>	A S4 PAC-object containing a norm list ('folder') with normalized data.frames. Can be generated by the PAC_norm function.
<code>value</code>	List of data.frames having the same column names as row names in Pheno (sample ID) and row names as row names in Anno (sequences.)

Value

The list of normalized tables (data.frames) from a PAC-object.
Updated S4 PAC-object.

See Also

Other PAC methods: [anno\(\)](#), [counts\(\)](#), [names,PAC-method](#), [pheno\(\)](#), [show.PAC\(\)](#), [summary\(\)](#)

Examples

```
load(system.file("extdata", "drosophila_sRNA_pac_filt_anno.Rdata",
                 package = "seqpac", mustWork = TRUE))

# extra fuinctionality with s4 PAC-object:
names(pac)
length(pac)
nrow(pac)
ncol(pac)
rownames(pac)
colnames(pac)
pheno(pac)
head(anno(pac))
head(counts(pac))
head(norm(pac)$cpm)
```

nrow,PAC-method	<i>number of sequences</i>
-----------------	----------------------------

Description

number of sequences
number of sequences

Usage

```
## S4 method for signature 'PAC'
nrow(x)

## S4 method for signature 'reanno'
nrow(x)
```

Arguments

x A S4 object of class PAC.

Value

The number of sequences in a PAC-object.
Number of sequences in the reanno-object.

Examples

```
load(system.file("extdata", "drosophila_sRNA_pac_filt_anno.Rdata",
  package = "seqpac", mustWork = TRUE))

# extra fuctionality with s4 PAC-object:
names(pac)
length(pac)
nrow(pac)
ncol(pac)
rownames(pac)
colnames(pac)
pheno(pac)
head(anno(pac))
head(counts(pac))
head(norm(pac)$cpm)
```

overview

overview(reanno)

Description

Access the overview table in a S4 reanno object

Usage

```
overview(object)

## S4 method for signature 'reanno'
overview(object)

## S4 replacement method for signature 'reanno,data.frame'
overview(object) <- value
```

Arguments

object	A S4 reanno-object containing an overview table.
value	Tibble data.frame with the same property as the Overview table of the reanno object, e.g. having the same names in seq column as row names (sequences) in the original PAC-object used to generate the reanno object. See examples and <code>?make_reanno</code> for details.

Value

The overview table of a reanno-object as a tibble data.frame (class `tbl_df`, `tbl`, `data.frame`).
Updated reanno-object.

See Also

Other reanno methods: [full\(\)](#)

Examples

```
#####
##### Create a reanno object

## First load a PAC- object

load(system.file("extdata", "drosophila_sRNA_pac_filt_anno.Rdata",
  package = "seqpac", mustWork = TRUE))
anno(pac) <- anno(pac)[,1, drop = FALSE]

## Then specify paths to fasta references
# If you are having problem see the vignette small RNA guide for more info.

trna_path <- system.file("extdata/trna", "tRNA.fa",
  package = "seqpac", mustWork = TRUE)
rrna_path <- system.file("extdata/rrna", "rRNA.fa",
  package = "seqpac", mustWork = TRUE)

ref_paths <- list(trna= trna_path, rrna= rrna_path)

## Add output path of your choice.
# Here we use the R temporary folder depending on platform
if(grepl("windows", .Platform$OS.type)){
  output <- paste0(tempdir(), "\\seqpac\\test")
}else{
  output <- paste0(tempdir(), "/seqpac/test")}

## Make sure it is empty (otherwise you will be prompted for a question)
out_fls <- list.files(output, recursive=TRUE)
closeAllConnections()
suppressWarnings(file.remove(paste(output, out_fls, sep="/")))

## Then map your PAC-object against the fasta references
map_reanno(pac, input=ref_paths, output=output,
  type="internal", mismatches=2, import="biotype",
  threads=2, keep_temp=FALSE)

## Then generate a reanno-object of the temporary bowtie-files
reanno_object <- make_reanno(output, PAC=pac, mis_fasta_check = TRUE)

## Accessing content and S4/S3 conversion:
names(reanno_object)
overview(reanno_object)
full(reanno_object)
rownames(reanno_object)
length(reanno_object)
nrow(reanno_object)
```

```
reanno_s3 <- as(reanno_object, "list")
reanno_s4 <- as.reanno(reanno_s3)
```

PAC-class

*S4 object for PAC***Description**

S4 version of the PAC object.

Usage

```
PAC(Pheno, Anno, Counts, norm, summary)
```

Arguments

Pheno	Phenotype data.frame table with sample ID as row names and where columns are variables associated with the samples. Can be generated using the make_pheno function.
Anno	Annotation data.frame table with unique sequences as row names and where columns are variables associated with the sequences.
Counts	Counts table (data.frame) with unique sequences as row names (identical to Anno) and where columns are sample ID (identical to row names for Pheno). Should contain raw counts and can be generated using the make_counts function.
norm	List of data.frames. May be regarded as a "folder" with normalized counts tables. The listed data.frames must have identical sequence names as Counts/Anno (rows) and sample IDs (columns) as Counts/Pheno. Can be generated using the PAC_norm function, but seqpac functions will attempt use any normalized table stored in norm that are in agreement with the above cafeterias.
summary	List of data.frames, just like norm, but contains summarized raw or normalized counts (e.g. means, standard errors, fold changes). Important, the listed data.frames must have identical sequence names as Counts/Anno (rows), but columns don't need sample IDs. Can be generated using the PAC_summary function, but seqpac functions will attempt use any summarized table stored in the summary "folder".

Details

Just like the S3 version, all sub-tables must have identical sequence (row) names. To extract tables from S4 objects, use the getter commands or use @ (e.g. PAC@Pheno) just like \$ for the S3 version (e.g. pheno(PAC)). You may also use a coercion method, e.g. PAC_S3 <- as(PAC_S4, "list")

Holds up to 5 slots:

Pheno data.frame > information about the samples

Anno data.frame > sequences annotations

Counts data.frame > sequence counts over samples

norm (optional) list of data.frames > normalized counts

summary (optional) list of data.frames > summarized counts or normalized counts

Value

Generates a S4 PAC-object.

Examples

```
load(system.file("extdata", "drosophila_sRNA_pac_filt_anno.Rdata",
                 package = "seqpac", mustWork = TRUE))

# check type
class(pac)
isS4(pac)

# Turns S4 PAC object into a S3
pac_s3 <- as(pac, "list")

# Turns S3 PAC object into a S4
pac_s4 <- as.PAC(pac_s3)
```

PAC_analyze

PAC_analyze

Description

Given a PAC object and targets, this wrapper will perform several analyses of the PAC object and return a PDF with plots. More specifically, the plot will contain a jitterplot of fold changes, stack bar of mean counts/cpm, size distribution stack bars, piecharts, PCA plots, and if specified, a volcano plot based off adjusted p-values and fold changes from a DESeq2 analysis.

Usage

```
PAC_analyze(
  PAC,
  pheno_target = NULL,
  norm = NULL,
  anno_target = NULL,
  model = NULL
)
```

Arguments

PAC	PAC-object containing Anno and a summary list-folder with summarized dataframe(s) for example generated by PAC_summary. The function can also be applied on a dataframe containing a column with annotation groupings (e.g. biotype) and a column with summarized data (e.g. log2 fold changes). Rows should be unique sequences as the rownames of a PAC summary object.
pheno_target	List with: 1st object being a character vector of target column in Pheno and 2nd object being a character vector of the target group(s) in the target Pheno column (1st object).
norm	Character indicating what type of data to be used. If 'counts', the raw counts in Counts will be used (default). If "cpm", cpm values will be used. As of now, only "counts" and "cpm" may be used for this wrapper.
anno_target	Character vector with the name of the target column in Anno or the name of the annotation column in case of input being a dataframe.
model	Character of model used to run PAC_deseq.

Value

a list of plots in R, and a pdf file in home directory called "Results_seqpac.pdf".

Examples

```
load(system.file("extdata", "drosophila_sRNA_pac_filt_anno.Rdata",
                 package = "seqpac", mustWork = TRUE))

result_list <- PAC_analyze(pac,
  pheno_target=list("stage"),
  norm="cpm",
  anno_target=list("Biotypes_mis0"),
  model=~stage+batch)
```

PAC_annotate	<i>Wrapper for annotation of PAC object</i>
--------------	---

Description

PAC_annotate Annotates the provided PAC object to a genome or biotypes

Usage

```
PAC_annotate(genome = NULL, biotype = NULL, output = NULL, PAC)
```



```
## load a PAC-object and remove previous mapping from anno:
load(system.file("extdata", "drosophila_sRNA_pac_filt_anno.Rdata",
                package = "seqpac", mustWork = TRUE))
anno(pac) <- anno(pac)[,1, drop = FALSE]

output <- paste0(tempdir(), "/seqpac/test")

#as we only test this against the mycoplasma genome,
#we expect a small overlap (0.03%)

pac <- PAC_annotate(PAC=pac,
                    genome = genome,
                    output = output)
```

PAC_check

Checks PAC format

Description

PAC_check Quickly checks rownames in PAC.

Usage

```
PAC_check(PAC, sample_check = NULL, seq_check = NULL)
```

Arguments

PAC	PAC-list object.
sample_check	Can be provided either as a data frame or the name of a data frame in PAC. Will check both the row and column names of the object for identical sample names in PAC. A simple character vector can also be provided.
seq_check	Can be provided either as a data frame or the name of an data frame in PAC. Will check only the row names of the object for identical sequence names in PAC. A simple character vector can also be provided.

Details

Given a PAC object the function will cross check the rownames and colnames across Pheno, Anno and Counts.

Value

If PAC format is correct will return logical TRUE, else it will throw an error.

See Also

<https://github.com/OestLab/seqpac> for updates on the current package.

Other PAC generation: `PAC_create()`, `make_PAC()`, `make_counts()`, `make_cutadapt()`, `make_pheno()`, `make_trim()`, `merge_lanes()`

Examples

```
load(system.file("extdata", "drosophila_sRNA_pac_filt_anno.Rdata",
                 package = "seqpac", mustWork = TRUE))

PAC_check(pac) # TRUE

# Remove first row in Anno and check compatibility
new_Anno <- anno(pac)[-1,]
#PAC_check(pac, seq_check=new_Anno) # Error

# Add to pac an check
# anno(pac) <- new_Anno #error
```

PAC_covplot

Plot sequence coverage over a reference

Description

PAC_covplot Plotting sequences in a PAC object using an PAC mapping object.

Usage

```
PAC_covplot(
  PAC,
  map,
  summary_target = NULL,
  map_target = NULL,
  style = "line",
  xseq = TRUE,
  colors = NULL,
  check_override = FALSE
)
```

Arguments

PAC	PAC-list object.
map	PAC_map object generated by <i>PAC_mapper</i> .

summary_target	List with 2 character vectors: 1st object: Character indicating the name of the target PAC summary object which should be used as input data for the plots. If left empty the 1st summary object will be used. 2nd object: Character vector indicating the names of the columns in the summary object from which individual graphs should be plotted. If left empty all summary columns are plotted in the same graph. Summaries are generated by the PAC_summary function or custom generated by the user and stored in the summary 'folder' of a PAC object.
map_target	(optional) Character vector. Important: This is similar to an anno_target, but instead extracts target references in the PAC_mapping object. Should contain search terms that can find unique strings in the reference names. The search terms are parsed to grepl("<search terms>", names(<PAC_mapper object>)). (default=NULL)
style	Character string. If style="line" (default) then graphs are plotted as simple path/line plots (ggplot2::geom_path). If style="solid" then the area under the line is filled with same color as the lines but with added transparency (ggplot2::geom_line + geom_ribbon, alpha=0.5).
xseq	Logical indicating whether the nucleotides should be plotted on the x-axis. Plotting long references with xseq=FALSE will increase script performance. (default=TRUE).
colors	Character vector indicating the rgb colors to be parsed to ggplot2 for plotting the coverage lines (default = c("black", "red", "grey", "blue"))
check_override	Logical, whether all sequences names in PAC must be found in the PAC_map object (TRUE) or not (FALSE). If the PAC_map object was generated from the provided PAC objects, then all sequences should match. If the PAC list object was subsetting or the PAC_map is used with an independent PAC list object in which not all sequences are available, then this function will create an error, unless check_override=TRUE. In such case, all missing sequences in the PAC object will automatically be assigned a 0-value in all summary_target columns.

Details

Given a PAC object and a map object generated by *PAC_mapper* this function will attempt to plot the sequence coverage over long references.

Value

Plot list with plots generated by ggplot2

See Also

<https://github.com/OestLab/seqpac> for updates on the current package.

Other PAC analysis: `PAC_deseq()`, `PAC_filter()`, `PAC_filtsep()`, `PAC_gtf()`, `PAC_jitter()`, `PAC_mapper()`, `PAC_nbias()`, `PAC_norm()`, `PAC_pca()`, `PAC_pie()`, `PAC_saturation()`, `PAC_sizedist()`, `PAC_stackbar()`, `PAC_summary()`, `PAC_trna()`, `as.PAC()`, `filtsep_bin()`, `map_rangetype()`, `trna_class()`


```
cowplot::plot_grid(plotlist= cov_tRNA_sub)
```

PAC_create

Wrapper for creation of PAC object

Description

PAC_create Merges lanes, performs adaptor trimming and makes a first PAC object with count and basic phenotypic information

Usage

```
PAC_create(lanes = NULL, trim = NULL, input, output = NULL)
```

Arguments

lanes	allows to define whether lanes should be merged with merge_lanes (may be necessary after using e.g BaseSpace to download fasta files from Illumina cloud). Default=NULL, where no lane merging will be performed.
trim	Character vector to define whether trimming of adaptors should be performed. Current accepted values are "default_neb" and "default_illumina", trimming adaptors from library preparation with NebNext or Illumina technologies respectively. More advanced and manual adaptor trimming is possible with make_trim . Default=NULL, where no adaptor trimming will be performed.
input	Character indicating path to where input files can be found in fasta or fastq format. Depending on value of lanes and trim, the function assumes whether to expect files in path to be trimmed or not and whether to merge lanes.
output	Character indicating path to where output files will be stored, if applicable (such as in merge_lanes). Default=NULL.

Details

Given (at minimum) an path to input files to be analyzed, PAC_create will wrap around all necessary preparatory steps ([merge_lanes](#), [make_counts](#), and [make_PAC](#)) to produce a PAC object. This wrapper comes with the caveat that it uses default values in most functions. If that does not suit your analysis, we recommend you to run each function on its own!

Value

PAC object with values in pheno and count

See Also

<https://github.com/OestLab/seqpac> for updates on the current package.

Other PAC generation: [PAC_check\(\)](#), [make_PAC\(\)](#), [make_counts\(\)](#), [make_cutadapt\(\)](#), [make_pheno\(\)](#), [make_trim\(\)](#), [merge_lanes\(\)](#)

Examples

```
#####
### test the map_rangetype function
# More complicated examples can be found in the vignette.
##-----

# First create an annotation blank PAC with group means
sys_path = system.file("extdata", package = "seqpac", mustWork = TRUE)
fq <- list.files(path = sys_path, pattern = "fastq", all.files = FALSE,
                 full.names = TRUE)

# Create an output folder
input <- paste0(tempdir(), "/lanes/")
output <- paste0(tempdir(), "/merged/")
dir.create(input, showWarnings=FALSE)
dir.create(output, showWarnings=FALSE)

# Fix compatible file names
file.copy(from = fq, to = input)
old_fls <- list.files(input, full.names=TRUE)
new_sample <- c(rep("sample1_", times=3), rep("sample2_", times=3))
new_lane <- rep(c("lane1", "lane2", "lane3"), times=2)
new_fls <- paste0(input, new_sample, new_lane, ".fastq.gz")
file.rename(from = old_fls, to = new_fls)

# Then merge the fastq files
pac <- PAC_create(lanes=TRUE, trim="default_neb", input, output)
```

PAC_deseq

A wrapper to DESeq2 that apply differential expression analysis on a PAC object

Description

PAC_deseq DESeq2 analysis on PAC_object.

Usage

```
PAC_deseq(
  PAC,
  model,
  deseq_norm = FALSE,
  test = "Wald",
  fitType = "local",
  threads = 1,
  pheno_target = NULL
)
```

Arguments

PAC	PAC object containing a Pheno data.frame with samples as row names, an Anno data.frame with sequences as row names and a Counts table with raw counts. The Counts table must have the sample names as column names and sequences as row names.
model	Character vector describing the statistical model based on column names in Pheno.
deseq_norm	Logical whether to return deseq normalized values or raw counts (default=FALSE).
test	Character parsed directly to DESeq that controls what type of statistical test that should be used. Alternatives are either "Wald" (Wald significance test) or "LTR" (likelihood ratio test/chi-squared test). See DESeq for more details. (Default="Wald")
fitType	Character parsed directly to DESeq that controls what type of dispersion fit that should be used. Alternatives are either "parametric" (dispersion-mean relation), "local" (local regression of log dispersions), "mean" (mean of gene-wise dispersion). See DESeq for more details. (Default="local")
threads	Integer number of threads to run in parallel.
pheno_target	(optional) List with: 1st object being a character indicating the main target column in Pheno. 2nd object being a character vector of the target group(s) in the target Pheno column (1st object). Important: In PAC_deseq, pheno_target controls the main comparative factor category from which a summarized table and plots will be generated. If, for instance, a target column named "groups" in pheno(PAC) contains "control" and "treatment" categories, setting pheno_target=list("groups", c("treatment", "controls")) ensures that "treatment" is presented 1st in the factor levels, making for example log2FC appear as "treatment vs control". As default, pheno_target=NULL will result in the factor levels being automatically sorted in ascending order, which in the example above would result in control vs treatment. If no pheno_target is given, the first feature in the model will also be the main comparison presented in the graphs and summary tables.

Details

Given a PAC object this function will apply a differential expression analysis using DESeq2.

Value

A list of objects:

1st object - Summarized result table merged with anno(PAC)

2nd object - Target graphs (p-val distribution and volcano)

3rd object - All output from [DESeq](#)

See Also

<https://github.com/OestLab/seqpac> for updates on the current package.

Other PAC analysis: `PAC_covplot()`, `PAC_filter()`, `PAC_filtsep()`, `PAC_gtf()`, `PAC_jitter()`, `PAC_mapper()`, `PAC_nbias()`, `PAC_norm()`, `PAC_pca()`, `PAC_pie()`, `PAC_saturation()`, `PAC_sizedist()`, `PAC_stackbar()`, `PAC_summary()`, `PAC_trna()`, `as.PAC()`, `filtsep_bin()`, `map_rangetype()`, `tRNA_class()`

Examples

```
# Note, these examples will generate some warnings since data is based on
# heavily down-sampled fastq files, where many sequences receives low counts
# in specific groups.

## Load test data
load(system.file("extdata", "drosophila_sRNA_pac_filt_anno.Rdata",
                 package = "seqpac", mustWork = TRUE))

## Simple model with embryonic stages using Wald test with local fit (default)
table(pheno(pac)$stage)
output_deseq <- suppressWarnings(PAC_deseq(pac, model= ~stage, threads=1))

## Batch corrected, graphs are generated for 'stage' (=first in the model)
output_deseq <- suppressWarnings(PAC_deseq(pac, model= ~stage + batch,
                                           threads=1))

## Using pheno_target
output_deseq <- suppressWarnings(PAC_deseq(pac, model= ~stage + batch,
                                           pheno_target=list("batch"),
                                           threads=1))

## With pheno_target we can change the direction for the comparison
# Stage5 vs Stage3 (reverse order):
output_deseq <- suppressWarnings(PAC_deseq(pac, model= ~stage + batch,
                                           pheno_target = list("stage", c("Stage5", "Stage3")),
                                           threads=1))

## In the output you find PAC merged results, target plots and output_deseq
names(output_deseq)
tibble::as_tibble(output_deseq$result)
```

PAC_filter

Filter a PAC object on sequence size and coverage

Description

PAC_filter Filter PAC objects.

Usage

```
PAC_filter(
  PAC,
  nucleotide_range = NULL,
  threshold = 0,
  coverage = 0,
  norm = "counts",
  subset_only = FALSE,
  stat = FALSE,
  pheno_target = NULL,
  anno_target = NULL,
  allbut = FALSE
)
```

Arguments

PAC	PAC-list object containing an Anno data.frame with sequences as row names and a Counts table with raw counts or counts per million (cpm).
nucleotide_range	Integer vector giving the size interval, as c(min,max), that should be saved (default=c(min,max)).
threshold	Integer giving the threshold in counts or normalized counts that needs to be reached for a sequence to be included (default=0).
coverage	Integer giving the percent of independent samples that need to reach the threshold for a sequence to be included (default=0).
norm	Character specifying if filtering should be done using "counts", "cpm" or another normalized data table in PAC\$norm (default="counts").
subset_only	Logical whether only subsetting using pheno_target and/or anno_target should be done. If subset=FALSE (default) both subsetting and other filtering will be done.
stat	(optional) Logical specifying if a coverage graph should be generated and if users should be prompted prior to proceeding. (default=FALSE).
pheno_target	(optional) List with: 1st object being a character vector of target column in Pheno, 2nd object being a character vector of the target group(s) in the target Pheno column (1st object). (default=NULL)
anno_target	(optional) List with: 1st object being a character vector of target column in Anno, 2nd object being a character vector of the target type/biotypes(s) in the target Anno column (1st object). (default=NULL)
allbut	(optional) TRUE or FALSE to give option to include all possible variables but the ones defined in pheno_target and anno_target. Useful when you wish to for example filter out one particular variable.

Details

Given a PAC object the function will extract sequences within a given size interval and percent coverage across independent samples.

Value

A list of objects: PAC object with filtered data. (optional) A coverage plot

See Also

<https://github.com/OestLab/seqpac> for updates on the current package.

Other PAC analysis: `PAC_covplot()`, `PAC_deseq()`, `PAC_filtsep()`, `PAC_gtf()`, `PAC_jitter()`, `PAC_mapper()`, `PAC_nbias()`, `PAC_norm()`, `PAC_pca()`, `PAC_pie()`, `PAC_saturation()`, `PAC_sizedist()`, `PAC_stackbar()`, `PAC_summary()`, `PAC_trna()`, `as.PAC()`, `filtsep_bin()`, `map_rangetype()`, `tRNA_class()`

Examples

```
load(system.file("extdata", "drosophila_sRNA_pac_filt_anno.Rdata",
  package = "seqpac", mustWork = TRUE))

###-----
## Extracts all sequences between 10-80 nt in length with at
## least 5 counts in 20% of all samples.
pac_lowfilt <- PAC_filter(pac, nucleotide_range=c(10,80), threshold=5,
  coverage=20, norm = "counts",
  pheno_target=NULL, anno_target=NULL)

###-----
## Extracts sequences with 22 nt size and the samples in Batch1 and Batch2.
pac_subset <- PAC_filter(pac, subset_only = TRUE,
  pheno_target=list("batch", c("Batch1", "Batch2")),
  anno_target=list("Size", "22"))

###-----
## Extracts all sequences with >=5 counts in 100% of samples a within stage
filtsep <- PAC_filtsep(pac, norm="counts", threshold=5,
  coverage=100, pheno_target= list("stage"))

pac_filt <- PAC_filter(pac, subset_only = TRUE,
  anno_target= unique(do.call("c", as.list(filtsep))))

###-----
## Extract all biotypes but tRNA in all samples but in sample fastq_1
pac_filt <- PAC_filter(pac, anno_target=list("Biotypes_mis0","tRNA"),
  pheno_target=list("sample","fastq1"), allbut=TRUE)
```

PAC_filtsep

*Generates filtered sequence name list separated on group***Description**

PAC_filtsep Group separated filtered name list

Usage

```
PAC_filtsep(
  PAC,
  norm = "counts",
  threshold = 10,
  coverage = 100,
  pheno_target = NULL,
  output = "sequence"
)
```

Arguments

PAC	PAC-list object containing an Pheno data.frame with samples as row names and a Counts table with raw counts or normalized counts table containing for example counts per million (cpm; generated by PAC_norm).
norm	Character specifying if filtering should be done using "counts", "cpm" or another normalized data table in PAC\$norm (default="counts").
threshold	Integer giving the threshold in counts PAC\$Counts or normalized counts (table in PAC\$norm) that needs to be reached for a sequence to be included (default=10).
coverage	Integer giving the percent of independent samples of each group that needs to reach the threshold for a sequence to be included (default=100).
pheno_target	(optional) List with: 1st object being a character vector of target column in Pheno, 2nd object being a character vector of the target group(s) in the target Pheno column (1st object). (default=NULL)
output	Specifies the output format. If output="sequence" (default), then a data.frame is returned where each column contains the sequences names that passed the filter for a specific group specified in pheno_target. If output="binary", then the resulting data.frame will be converted into a binary (hit=1, no hit=0) data.frame. See filtsep_bin .

Details

Given a PAC object the function will extract sequences within the given filter within a given group.

Value

A data.frame (see output for details).

See Also

<https://github.com/OestLab/seqpac> for updates on the current package.

Other PAC analysis: `PAC_covplot()`, `PAC_deseq()`, `PAC_filter()`, `PAC_gtf()`, `PAC_jitter()`, `PAC_mapper()`, `PAC_nbias()`, `PAC_norm()`, `PAC_pca()`, `PAC_pie()`, `PAC_saturation()`, `PAC_sizedist()`, `PAC_stackbar()`, `PAC_summary()`, `PAC_trna()`, `as.PAC()`, `filtsep_bin()`, `map_rangetype()`, `tRNA_class()`

Examples

```
load(system.file("extdata", "drosophila_sRNA_pac_filt_anno.Rdata",
                package = "seqpac", mustWork = TRUE))

## Keep sequences with 5 counts (threshold) in 100% (coverage) of
## samples in a group:
# Use PAC_filtsep to find sequences
filtsep <- PAC_filtsep(pac, norm="counts", threshold=5,
                      coverage=100, pheno_target= list("stage"))

# Filter by unique sequences passing filtsep
filtsep <- unique(do.call("c", as.list(filtsep)))
pac_filt <- PAC_filter(pac, subset_only = TRUE, anno_target= filtsep)

# Find overlap
olap <- reshape2::melt(filtsep,
                      measure.vars = c("Stage1", "Stage3", "Stage5"),
                      na.rm=TRUE)

## Upset plot using the UpSetR package
# (when output="binary" PAC_filtsep uses filtsep_bin for binary conversion)
# Use PAC_filtsep with binary output
filtsep_bin <- PAC_filtsep(pac, norm="counts", threshold=5,
                          coverage=100, pheno_target= list("stage"),
                          output="binary")

# Plot Wenn diagram or UpSetR
#
# plot(venneuler::venneuler(data.frame(olap[,2], olap[,1])))
#
# UpSetR::upset(filtsep_bin, sets = colnames(filtsep_bin),
#               mb.ratio = c(0.55, 0.45), order.by = "freq", keep.order=TRUE)
```

PAC_gtf

Annotate against a GTF file

Description

This function will annotate a PAC object using input from a GTF/GFF file.

Usage

```
PAC_gtf(
  PAC,
  genome = NULL,
  mismatches = 3,
  return = "simplify",
  stranded = FALSE,
  gtf = NULL,
  targets = NULL,
  threads = 1
)
```

Arguments

PAC	PAC-list object. The Anno object needs to contain genome coordinates in the following columns: "chr", "start", "end", and "strand". These coordinates are easily obtained by using PAC_genome
genome	Character indicating the path to a reference genome file in fasta format. Note that a bowtie index, created by for example Rbowtie::bowtie-build, must be contained in the same folder and having the same basename as the fasta.
mismatches	Integer indicating the number of allowed mismatches for the genome mapping.
return	Character indicating what information to return. If return="simplify" (default), a table with one unique sequence per row will be returned. Multiple hits between genomic coordinates and gtf coordinates will be merged and only unique annotations will be reported. If return="full", a list will be returned containing 1 table reporting all annotations for each genomic coordinate of each unique sequence. If return="all", both a simplified table and a full annotation list will be returned as a list. If return="merge", a simplified table will be merged with the Anno table of the provided PAC object, and the updated PAC object containing the new annotations will be returned.
stranded	Logical whether mapping should be strand specific. If stranded=TRUE, then hits between feature and PAC read sequence will not be reported if the feature is located on opposite strand. If stranded=FALSE (default), then both sense and anti-sense overlaps will be reported. Note, stranded=FALSE is recommended since PAC_gtf will report on which strand each feature/sequence is mapping. Thus, strand specific analysis can be done in hindsight.
gtf	Named list of characters, indicating file path(s) to other gtf files with differing formats. Can also directly be provided as a tibble dataframe in a named list.
targets	Named list of character vectors indicating target columns in the listed gtf files in <i>gtf_other</i> . Important, the listed objects must have the same length and names as in <i>gtf</i> . The vector indicates the column names as if imported by rtracklayer::readGFF.
threads	Integer indicating the number of parallel processes.

Details

Given a PAC object and a gtf formatted annotation file(s), this function will attempt to annotate sequences mapped to a reference genome against genomic coordinates in the gtf file(s). In case no genomic mapping coordinates are available in the PAC object, the function provides a backdoor into the *PAC reannotation* workflow, where genome mapping is performed using Bowtie.

Value

List, tibble dataframe or updated PAC object. See option *return* for more information.

See Also

<https://github.com/OestLab/seqpac> for updates.

Other PAC analysis: `PAC_covplot()`, `PAC_deseq()`, `PAC_filter()`, `PAC_filtsep()`, `PAC_jitter()`, `PAC_mapper()`, `PAC_nbias()`, `PAC_norm()`, `PAC_pca()`, `PAC_pie()`, `PAC_saturation()`, `PAC_sizedist()`, `PAC_stackbar()`, `PAC_summary()`, `PAC_trna()`, `as.PAC()`, `filtsep_bin()`, `map_rangetype()`, `trna_class()`

Examples

```
# Load PAC
load(system.file("extdata", "drosophila_sRNA_pac_filt_anno.Rdata",
                 package = "seqpac", mustWork = TRUE))

# Create a gtf file from PAC coordinates
anno <- anno(pac)
anno <- anno[!grepl("Warning", anno$mis0_chromosomes_genome),]
anno <- anno[!is.na(anno$mis0_chromosomes_genome),]
coord <- anno$mis0_chromosomes_genome
coord <- suppressWarnings(do.call("rbind", strsplit(coord, "\\|"))[,1])
coord <- suppressWarnings(do.call("rbind", strsplit(coord, "\\;start=|;")))
gr <- GenomicRanges::GRanges(seqnames=coord[,1],
                             IRanges::IRanges(as.numeric(coord[,2]),
                                                as.numeric(coord[,2])+
                                                anno$Size ),
                             strand=coord[,3])

GenomicRanges::mcols(gr) <- data.frame(biotype=anno$Biotypes_mis3,
                                       bio_zero=as.character(anno$mis0_bio))
spl <- sample(1:length(gr), round(length(gr)*0.3), replace=FALSE)
gr1 <- gr[spl]
gr2 <- gr[!1:length(gr) %in% spl]

# Prepare temp folder and save artificial gtf
out1 <- paste0(tempdir(), "/temp1.gtf")
out2 <- paste0(tempdir(), "/temp2.gtf")

rtracklayer::export(gr1, out1, format="gtf")
rtracklayer::export(gr2, out2, format="gtf")

# Make sure PAC contains full genome coordinates
```

```

# (In the add_reanno function you may choose how many coordinates to report)
# (If there are more, a 'Warning' will be added to the annotation)
# (Here we remove those to avoid problems)

new_anno <- anno(pac) [, grepl("chromosomes_genome|Size",
                              colnames(anno(pac)))]

test <- new_anno[, 3:ncol(new_anno)]
test <- apply(test, 2, function(x){substr(x, 1, 7)})
test <- apply(test, 1, function(x){paste(x, collapse = "")})
new_anno$temp <- ifelse(grepl("Warning", test), "rm", "keep")
anno(pac) <- new_anno
pac_small <- PAC_filter(pac, subset_only=TRUE, nucleotide_range = c(20,22),
                       anno_target=list("temp", "keep"))

# Run PAC_gtf
gtf <- list(gtf1=out1, gtf2=out2)
target <- list(gtf1=c("biotype","bio_zero"), gtf2=c("biotype","bio_zero"))

pac_merge <- PAC_gtf(pac_small, mismatches=0, return="merge",
                    gtf=gtf, target=target, threads=2)

```

PAC_jitter

Plots jitter plot from PAC object

Description

PAC_jitter Plots a jitter plot using the information in the Anno and Counts tables in a PAC object.

Usage

```

PAC_jitter(
  PAC,
  summary_target = NULL,
  anno_target = NULL,
  style = "jitter",
  limits = NULL,
  ypos_n = NULL,
  colors = NULL,
  box = TRUE
)

```

Arguments

PAC	PAC-object containing Anno and a summary list-folder with summarized dataframe(s) for example generated by PAC_summary. The function can also be applied on a dataframe containing a column with annotation groupings (e.g. biotype) and a column with summarized data (e.g. log2 fold changes). Rows should be unique sequences as the rownames of a PAC summary object.
-----	---

summary_target	List with: 1st object being a character vector of the target dataframe in summary and 2nd object being a character vector of the target column(s) in that dataframe. In case of input being a dataframe, summary_target can be a character vector indicating the column with the summarized data.
anno_target	Character vector with the name of the target column in Anno or the name of the annotation column in case of input being a dataframe.
style	Character. If style="jitter" (default) the jitter-plots will be returned. If style="violin", a violin-plot will instead be returned.
limits	Integer vector with the y-limits to be parsed to ggplot2.
ypos_n	Integer setting the y position of the n counts to be parsed to ggplot2.
colors	Character vector with color codes to be parsed to ggplot2.
box	Logical whether a boxplot should be plotted or not (default=TRUE)

Details

Given a PAC object with grouped summaries, the function will use column(s) in the Anno object to group assigned summary columns (summary_target) and plot a jitter plot.

Value

A plot-list object with jitter plots generated by ggplot2.

See Also

<https://github.com/OestLab/seqpac> for updates on the current package.

Other PAC analysis: `PAC_covplot()`, `PAC_deseq()`, `PAC_filter()`, `PAC_filtsep()`, `PAC_gtf()`, `PAC_mapper()`, `PAC_nbias()`, `PAC_norm()`, `PAC_pca()`, `PAC_pie()`, `PAC_saturation()`, `PAC_sizedist()`, `PAC_stackbar()`, `PAC_summary()`, `PAC_trna()`, `as.PAC()`, `filtsep_bin()`, `map_rangetype()`, `tRNA_class()`

Examples

```
## Prepare
load(system.file("extdata", "drosophila_sRNA_pac_filt_anno.Rdata",
                 package = "seqpac", mustWork = TRUE))
pac <- PAC_norm(pac, norm="cpm")

pac <- PAC_summary(PAC=pac, norm = "cpm", type = "log2FC",
                  pheno_target=list("stage"))
pac <- PAC_summary(PAC=pac, norm = "cpm", type = "percentgrand",
                  pheno_target=list("stage"))

## Jitter plots
plots_FC <- PAC_jitter(pac, summary_target=list("Log2FC_stage"),
                      anno_target=list("Biotypes_mis0"))
plots_FCgrand <- PAC_jitter(pac, summary_target=list("percGrand_stage"),
                           anno_target=list("Biotypes_mis0"))

cowplot::plot_grid(plotlist=plots_FC, nrow = 3, ncol = 1)
```

```

cowplot::plot_grid(plotlist=plots_FCgrand, nrow = 3, ncol = 1)

## Violin plots instead
plots_FC <- PAC_jitter(pac, style="violin",
                      summary_target=list("Log2FC_stage"),
                      anno_target=list("Biotypes_mis0"))
plots_FCgrand <- PAC_jitter(pac, style="violin",
                           summary_target=list("percGrand_stage"),
                           anno_target=list("Biotypes_mis0"))

cowplot::plot_grid(plotlist=plots_FC, nrow = 3, ncol = 1)
cowplot::plot_grid(plotlist=plots_FCgrand, nrow = 3, ncol = 1)

## Violin with changed biotype order
new_order <- as.character(unique(anno(pac)$Biotypes_mis0))[c(2,4,3,6,7,5,1)]
plots_FC <- PAC_jitter(pac, style="violin",
                      summary_target=list("Log2FC_stage"),
                      anno_target=list("Biotypes_mis0", new_order))
cowplot::plot_grid(plotlist=plots_FC, nrow = 3, ncol = 1)

```

PAC_mapper

Advanced sequence mapping of a PAC object

Description

PAC_mapper Map sequences against a small reference.

Usage

```

PAC_mapper(
  PAC,
  input,
  mismatches = 0,
  multi = "remove",
  threads = 1,
  N_up = "",
  N_down = "",
  report_string = FALSE,
  override = FALSE
)

```

Arguments

PAC PAC-list object.

input	Character indicating the path to the fasta (.fa) reference file or a DNASTringSet with already loaded reference sequences. If a Bowtie index is missing for the reference, PAC_mapper will attempt to temporarily generate such index automatically. Thus, large references are discouraged. Instead, we suggest you use the original reanno workflow for large references.
mismatches	Integer indicating the number of mismatches that should be allowed in the mapping.
multi	Character indicating how to deal with multimapping. If multi="keep", query sequences that maps multiple times to the same reference sequence will be reported >1 times in the output (indicated by .1, .2, .3 etc. in the reported sequence name). If multi="remove" (default), then all multimapping sequences will be removed, resulting in 1 row for each query sequence that maps to the target reference sequence. The function will always give a warning if a query sequence maps to multiple sites within a reference sequence. However, this function discriminate multimapping only within a reference sequence. Thus, if the fasta input contains multiple reference sequences, a query sequence may be reported in multiple references sequences.
threads	Integer indicating the number of parallel processes that should be used.
N_up	Character indicating a sequence that should be added to the reference at the 5' end prior to mapping. A wild card nucleotides "NNN" (any of C, T, G, A) can for example be added for mapping non-perfect reference hits. No nucleotides are added by default.
N_down	Character. Same as N_up but indicating a sequence that should be added to the reference at the 3' end. Useful for tRNA analysis where the reference do not contain pre-processed tRNA. Setting N_down="NNN" or "CCA" (in many species CCA is added to mature tRNA) will allow mapping against the mature tRNA. No nucleotides are added by default.
report_string	Logical whether an alignment string that shows where sequences align against the reference in a character format. Works well with tRNA, but makes the Alignments object difficult to work with when longer references are used (default=FALSE).
override	Logical whether or not the map_reanno function should prompt you for a question if there are files in the temporary path. As default, override=FALSE will prevent deleting large files by accident, but requires an interactive R session. Setting override=TRUE may solve non-interactive problems.

Details

Given a PAC object and the path to a fasta reference file, this function will map sequences in the PAC using a 'backdoor' into the reanno workflow.

Value

Stacked list, where each object on the highest level contains: (Object 1) Reference name and sequence. (Object 2) Data.frame showing the mapping results of each query sequence that mapped to Object 1.

See Also

<https://github.com/OestLab/seqpac> for updates on the current package.

Other PAC analysis: `PAC_covplot()`, `PAC_deseq()`, `PAC_filter()`, `PAC_filtsep()`, `PAC_gtf()`, `PAC_jitter()`, `PAC_nbias()`, `PAC_norm()`, `PAC_pca()`, `PAC_pie()`, `PAC_saturation()`, `PAC_sizedist()`, `PAC_stackbar()`, `PAC_summary()`, `PAC_trna()`, `as.PAC()`, `filtsep_bin()`, `map_rangetype()`, `tRNA_class()`

Examples

```
#####
### Simple example of how to use PAC_mapper
# Note: More details, see vignette and manuals.)
# Also see: ?map_rangetype, ?tRNA_class or ?PAC_trna, ?PAC_covplot
# for more examples on how to use PAC_mapper.

## Load PAC-object, make summaries and extract rRNA and tRNA
load(system.file("extdata", "drosophila_sRNA_pac_filt_anno.Rdata",
                 package = "seqpac", mustWork = TRUE))

pac <- PAC_summary(pac, norm = "cpm", type = "means",
                  pheno_target=list("stage", unique(pheno(pac)$stage)))

pac_rRNA <- PAC_filter(pac, anno_target = list("Biotypes_mis0", "rRNA"))

## Give paths to a fasta reference (with or without bowtie index)
# (Here we use an rRNA/tRNA fasta included in seqpac)

ref_rRNA <- system.file("extdata/rRNA", "rRNA.fa",
                       package = "seqpac", mustWork = TRUE)

## Map using PAC-mapper
map_rRNA <- PAC_mapper(pac_rRNA, mismatches=0,
                      threads=1, input=ref_rRNA, override=TRUE)
```

PAC_nbias

Generates a nucleotide bias analysis from a PAC object

Description

PAC_nbias analyses nucleotide bias.

Usage

```
PAC_nbias(
  PAC,
  position = 1,
```

```

    norm = NULL,
    nucleotide_range = NULL,
    anno_target = NULL,
    pheno_target = NULL,
    summary_target = NULL,
    colors = NULL,
    ymax = NULL,
    data_only = FALSE
  )

```

Arguments

PAC	PAC-list object containing an Anno data.frame with sequences as row names and a count table with raw counts.
position	Integer indicating the nucleotide position from 3' to 5' position (default=1).
norm	Character indicating what type of data to be used. If type="counts" the plots will be based on the raw Counts. If type="cpm" the analysis will be done on cpm values returned from PAC_norm function and stored in the norm folder of the PAC-list object. The name of any other table in the norm(PAC) folder can also be used.
nucleotide_range	Integer vector indicating the sequence size range (default=c(min, max)).
anno_target	List with: 1st object being character vector of target column(s) in Anno, 2nd object being a character vector of the target biotypes(s) in the target column (1st object). (default=NULL)
pheno_target	List with: 1st object being character vector of target column(s) in Pheno, 2nd object being a character vector of the target group(s) in the target column (1st object). (default=NULL)
summary_target	List with: 1st object being character vector of target object in summary(PAC), 2nd object being a character vector of the target column(s) in the target summary object (1st object). (default=NULL)
colors	Character vector with RGB color codes to be parsed to ggplot2.
ymax	Integer that sets the maximum y for all plots (all plots gets the same y-axes). If ymax=NULL, then ggplot2 will automatically set ymax for each plot individually (different y-axes).
data_only	logical. If data_only=TRUE a data.frame a simple Anno object is returned with a Size and a Nucleotide bias column. As default, data_only=FALSE then graphs are returned in addition to data.

Details

Given a PAC object the function will attempt to extract the ratios of specific nucleotides at a given position in sequences in the Anno data.frame in relation to the sequence counts in Counts.

Value

A list of objects: 1st object (Histograms::Samples): Individual histograms showing the nucleotide ratios per sample over the specified range. 2nd object (Data::Samples): Data used to generate the plots.

See Also

<https://github.com/OestLab/seqpac> for updates on the current package.

Other PAC analysis: `PAC_covplot()`, `PAC_deseq()`, `PAC_filter()`, `PAC_filtsep()`, `PAC_gtf()`, `PAC_jitter()`, `PAC_mapper()`, `PAC_norm()`, `PAC_pca()`, `PAC_pie()`, `PAC_saturation()`, `PAC_sizedist()`, `PAC_stackbar()`, `PAC_summary()`, `PAC_trna()`, `as.PAC()`, `filtsep_bin()`, `map_rangetype()`, `tRNA_class()`

Examples

```
# Load a PAC-object
load(system.file("extdata", "drosophila_sRNA_pac_filt_anno.Rdata",
                package = "seqpac", mustWork = TRUE))
output_nbias <- PAC_nbias(pac)
cowplot::plot_grid(plotlist=output_nbias$Histograms)

# Only miRNA (Oops, heavy T-bias on 1st nt; are they piRNA?)
table(anno(pac)$Biotypes_mis0)
output_nbias <- PAC_nbias(pac, anno_target = list("Biotypes_mis0", "miRNA") )
cowplot::plot_grid(plotlist=output_nbias$Histograms)

# Switch to 10:th nt bias
output_nbias <- PAC_nbias(pac, position=10,
                        anno_target = list("Biotypes_mis0", "miRNA"))
cowplot::plot_grid(plotlist=output_nbias$Histograms)

# Summarized over group cpm means
pac_test <- PAC_summary(pac, norm = "cpm", type = "means",
                      pheno_target=list("stage"), merge_pac=TRUE)
output_nbias <- PAC_nbias(pac_test, summary_target = list("cpmMeans_stage") )
cowplot::plot_grid(plotlist=output_nbias$Histograms)
```

PAC_norm

Generates normalized values from a PAC object

Description

PAC_norm generates normalized values from a PAC object

Usage

```
PAC_norm(PAC, norm = "cpm", merge_pac = TRUE)
```

Arguments

PAC	PAC-list object containing an Anno data.frame with sequences as row names and a Count table with raw counts.
norm	Character indicating what type of normalization method that should be applied to the counts(PAC). If norm="cpm", counts per million reads is returned. Each sequence is then divided against the total number of counts in a given sample. If norm="vst", counts(PAC) will be imported into the varianceStabilizingTransformation function in the DESeq2-package with the options blind=TRUE and fitType="mean". If norm="rlog", counts(PAC) will instead be imported into the rlog function also available in DESeq2-package (options blind=TRUE and fitType="mean") for a log2 transformed version of vst, and is more robust to varying library sizes.
merge_pac	logical whether the normalized table should be returned and stored in the norm(PAC) 'folder' of the provided PAC object (TRUE) or be returned as a data frame.

Details

Using the counts in a PAC object to generate normalized values in a data.frame with the same rownames as in the original PAC object

Value

A normalized count table, or if pac_merge=TRUE, a PAC object with normalized counts table added to the norm folder (norm(PAC)).

See Also

<https://github.com/OestLab/seqpac> for updates on the current package.

Other PAC analysis: [PAC_covplot\(\)](#), [PAC_deseq\(\)](#), [PAC_filter\(\)](#), [PAC_filtsep\(\)](#), [PAC_gtf\(\)](#), [PAC_jitter\(\)](#), [PAC_mapper\(\)](#), [PAC_nbias\(\)](#), [PAC_pca\(\)](#), [PAC_pie\(\)](#), [PAC_saturation\(\)](#), [PAC_sizedist\(\)](#), [PAC_stackbar\(\)](#), [PAC_summary\(\)](#), [PAC_trna\(\)](#), [as.PAC\(\)](#), [filtsep_bin\(\)](#), [map_rangetype\(\)](#), [tRNA_class\(\)](#)

Examples

```
load(system.file("extdata", "drosophila_sRNA_pac_filt_anno.Rdata",
                 package = "seqpac", mustWork = TRUE))
pac_norm <- PAC_norm(pac, norm="cpm")
df_norm <- PAC_norm(pac, norm = "vst", merge_pac = FALSE)
```

PAC_pca

*Principle component analysis with scatterplots from PAC***Description**

PAC_pca PAC principle component analysis.

Usage

```
PAC_pca(
  PAC,
  norm = "counts",
  style = "pheno",
  graphs = TRUE,
  pheno_target = NULL,
  anno_target = NULL,
  labels = NULL
)
```

Arguments

PAC	PAC-list object.
norm	Character what data to plot. (default="counts")
style	Character indicating what type of pca and plot to be drawn. When style="pheno" then results will be drawn from a sample perspective, while if style="anno" then results will be presented from a sequence perspective. If style="both" then a bi-plot will be drawn, representing both pheno and anno features. Note, the 1st object (target column name) in the pheno_target or anno_target can be used to specifically highlight sample and sequence groups (see below).
graphs	Logical whether or not scatter plots should be plotted. (default=TRUE)
pheno_target	List with: 1st object being a character vector of target column(s) in Pheno, 2nd object being a character vector of the target group(s) in the target column (1st object). The 1st object also control group colors when style="pheno" or "both". (default=NULL)
anno_target	List with: 1st object being a character vector of target column(s) in Anno, 2nd object being a character vector of the target features(s) in the target column (1st object). The 1st object also control sequence feature colors when style="anno". (default=NULL)
labels	If labels="sample", then points will be labeled with the names rownames in pheno(PAC) or anno(PAC) depending on style="pheno" or style="anno", respectively. Point labels can also be manually provided as a character vector in the same length as the intended target. As default, labels=NULL where only point are plotted.

Details

Given a PAC object the function will perform a principle component analysis by calling the PCA function in the FactoMineR package, and then plot scatter plots with the fviz_pca functions of the factoextra package.

Value

A PCA list object generated by the PCA function in the FactoMineR package

See Also

<https://github.com/OestLab/seqpac> for updates on the current package.

Other PAC analysis: `PAC_covplot()`, `PAC_deseq()`, `PAC_filter()`, `PAC_filtsep()`, `PAC_gtf()`, `PAC_jitter()`, `PAC_mapper()`, `PAC_nbias()`, `PAC_norm()`, `PAC_pie()`, `PAC_saturation()`, `PAC_sizedist()`, `PAC_stackbar()`, `PAC_summary()`, `PAC_trna()`, `as.PAC()`, `filtsep_bin()`, `map_rangetype()`, `trna_class()`

Examples

```
# Load a PAC-object
load(system.file("extdata", "drosophila_sRNA_pac_filt_anno.Rdata",
                 package = "seqpac", mustWork = TRUE))

# Simple sample counts pca and scatterplots with no groupings:
pca_cnt <- PAC_pca(pac, norm="counts")

# Sample cpm pca and scatterplots with color groupings from
# pheno(PAC)$style column:
pca_cpm <- PAC_pca(pac, norm="cpm", style="pheno",
                  pheno_target=list("stage"))

# Same but with or without text labels:
pca_cpm_lab <- PAC_pca(pac, norm="cpm", style="pheno",
                      pheno_target=list("stage"), labels="samples")
pca_cpm_lab2 <- PAC_pca(pac, norm="cpm", style="pheno",
                       pheno_target=list("stage"),
                       labels=pheno(pac)$batch)

# Cpm pca with anno(PAC) sequence features instead of Pheno samples and
# restricted to read size 20-22:
pca_cpm_anno <- PAC_pca(pac, norm="cpm", style="anno",
                       anno_target=list("Size", 20:22))

# Cpm pca as biplot:
pca_cpm_bi <- PAC_pca(pac, norm="cpm", style="both",
                     pheno_target=list("stage"))

# Plot individual graphs
pca_cpm_bi$graphs[[1]]
pca_cpm_lab$graphs[[1]]
pca_cpm_anno$graphs[[3]]
```

```
# Extract pca output
pca_cpm_anno$pca
```

PAC_pie

Pie plot from PAC

Description

PAC_pie Pie chart of annotation features (anno_target).

Usage

```
PAC_pie(
  PAC,
  anno_target = NULL,
  pheno_target = NULL,
  colors = NULL,
  norm = "counts",
  labels = "all",
  no_anno = TRUE,
  summary_target = NULL,
  angle = -25
)
```

Arguments

PAC	PAC-list object.
anno_target	List with: 1st object being character vector of target column(s) in Anno, 2nd object being a character vector of the target biotype(s) in the target column (1st object). Important, the 2nd object is order sensitive, meaning that categories will appear in the same order in the pie. (default=NULL)
pheno_target	List with: 1st object being character vector of target column(s) in Pheno, 2nd object being a character vector of the target group(s) in the target column (1st object). Important, the 2nd object is order sensitive, meaning that categories will appear in the same order in the pie. (default=NULL)
colors	Character vector RGB color codes as generated by for example grDevices::colorRampPalette. If colors=NULL (default), colors will be generated using a default color palette. Important: In default mode, categories named "other" or no_anno" will automatically receive a grey color.
norm	Character vector defining what normalized data to plot. (default="counts")
labels	Character that sets what labels to plot in the actual pie besides the legend. If labels="all" (default), then the anno_target variables will be combined with percentage. If labels="percent" only percentages will be plotted with the pie. If labels="none", no labels will be present besides the legend.

no_anno	Logical whether PAC sequences without an annotation should be removed prior to plotting. Specifically, if no_anno=FALSE, then sequences annotated with "no_anno" in the anno_target column will be removed prior to plotting. When no_anno=TRUE (default), then all sequences will be included (unless excluded in the anno_target object).
summary_target	List with 1st object being a character of which summary to use, produced by PAC_summary. (default=NULL)
angle	Integer (positive or negative) that sets the rotation of the pie.

Details

Given a PAC object the function will summarize the counts into percent of a specified anno_target and plot a pie chart.

Value

A pie plot

See Also

<https://github.com/OestLab/seqpac> for updates on the current package.

Other PAC analysis: `PAC_covplot()`, `PAC_deseq()`, `PAC_filter()`, `PAC_filtsep()`, `PAC_gtf()`, `PAC_jitter()`, `PAC_mapper()`, `PAC_nbias()`, `PAC_norm()`, `PAC_pca()`, `PAC_saturation()`, `PAC_sizedist()`, `PAC_stackbar()`, `PAC_summary()`, `PAC_trna()`, `as.PAC()`, `filtsep_bin()`, `map_rangetype()`, `tRNA_class()`

Examples

```
#####
### Pie charts in seqpac
##-----

load(system.file("extdata", "drosophila_sRNA_pac_filt_anno.Rdata",
                 package = "seqpac", mustWork = TRUE))

# Choose an anno_target and plot all samples (default):
output_pie <- PAC_pie(pac, anno_target=list("Biotypes_mis0"))
output_pie[[1]]
output_pie[[6]]

# Defining a summary_target will show summarised data:
pac<-PAC_summary(pac)
PAC_pie(pac, anno_target=list("Biotypes_mis0"),
summary_target=list("countsMeans_All"))
# Rotate:
PAC_pie(pac, anno_target=list("Biotypes_mis0"),
summary_target=list("countsMeans_All"), angle=180)
```

```
# Make ordered pie charts of grand mean percent of all samples
ord_bio <- as.character(sort(unique(anno(pac)$Biotypes_mis3)),
                        unique(anno(pac)$Biotypes_mis0))
output_pie_1 <- PAC_pie(pac, anno_target=list("Biotypes_mis0", ord_bio),
                        labels = "percent", summary_target=list("countsMeans_All"))
output_pie_2 <- PAC_pie(pac, anno_target=list("Biotypes_mis3", ord_bio),
                        labels = "percent", summary_target=list("countsMeans_All"))
cowplot::plot_grid(plotlist=c(output_pie_1, output_pie_2), nrow=2,
                    scale = 1.0)

# Shortcut to remove no annotations ("no_anno") in the anno_target column
PAC_pie(pac, anno_target=list("Biotypes_mis3"), summary_target=list("countsMeans_All"),
        no_anno=TRUE)
PAC_pie(pac, anno_target=list("Biotypes_mis3"), summary_target=list("countsMeans_All"),
        no_anno=FALSE)
```

PAC_saturation

Filter a PAC object on sequence size and coverage

Description

PAC_saturation Performs an sequence diversity/saturation analysis on a PAC objects.

Usage

```
PAC_saturation(PAC, resample = 10, steps = 10, thresh = c(1, 10), threads = 1)
```

Arguments

PAC	PAC-list object containing a Counts data.frame with sequences as row names and samples as column names.
resample	Integer setting the number of permutations at each percentage step (default=10).
steps	Integer defining the number of percentage steps between 0-100 original dataset (default=10).
thresh	Integer vector containing mean count thresholds that will be targeted. Default is set to c(1,10), where each new occurrence reaching 1 count (≥ 1) and each new occurrence reaching 10 counts (≥ 10) will be analyzed.
threads	Number of cores to be used for performing the permutations.

Details

Given a PAC object the function will perform a sequence saturation analysis. This is done by down-sampling the original dataset by permutation at different percentages of the original dataset. The closer the curve at the original sequence depth (100 diversity of sequences for the original dataset. Approaching the plateau usually means that the sequencing depth of the library have sampled the full population of sequences available in the sample. Here we use an none-linear least square ([nls](#)) model with a self-starter for asymptotic regression (SSasympt) to describe the rate in which the library approaches the plateau.

Value

A list with ggplot2 graph objects: The 1:st graph shows saturation/diversity result at the 1:st threshold. The 2:nd graph shows saturation/diversity result at the 2:nd threshold, etc.

See Also

<https://github.com/OestLab/seqpac> for updates on the current package.

Other PAC analysis: `PAC_covplot()`, `PAC_deseq()`, `PAC_filter()`, `PAC_filtsep()`, `PAC_gtf()`, `PAC_jitter()`, `PAC_mapper()`, `PAC_nbias()`, `PAC_norm()`, `PAC_pca()`, `PAC_pie()`, `PAC_sizedist()`, `PAC_stackbar()`, `PAC_summary()`, `PAC_trna()`, `as.PAC()`, `filtsep_bin()`, `map_rangetype()`, `trna_class()`

Examples

```
# OBS! The example below is using already down-sampled data. Still, sequence
# diversity is rather saturated on >=1 occurrence. meaning that most sequences
# in the samples has been caught. Nonetheless, sequences reaching >=2
# occurrences have not plateaued.
```

```
load(system.file("extdata", "drosophila_sRNA_pac_filt_anno.Rdata",
                 package = "seqpac", mustWork = TRUE))
```

```
plot_lst <- PAC_saturation(pac, resample=10, steps=10,
                          thresh=c(1,2), threads=1)
names(plot_lst)
cowplot::plot_grid(plotlist=plot_lst)
```

PAC_sizedist

Generates size distribution plots from a PAC object

Description

PAC_sizedist plotting size distribution with bar charts, allowing for visualization of sequence classes and summaries.

Usage

```
PAC_sizedist(
  PAC,
  norm = "counts",
  nucleotide_range = NULL,
  anno_target,
  pheno_target = NULL,
  summary_target = NULL,
  colors = NULL,
  ymax = NULL
)
```

Arguments

PAC	PAC-list object containing an Anno data.frame with sequences as row names and a count table with raw counts.
norm	Character indicating what type of data to be used. If type="counts" the plots will be based on the raw Counts. If type="cpm" the analysis will be done on cpm values returned from PAC_norm function and stored in the norm folder of the PAC-list object. The name of any other table in the norm(PAC) folder can also be used.
nucleotide_range	Integer vector giving the nucleotide_range in sequence lengths (default=c(min, max)).
anno_target	List with: 1st object being character vector of target column(s) in Anno, 2nd object being a character vector of the target biotypes(s) in the target column (1st object). (default=NULL)
pheno_target	List with: 1st object being character vector of target column(s) in Pheno, 2nd object being a character vector of the target group(s) in the target column (1st object). (default=NULL)
summary_target	List with: 1st object being character target object in summary(PAC), 2nd object being a character vector of the target columns(s) in the target object (1st object). (default=NULL)
colors	Character vector with RGB color codes to be parsed to ggplot2.
ymax	Integer that sets the maximum y for all plots (all plots gets the same y-axes). If ymax=NULL, then ggplot2 will automatically set ymax for each plot individually (different y-axes).

Details

Given a PAC object the function will attempt to order sequences by their size (number of nucleotides) and visualize the contribution of specific classes of sequences (e.g. sRNA classes) at each size point.

Value

A list of objects: 1st object (Histograms::Samples): Individual histograms showing the nucleotide ratios per sample over the specified range. 2nd object (Data::Samples): Data used to generate the plots. 3rd object (Stacked_bars::Groups): Stacked bars showing the mean ratios of each nucleotide per group over the specified range. 4th object (Error_bars::Groups): Error bar plots with mean ratio of each nucleotide per group over the specified range.

See Also

<https://github.com/OestLab/seqpac> for updates on the current package.

Other PAC analysis: `PAC_covplot()`, `PAC_deseq()`, `PAC_filter()`, `PAC_filtsep()`, `PAC_gtf()`, `PAC_jitter()`, `PAC_mapper()`, `PAC_nbias()`, `PAC_norm()`, `PAC_pca()`, `PAC_pie()`, `PAC_saturation()`, `PAC_stackbar()`, `PAC_summary()`, `PAC_trna()`, `as.PAC()`, `filtsep_bin()`, `map_rangetype()`, `trna_class()`

Examples

```
#####
### Stacked bars in seqpac
##-----
load(system.file("extdata", "drosophila_sRNA_pac_filt_anno.Rdata",
                 package = "seqpac", mustWork = TRUE))

PAC_filt <- PAC_norm(pac, norm="cpm")
PAC_filt <- PAC_summary(PAC=PAC_filt, norm = "cpm",
                       type = "means", pheno_target=list("stage"))

ord <- c("no_anno", "other", "miRNA", "tRNA", "rRNA", "snoRNA", "lncRNA")

sizedist_plots <- PAC_sizedist(PAC_filt,
                              anno_target=list("Biotypes_mis0", ord),
                              summary_target=list("cpmMeans_stage"))
cowplot::plot_grid(plotlist=sizedist_plots[[1]], nrow = 3, ncol = 1)

sizedist_plots <- PAC_sizedist(PAC_filt, norm="counts",
                              anno_target=list("Biotypes_mis0", ord),
                              pheno_target=list("batch", "Batch1"))
cowplot::plot_grid(plotlist=sizedist_plots[[1]], nrow = 3, ncol = 1)
```

PAC_stackbar

Plot stackbars from PAC

Description

PAC_stackbar Generates a graph that stack classes up to 100

Usage

```
PAC_stackbar(
  PAC,
  anno_target = NULL,
  pheno_target = NULL,
  summary_target = NULL,
  color = NULL,
  width = 1,
  no_anno = TRUE,
  total = TRUE,
  norm = "counts",
  style = "percent"
)
```

Arguments

PAC	PAC-list object.
anno_target	List with: 1st object being character vector of target column(s) in Anno, 2nd object being a character vector of the target biotype(s) in the target column (1st object). Important, the 2nd object is order sensitive, meaning that categories will appear in the same order in the stacked bar graph. (default=NULL)
pheno_target	List with: 1st object being character vector of target column(s) in Pheno, 2nd object being a character vector of the target group(s) in the target column (1st object). Important, the 2nd object is order sensitive, meaning that categories will appear in the same order in the stacked bar graph. (default=NULL)
summary_target	List with: 1st object being character vector of target column(s) in Summary. This is produced by PAC_summary. If no summary_target is defined, samples will be plotted individually. (default=NULL)
color	Character vector with rgb color hex codes in the same length as the number of biotypes. For example see: https://www.coolgenerator.com/rgb-color-generator . color=NULL will generate the default color scheme.
width	Integer adjusting the width of the bars (default=1). Works best with few or singular bars.
no_anno	Logical. If TRUE sequences with no annotations will be plotted, while FALSE will skip sequences with 'no_anno' in the column defined by anno_target (default=TRUE). Note that you can always use PAC_filter to remove anno_targets from PAC prior to plotting.
total	Logical, whether the total counts should be added at the bottom of each graph (default=TRUE).
norm	Character vector defining what data to base analysis on, e.g "counts" for raw counts (default), "cpm" for normalized data or any other column in norm section of PAC object.
style	Character vector defining how data is to be presented in stack bar, where default is "percent", showing the percentage of the anno_target of all reads. Other option is "total", where the total amount of counts/normalized reads are stacked in one stack per anno_target.

Details

Given a PAC object the function will attempt to extract group information from Pheno, class information from Anno, or summarized data to generate a stacked (percent or total counts) bar.

Value

A stacked bar plot generated by ggplot2

See Also

<https://github.com/OestLab/seqpac> for updates on the current package.

Other PAC analysis: [PAC_covplot\(\)](#), [PAC_deseq\(\)](#), [PAC_filter\(\)](#), [PAC_filtsep\(\)](#), [PAC_gtf\(\)](#), [PAC_jitter\(\)](#), [PAC_mapper\(\)](#), [PAC_nbias\(\)](#), [PAC_norm\(\)](#), [PAC_pca\(\)](#), [PAC_pie\(\)](#), [PAC_saturation\(\)](#),

```
PAC_sizedist(), PAC_summary(), PAC_trna(), as.PAC(), filtsep_bin(), map_rangetype(),
tRNA_class()
```

Examples

```
load(system.file("extdata", "drosophila_sRNA_pac_filt_anno.Rdata",
                 package = "seqpac", mustWork = TRUE))

#####
### Stacked bars in seqpac
##-----

# Choose an anno_target and plot samples
PAC_stackbar(pac, anno_target=list("Biotypes_mis0"))

# 'no_anno' and 'other' will always end on top not matter the order
ord_bio <- as.character(sort(unique(anno(pac)$Biotypes_mis3)))
p1 <- PAC_stackbar(pac, anno_target=list("Biotypes_mis0", ord_bio))
p2 <- PAC_stackbar(pac, anno_target=list("Biotypes_mis0", rev(ord_bio)))
cowplot::plot_grid(plotlist=list(p1, p2))
# (Hint: if you don't want them to appear on top, rename them)

# Reorder samples by pheno_targets
PAC_stackbar(pac, pheno_target=list("batch"),
             anno_target=list("Biotypes_mis0"))

# Summarized over pheno_target
# (as default PAC_stackbar orders by pheno_target but plots all samples,
# unless a summary_target is defined)

pac <- PAC_summary(pac, pheno_target=list("stage"))
PAC_stackbar(pac, anno_target=list("Biotypes_mis0"),
             summary_target=list("countsMeans_stage"))

# Summarized over a grand mean of all samples

pac <- PAC_summary(pac)
PAC_stackbar(pac, anno_target=list("Biotypes_mis0"),
             summary_target=list("countsMeans_All"))
```

PAC_summary

Summarizes PAC objects

Description

PAC_summary summarizes data stored in a PAC object.

Usage

```
PAC_summary(
  PAC,
  norm = "counts",
  type = "means",
  pheno_target = NULL,
  rev = FALSE,
  merge_pac = TRUE
)
```

Arguments

PAC	PAC object containing a Pheno data.frame with samples as row names and a Counts table with raw counts. Optionally, the PAC object may contain normalized counts tables, saved in the norm list ('folder'). Such normalized table can be generated using the PAC_norm function.
norm	Character indicating what type of data to be used. If 'counts', the raw counts in Counts will be used (default). Given any other value, the function will search for the value as a name on a data.frame stored in the normalized list-folder.
type	Character indicating what type of summary to be applied to the data. The function currently supports: type="means" # Group means type="sd" # Group standard deviation type="se" # Group standard error of the mean type="log2FC" # Group log2 fold changes against other groups type="log2FCgrand" # Group log2 fold changes against a grand mean type="percentgrand" # Group log2 fold changes against a grand mean
pheno_target	List with: 1st object being a character vector of target column in Pheno 2nd object being a character vector of the target group(s) in the target Pheno column (1st object).
rev	Logical whether pairwise comparisons (e.g. log2FC) should be reversed (default=FALSE).
merge_pac	Logical whether simplified annotation vector should automatically be added to the Anno object of the input PAC list object (default=TRUE). If merge_pac=FALSE a data.frame is returned.

Details

Given a PAC object this function summarize data in counts(PAC) or in the norm 'folder' according to a grouping column in pheno(PAC).

Value

A PAC object with a pheno_summary folder containing the summarized data in a data.frame. The data.frame will be named according to the pheno_target, type and norm input.

See Also

<https://github.com/OestLab/seqpac> for updates on the current package.

Other PAC analysis: `PAC_covplot()`, `PAC_deseq()`, `PAC_filter()`, `PAC_filtsep()`, `PAC_gtf()`, `PAC_jitter()`, `PAC_mapper()`, `PAC_nbias()`, `PAC_norm()`, `PAC_pca()`, `PAC_pie()`, `PAC_saturation()`, `PAC_sizedist()`, `PAC_stackbar()`, `PAC_trna()`, `as.PAC()`, `filtsep_bin()`, `map_rangetype()`, `tRNA_class()`

Examples

```
load(system.file("extdata", "drosophila_sRNA_pac_filt_anno.Rdata",
  package = "seqpac", mustWork = TRUE))

PAC_check(pac) # TRUE

# Easy to generate simple group summaries
pac <- PAC_summary(pac, norm = "cpm",
  type = "means", pheno_target=list("stage"))
pac <- PAC_summary(pac, norm = "cpm",
  type = "se", pheno_target=list("stage"))
pac <- PAC_summary(pac, norm = "cpm",
  type = "log2FC", pheno_target=list("stage"))

names(summary(pac)) # Names of individual summaries
head(summary(pac)$cpmMeans_stage) # View individual summaries
summary(pac) # View merge summaries
df <- as.data.frame(tibble::as_tibble(summary(pac))) # Merge multiple summaries
head(df)

# If a pheno_target is left out, a mean of all samples will be returned:
load(system.file("extdata", "drosophila_sRNA_pac_filt_anno.Rdata",
  package = "seqpac", mustWork = TRUE))
pac <- PAC_summary(pac, norm = "cpm", type = "mean")
pac <- PAC_summary(pac, norm = "cpm", type = "percentgrand")
names(summary(pac))
summary(pac)
```

PAC_trna

tsRNA analysis of PAC object

Description

Analyzing and plotting tsRNAs between groups.

Usage

```
PAC_trna(
  PAC,
  norm = "cpm",
  filter = 100,
  join = FALSE,
```

```

top = 15,
log2fc = FALSE,
pheno_target = NULL,
anno_target_1 = NULL,
ymax = NULL,
anno_target_2 = NULL,
paired = FALSE,
paired_IDs = NULL
)

```

Arguments

PAC	PAC-list object. The Anno object needs to contain at least two tsRNA classification columns, typically isotype (e.g. LysCTT, GlyGCC) and rangetype (5'tsRNA, 5'tsRNA-halves, i'-tsRNA, 3'tsRNA, 3'tsRNA-halves). These can be generated by first mapping sequences to tRNA reference using PAC_mapper and extract isotypes from tRNA names, and then obtain rangetype classifications using map_rangetype .
norm	Character indicating what type of data in PAC to be used as input. If norm="raw" the raw counts in Counts will be used. Given any other character string, the function will search for the string as a name on a dataframe stored in the PAC\$norm list-folder (created for example by PAC_rpm), (default="rpm")
filter	Integer specifying the minimum coverage value (specified by norm) for a sequence to be included in the analysis. If filter=10 and norm="rpm", only sequences that have at least 10 rpm in 100 will be included. (default=NULL).
join	Logical, whether separate expression tables of anno_target_1 and anno_target_2 should be generated for each group in pheno_target (FALSE), or if pheno_target groups should be joined (TRUE).
top	Integer, specifying the number of the most highly expressed classifications in anno_target_1 that should be reported in the graphs.
log2fc	Logical, whether log2 fold change point-bars (independent pheno_target groups) or errorbars (paired pheno_target groups) should be reported.
pheno_target	List with: 1st object being a character vector of the target column name in Pheno, and the 2nd object being a character vector specifying the names of two target groups in the target column (1st object). Note, that this function will only work using two groups from the pheno_target column, like this: pheno_target=list(<Pheno_column_name>, c(<group_1>, <group_2>)). If there are more groups than two in the target column, unspecified groups will not be analyzed.
anno_target_1	List with: 1st object being a character of the target column name in Anno, 2nd object being a character vector of the target classifications in the 1st target Anno column. The anno_target_1 is typically used for isotype classification in tsRNA analysis (example: LysCTT, GlyGGC etc). Specifying only the column name, will automatically include all classifications in the anno_target column.
ymax	Integer that sets the maximum y for all mean plots (all plots gets the same y-axes). If ymax=NULL (default), then ggplot2 will automatically set ymax for each plot individually (different y-axes).


```

ss_file <- system.file("extdata/trna2", "tRNA2.ss",
                      package = "seqpac", mustWork = TRUE)

##-----##
## Create a map object using PAC_mapper ##
map_object <- PAC_mapper(pac_trna, input=trna_ref,
                        N_up = "NNN", N_down = "NNN",
                        mismatches=0, threads=2,
                        report_string=TRUE, override = TRUE)
# Warning: override = TRUE, will remove everything in temporary output path.
# Note: bowtie indexes are not needed for PAC_mapper.

##-----##
## Coverage plots of tRNA using PAC_covplot ##

# Single tRNA targeting a summary data.frame
PAC_covplot(pac_trna, map=map_object, summary_target= list("cpmMeans_stage"),
            map_target="tRNA-Ala-AGC-1-1")

# Find tRNAs with many fragments and then plot
n_tRFs <- unlist(lapply(map_object, function(x){nrow(x[[2]])}))
selct <- (names(map_object)[n_tRFs>1])[c(1, 16, 25, 43)]
cov_plt <- PAC_covplot(pac_trna, map=map_object,
                      summary_target= list("cpmMeans_stage"),
                      map_target=selct)
cowplot::plot_grid(plotlist=cov_plt, nrow=2, ncol=2)

##-----##
## Classify tRNA fragment with map_rangetype ##

# Classify according to loop structure using ss file provided with seqpac
map_object_ss <- map_rangetype(map_object, type="ss", ss=ss_file,
                              min_loop_width=4)
# Note 1: You may download your own ss file at for example GtRNAdb
# Note 2: The ss file must match the reference used in creating the map_object

# Classify instead according to nucleotide position or percent intervals
map_object_rng <- map_rangetype(map_object, type="nucleotides",
                              intervals=list(start=1:5, end=95:100))

map_object_prc <- map_rangetype(map_object, type="percent",
                              intervals=list(start=1:5, mid=45:50,
                                             end=95:100))

# Compare the output (same fragment different classifications)
map_object_ss[[2]]
map_object_prc[[2]]
map_object_rng[[2]]

```

```
##-----##
## Classify tRFs with tRNA_class ##
# Warning: this function currently only works using map objects
# created using ss files with specific names for loop1/2/3 columns:
colnames(map_object_ss[[2]][[2]])

# Important: We added three "Ns" in PAC_mapper (above). If, terminal
# classification should correspond to the original tRNA sequences we need to
# account for these Ns when defining what is a terminal fragment. Here, we set
# "terminal=5", which means that PAC sequences will receive 5'/3'
# classification if they map to the first two or last two nucleotides of the
# original full-length tRNAs (2+NNN =5).

pac_trna <- tRNA_class(pac_trna, map_object_ss, terminal=5)

##-----##
## Plot some results with PAC_trna ##

# Here is one example of tRNA visualization using our PAC_trna function:
trna_result <- PAC_trna(pac_trna, norm="cpm", filter = NULL,
                        join = TRUE, top = 15, log2fc = TRUE,
                        pheno_target = list("stage", c("Stage1", "Stage5")),
                        anno_target_1 = list("type"),
                        anno_target_2 = list("class"))

cowplot::plot_grid(trna_result$plots$Expression_Anno_1$Grand_means,
                  trna_result$plots$Log2FC_Anno_1,
                  trna_result$plots$Percent_bars$Grand_means,
                  nrow=1, ncol=3)

# Note: There are no 3' fragments in our test data.

# By setting join = FALSE you will get group means instead of grand means:
trna_result <- PAC_trna(pac_trna, norm="cpm", filter = NULL,
                        join = FALSE, top = 15, log2fc = TRUE,
                        pheno_target = list("stage", c("Stage1", "Stage3")),
                        anno_target_1 = list("type"),
                        anno_target_2 = list("class"))

cowplot::plot_grid(trna_result$plots$Expression_Anno_1$Stage1,
                  trna_result$plots$Expression_Anno_1$Stage3,
                  trna_result$plots$Log2FC_Anno_1,
                  trna_result$plots$Percent_bars$Stage1,
                  trna_result$plots$Percent_bars$Stage3,
                  nrow=1, ncol=5)

##-----##
## Clean up temp folder ##
# (Sometimes needed for automated examples)

closeAllConnections()
fls_temp <- tempdir()
fls_temp <- list.files(fls_temp, recursive=TRUE,
```

```

                                full.names = TRUE)
suppressWarnings(file.remove(fls_temp))

```

pheno	<i>pheno(PAC)</i>
-------	-------------------

Description

Access sample information in a PAC-object

Usage

```

pheno(object)

## S4 method for signature 'PAC'
pheno(object)

## S4 replacement method for signature 'PAC,data.frame'
pheno(object) <- value

```

Arguments

object	A S4 PAC-object containing Pheno, Anno and Counts tables. Can be generated by the 'PAC generation' family of functions, for example see <code>?make_counts</code> .
value	Data.frame having the same row names as column names in Counts (sample IDs).

Value

The Pheno data.frame table from a S4 PAC-object
Updated S4 PAC-object.

See Also

Other PAC methods: [anno\(\)](#), [counts\(\)](#), [names,PAC-method](#), [norm\(\)](#), [show.PAC\(\)](#), [summary\(\)](#)

Examples

```

load(system.file("extdata", "drosophila_sRNA_pac_filt_anno.Rdata",
                 package = "seqpac", mustWork = TRUE))

# extra fuctionality with s4 PAC-object:
names(pac)
length(pac)
nrow(pac)
ncol(pac)
rownames(pac)

```

```

colnames(pac)
pheno(pac)
head(anno(pac))
head(counts(pac))
head(norm(pac)$cpm)

```

reanno-class	<i>S4 object for reanno output</i>
--------------	------------------------------------

Description

Holds the imported information from mapping using the `map_reanno` function. All information are held in tibble class (tbl/tbl_df) tables from the tibble package.

Usage

```
reanno(Overview, Full_anno)
```

Arguments

Overview	A tibble data.frame with summarized results from mapping using the <code>map_reanno</code> function that has been imported into R using the <code>make_reanno</code> function. Rows represents sequences from the original PAC-object.
Full_anno	A multi-level list with tibble data.frames that contains all that was imported by <code>make_reanno</code>

Value

Contains two slots: 1. Overview: A table holding a summary of the mapping. 2. Full_anno: Lists of tables holding the full imports per mismatch cycle (mis0, mis1 etc) and reference (mi0-ref1, mis0-ref2, mis1-ref1, mis1-ref2 etc).

Examples

```

##### Create an reanno object

## First load a PAC- object

load(system.file("extdata", "drosophila_sRNA_pac_filt_anno.Rdata",
                 package = "seqpac", mustWork = TRUE))

## Then specify paths to fasta references
# If you are having problem see the vignette small RNA guide for more info.

trna_file <- system.file("extdata/trna", "tRNA.fa",
                        package = "seqpac", mustWork = TRUE)
trna_dir <- dirname(trna_file)

```

```

if(!sum(stringr::str_count(list.files(trna_dir), ".ebwt")) ==6){
  Rbowtie::bowtie_build(trna_file,
                        outdir=trna_dir,
                        prefix="tRNA", force=TRUE)
}

ref_paths <- list(trna= trna_file)

## Add output path of your choice.
# Here we use the R temporary folder depending on platform
output <- paste0(tempdir(), "/seqpac/test")

## Make sure it is empty (otherwise you will be prompted for a question)
out_fls <- list.files(output, recursive=TRUE)
closeAllConnections()
suppressWarnings(file.remove(paste(output, out_fls, sep="/")))

## Then map your PAC-object against the fasta references
map_reanno(pac, input=ref_paths, output=output,
            type="internal", mismatches=0, import="biotype",
            threads=2, keep_temp=FALSE, override=TRUE)

## Then import and generate a reanno-object of the temporary bowtie-files
reanno <- make_reanno(output, PAC=pac, mis_fasta_check = TRUE)

## Now make some search terms against reference names to create shorter names
# Theses can be used to create factors in downstream analysis
# One search hit (regular expressions) gives one new short name
bio_search <- list(trna =c("_tRNA", "mt:tRNA"))

## You can merge directly with your PAC-object by adding your original
# PAC-object, that you used with map_reanno, to merge_pac option.
pac <- add_reanno(reanno, bio_search=bio_search,
                  type="biotype", bio_perfect=FALSE,
                  mismatches = 0, merge_pac=pac)

## Turn your S3 list to an S4 reanno-object
isS4(reanno)
names(reanno)
table(overview(reanno)$trna)

reanno_s3 <- as(reanno, "list")
isS4(reanno_s3)

# Turns S3 reanno object into a S4
reanno_s4 <- as.reanno(reanno_s3)
isS4(reanno_s4)

```

rownames,PAC-method	<i>sequence names</i>
---------------------	-----------------------

Description

sequence names
sequences in reanno

Usage

```
## S4 method for signature 'PAC'  
rownames(x)  
  
## S4 method for signature 'reanno'  
rownames(x)
```

Arguments

x A S4 object of class PAC.

Value

The sequences (row names) held by the Counts/Anno tables of a PAC-object.
Sequences in the reanno-object.

Examples

```
load(system.file("extdata", "drosophila_sRNA_pac_filt_anno.Rdata",  
                  package = "seqpac", mustWork = TRUE))  
  
# extra fuctionality with s4 PAC-object:  
names(pac)  
length(pac)  
nrow(pac)  
ncol(pac)  
rownames(pac)  
colnames(pac)  
pheno(pac)  
head(anno(pac))  
head(counts(pac))  
head(norm(pac)$cpm)
```

`show.PAC``show(PAC)`

Description

Overview (show method) of a S4 PAC

Usage

```
show.PAC(object)
```

Arguments

`object` A S4 object of class PAC.

Value

Simple statistical overview of a PAC-object.

See Also

Other PAC methods: [anno\(\)](#), [counts\(\)](#), [names](#), [PAC-method](#), [norm\(\)](#), [pheno\(\)](#), [summary\(\)](#)

Examples

```
load(system.file("extdata", "drosophila_sRNA_pac_filt_anno.Rdata",
                 package = "seqpac", mustWork = TRUE))

# extra fuctionality with s4 PAC-object:
names(pac)
length(pac)
nrow(pac)
ncol(pac)
rownames(pac)
colnames(pac)
pheno(pac)
head(anno(pac))
head(counts(pac))
head(norm(pac)$cpm)
```

simplify_reanno	<i>Generates one-hit hierarchial biotype annotation</i>
-----------------	---

Description

simplify_reanno adds hierarchical classification to PAC.

Usage

```
simplify_reanno(
  input,
  hierarchy,
  mismatches = 2,
  bio_name = "Biotypes",
  merge_pac = FALSE,
  target_columns = NULL
)
```

Arguments

input	Data.frame with biotype/mismatch matrix generated by add_reanno. A PAC-list object can also be provided here where the Anno data.frame will be used as input. Important, PAC objects need biotype columns generated by add_reanno .
hierarchy	List with character vectors. The vectors search terms constructed as regular expressions that will be parsed to grepl() and searched for in the mismatch biotype reannotation matrix (colnames: mis0_bio, mis1_bio, mis2_bio, mis3_bio). Important <i>hierarchy</i> is order sensitive. For example if <i>hierarchy</i> is defined as list(rRNA="rRNA", miRNA="miRNA", piRNA="piRNA") a sequence annotated with 0 mismatches for all three biotypes will only be reported as rRNA since this biotype was put first in the list. Sequences annotated to neither of the biotypes listed in <i>hierarchy</i> , that is still annotated to something will be assigned as "other". Sequences without an annotation down to the specified mismatch level (mismatches will be assigned as "no_anno").
mismatches	Integer indicating the number of allowed mismatches. Note that this value can never be larger than the maximum number of mismatches used in the reannotation workflow (default=0).
bio_name	Character naming the final biotype column (default="Biotype")
merge_pac	Logical whether the simplified annotation column should automatically be added to the Anno object if a PAC list object were given as input (default=FALSE).
target_columns	Character vector naming the target columns for the reannotation matrix. When target_columns=NULL(default), the function assumes that one (and only one) reannotation matrix for biotypes generatated by add_reanno is present in anno(PAC) (colnames = mis0_bio, mis1_bio, mis2_bio etc). With target_columns two alternative reannotation matrices can be discriminated.

Details

This function deals with and can be used to explore the effect of multimapping. Given a biotype reannotation matrix (colnames = mis0_bio, mis1_bio, mis2_bio etc) created by `add_reanno` and usually added to the `Anno` data.frame of a PAC object, the function will generate a one-hit biotype vector according to a user-defined biotype hierarchy.

Value

Character vector with single best-hit biotypes with fewest mismatches.

See Also

<http://bowtie-bio.sourceforge.net/index.shtml> for information about Bowtie <https://github.com/OestLab/seqpac> for updates on the current package.

Other PAC reannotation: `PAC_annotate()`, `add_reanno()`, `as_reanno()`, `import_reanno()`, `make_conv()`, `make_reanno()`, `map_reanno()`

Examples

```
##' #####
##### hierarchical classification using simplify_reanno
closeAllConnections()
#####
##### Create an reanno object

### First, if you haven't already generated Bowtie indexes for the included
# fasta references you need to do so. If you are having problem see the small
# RNA guide (vignette) for more info.

## tRNA:
trna_file <- system.file("extdata/trna", "trna.fa",
                        package = "seqpac", mustWork = TRUE)
trna_dir <- dirname(trna_file)

if(!sum(stringr::str_count(list.files(trna_dir), ".ebwt")) == 6){
  Rbowtie::bowtie_build(trna_file,
                      outdir=trna_dir,
                      prefix="trna", force=TRUE)
}

## rRNA:
rrna_file <- system.file("extdata/rrna", "rrna.fa",
                        package = "seqpac", mustWork = TRUE)
rrna_dir <- dirname(rrna_file)

if(!sum(stringr::str_count(list.files(rrna_dir), ".ebwt")) == 6){
  Rbowtie::bowtie_build(rrna_file,
                      outdir=rrna_dir,
                      prefix="rrna", force=TRUE)
}
```

```
## Then load a PAC-object and remove previous mapping from anno:
load(system.file("extdata", "drosophila_sRNA_pac_filt_anno.Rdata",
                 package = "seqpac", mustWork = TRUE))
anno(pac) <- anno(pac)[,1, drop = FALSE]

## You may add an output path of your choice, but here we use a temp folder:
output <- paste0(tempdir(), "/seqpac/test")
ref_paths <- list(trna= trna_file, rrna= rrna_file)

## Then map the PAC-object against the fasta references.
# Warning: if you use your own data, you may want to use override=FALSE, to avoid
# deleting previous mapping by mistake.

map_reanno(pac, input=ref_paths, output=output,
            type="internal", mismatches=1, import="biotype",
            threads=2, keep_temp=FALSE, override=TRUE)

## Then import and generate a reanno-object of the temporary bowtie-files
reanno_biotype <- make_reanno(output, PAC=pac, mis_fasta_check = TRUE)

## Now make some search terms against reference names to create shorter names
# Theses can be used to create factors in downstream analysis
# One search hit (regular expressions) gives one new short name

bio_search <- list(
  rrna=c("5S", "5.8S", "12S", "16S", "18S", "28S", "pre_S45"),
  trna =c("_tRNA", "mt:tRNA"))

## You can merge directly with your PAC-object by adding your original
# PAC-object, that you used with map_reanno, to merge_pac option.

pac <- add_reanno(reanno_biotype, bio_search=bio_search,
                  type="biotype", bio_perfect=FALSE,
                  mismatches = 1, merge_pac=pac)

## Hierarchical classification with simplify_reanno
table(anno(pac)$mis0_bio)
table(anno(pac)$mis1_bio)

# Setup hierarchy(rRNA prioritized before tRNA)
hierarchy_1 <- list(rrna="rrna_",
                    trna="trna_")

# Setup hierarchy(specifies the hierarchy among rRNA)
hierarchy_2 <- list(r18S="rrna_18S",
                    r28S="rrna_28S",
                    r5.8S="rrna_5.8S",
                    r5S="rrna_5S",
                    pre45S="pre_S45",
                    other_rrna="rrna_other")
```

```
# No mismatches for hierarchy_1:
test1 <- simplify_reanno(input=pac, hierarchy=hierarchy_1, mismatches=0,
                          bio_name="Biotypes_mis0", merge_pac=FALSE)

# Up to 2 mismatches for rRNA hierarchy_2
test2 <- simplify_reanno(input=pac, hierarchy=hierarchy_2, mismatches=1,
                          bio_name="rRNA_biotypes_mis2", merge_pac=FALSE)

table(test1$Biotypes_mis0)
table(test2$rRNA_biotypes_mis2)

# You can add the new column to your PAC-object using merge_pac=TRUE:
pac <- simplify_reanno(input=pac, hierarchy=hierarchy_2, mismatches=1,
                       bio_name="rRNA_biotypes_mis1", merge_pac=TRUE)

table(anno(pac)$rRNA_biotypes_mis1)
```

summary

summary(PAC)

Description

Access the summary list with summarized tables in S4 PAC

Usage

```
summary(object)

## S4 method for signature 'PAC'
summary(object)

## S4 replacement method for signature 'PAC,list'
summary(object) <- value
```

Arguments

object	A S4 PAC-object containing a summary list ('folder') with normalized data.frames. Can be generated by the PAC_summary function.
value	List of data.frames having the same row names as row names in Anno (sequences).

Value

The list of summary tables (data.frames) from a PAC-object.
Updated S4 PAC-object.

See Also

Other PAC methods: [anno\(\)](#), [counts\(\)](#), [names](#), [PAC-method](#), [norm\(\)](#), [pheno\(\)](#), [show.PAC\(\)](#)

Examples

```
load(system.file("extdata", "drosophila_sRNA_pac_filt_anno.Rdata",
                 package = "seqpac", mustWork = TRUE))

# extra fuctionality with s4 PAC-object:
names(pac)
length(pac)
nrow(pac)
ncol(pac)
rownames(pac)
colnames(pac)
pheno(pac)
head(anno(pac))
head(counts(pac))
head(norm(pac)$cpm)
```

tRNA_class	<i>Classify tsRNA/tRFs from PAC object</i>
------------	--

Description

Classifying tRFs/tsRNA into 5'-tRF/halves, 3'-tRFs/halves, i'-tRFs/halves, as well as acceptor and decoder isotypes.

Usage

```
tRNA_class(PAC, map, terminal = 5)
```

Arguments

PAC	PAC-list object.
map	Map object with loop annotations generated by PAC_mapper followed by map_rangetype functions. Note, only full length reference tRNAs annotated with 3 loops in the ss file (see: map_rangetype) are used in the classifications. Explanation: sometimes tRNAscan-SE scans results in ss files with incomplete secondary structures with either more or fewer loops than the D, T and anticodon loops found in canonical tRNAs. Such tRNAs will be removed from the analysis.
terminal	Integer setting the terminal tolerance. Example, when terminal is set to 5 (default) sequenced reads starting within the 5 first nucleotides of the full-length tRNA reference will be classified as 5'. If it instead ends within the 5 last nucleotides of the full-length tRNA reference it will be classified as 3'. If a read neither starts nor ends within the terminal threshold it will be classified as i'

(internal). Note, for tsRNA/tRF classification we recommend a threshold of terminal=2 (not default). Advisably, however, `PAC_mapper` runs with `N_up = "NNN"` and `N_down = "NNN"` when mapping tsRNA/tRFs. Thus, the default threshold of 5 is used when full-length tRNAs has been extended with NNN up- and downstream to compensate for possible fragments from mature tRNAs (3+2). Always double check the exact fragment using `PAC_covplot`.

Details

Given a map object with range types generated by `PAC_mapper` followed by `map_rangetype` functions, sequences in a PAC object are classified according the terminals (5'/3'/i'), anticodon loop (half/tRF), and isotype (decoder/acceptor) of the full length tRNA.

Value

Merged PAC object with an extended Anno table containing the tRF/tsRNA classifications.

See Also

<https://github.com/OestLab/seqpac> for updates on the current package.

Other PAC analysis: `PAC_covplot()`, `PAC_deseq()`, `PAC_filter()`, `PAC_filtsep()`, `PAC_gtf()`, `PAC_jitter()`, `PAC_mapper()`, `PAC_nbias()`, `PAC_norm()`, `PAC_pca()`, `PAC_pie()`, `PAC_saturation()`, `PAC_sizedist()`, `PAC_stackbar()`, `PAC_summary()`, `PAC_trna()`, `as.PAC()`, `filtsep_bin()`, `map_rangetype()`

Examples

```
# Important: See ?PAC_trna on how to do tRNA analysis using Seqpac

#'#####

### tRNA classification in seqpac
# (More details see vignette and manuals.)

##-----##
## Setup environment for testing ##

# First create an annotation blank PAC with group means
load(system.file("extdata", "drosophila_sRNA_pac_filt_anno.Rdata",
  package = "seqpac", mustWork = TRUE))
anno(pac) <- anno(pac)[,1, drop=FALSE]
pac_trna <- PAC_summary(pac, norm = "cpm", type = "means",
  pheno_target=list("stage"), merge_pac = TRUE)

# Then load paths to example trna ref and ss files
trna_ref <- system.file("extdata/trna2", "tRNA2.fa",
  package = "seqpac", mustWork = TRUE)

ss_file <- system.file("extdata/trna2", "tRNA2.ss",
  package = "seqpac", mustWork = TRUE)
```

```
##-----##
## Create a map object using PAC_mapper ##
map_object <- PAC_mapper(pac_trna, input=trna_ref,
                        N_up = "NNN", N_down = "NNN",
                        mismatches=0, threads=2,
                        report_string=TRUE, override = TRUE)
# Warning: override = TRUE, will remove everything in temporary output path.
# Note: bowtie indexes are not needed for PAC_mapper.

##-----##
## Classify tRNA fragment with map_rangetype ##

# Classify according to loop structure using ss file provided with seqpac
map_object_ss <- map_rangetype(map_object, type="ss", ss=ss_file,
                              min_loop_width=4)
# Note 1: You may download your own ss file at for example GtRNAdb
# Note 2: The ss file must match the reference used in creating the map_object

map_object_ss[[1]]
```

Index

* PAC analysis

- as.PAC, 9
- filtsep_bin, 14
- map_rangetype, 39
- PAC_covplot, 59
- PAC_deseq, 63
- PAC_filter, 65
- PAC_filtsep, 68
- PAC_gtf, 69
- PAC_jitter, 72
- PAC_mapper, 74
- PAC_nbias, 76
- PAC_norm, 78
- PAC_pca, 80
- PAC_pie, 82
- PAC_saturation, 84
- PAC_sizedist, 85
- PAC_stackbar, 87
- PAC_summary, 89
- PAC_trna, 91
- tRNA_class, 105

* PAC generation

- make_counts, 22
- make_cutadapt, 26
- make_PAC, 28
- make_pheno, 30
- make_trim, 35
- merge_lanes, 46
- PAC_check, 58
- PAC_create, 62

* PAC methods

- anno, 8
- counts, 13
- names, PAC-method, 47
- norm, 50
- pheno, 96
- show.PAC, 100
- summary, 104

* PAC reannotation

- add_reanno, 3
- as.reanno, 10
- import_reanno, 17
- make_conv, 20
- make_reanno, 32
- map_reanno, 41
- PAC_annotate, 56
- simplify_reanno, 101

* reanno methods

- full, 15
- overview, 52

- add_reanno, 3, 10, 18, 21, 33, 43, 57, 101, 102
- anno, 8, 13, 48, 51, 96, 100, 105
- anno, (anno), 8
- anno, PAC-method (anno), 8
- anno<- (anno), 8
- anno<- ,PAC, data.frame-method (anno), 8
- as.PAC, 9, 14, 40, 60, 65, 67, 69, 71, 73, 76, 78, 79, 81, 83, 85, 86, 89, 91, 93, 106
- as.reanno, 5, 10, 18, 21, 33, 43, 57, 102

- colnames, PAC-method, 12
- counts, 8, 13, 48, 51, 96, 100, 105
- counts, (counts), 13
- counts, PAC-method (counts), 13
- counts<- (counts), 13
- counts<- ,PAC, data.frame-method (counts), 13

- DESeq, 64

- filtsep_bin, 9, 14, 40, 60, 65, 67–69, 71, 73, 76, 78, 79, 81, 83, 85, 86, 89, 91, 93, 106
- full, 15, 53
- full, (full), 15
- full, reanno-method (full), 15
- full<- (full), 15
- full<- ,reanno, list-method (full), 15

- grepl, [4](#)
- import_reanno, [4](#), [5](#), [10](#), [17](#), [18](#), [21](#), [33](#), [43](#), [57](#), [102](#)
- lapply, [40](#)
- length, PAC-method, [19](#)
- length, reanno-method
(length, PAC-method), [19](#)
- make_conv, [5](#), [10](#), [18](#), [20](#), [33](#), [43](#), [57](#), [102](#)
- make_counts, [22](#), [27](#), [28](#), [30](#), [38](#), [46](#), [59](#), [62](#)
- make_cutadapt, [23](#), [25](#), [26](#), [28](#), [30](#), [38](#), [46](#), [59](#), [62](#)
- make_PAC, [25](#), [27](#), [28](#), [30](#), [38](#), [46](#), [59](#), [62](#)
- make_pheno, [25](#), [27](#), [28](#), [30](#), [38](#), [46](#), [59](#), [62](#)
- make_reanno, [4](#), [5](#), [10](#), [18](#), [21](#), [32](#), [43](#), [57](#), [97](#), [102](#)
- make_trim, [23](#), [25](#), [27](#), [28](#), [30](#), [35](#), [46](#), [59](#), [62](#)
- map_rangetype, [9](#), [14](#), [39](#), [60](#), [65](#), [67](#), [69](#), [71](#), [73](#), [76](#), [78](#), [79](#), [81](#), [83](#), [85](#), [86](#), [89](#), [91–93](#), [105](#), [106](#)
- map_reanno, [4](#), [5](#), [10](#), [18](#), [21](#), [32](#), [33](#), [39](#), [41](#), [57](#), [97](#), [102](#)
- merge_lanes, [25](#), [27](#), [28](#), [30](#), [38](#), [46](#), [59](#), [62](#)
- names, PAC-method, [47](#)
- names, reanno-method (names, PAC-method), [47](#)
- ncol, PAC-method, [49](#)
- nls, [84](#)
- norm, [8](#), [13](#), [48](#), [50](#), [96](#), [100](#), [105](#)
- norm, (norm), [50](#)
- norm, PAC-method (norm), [50](#)
- norm<- (norm), [50](#)
- norm<-, PAC, list-method (norm), [50](#)
- nrow, PAC-method, [51](#)
- nrow, reanno-method (nrow, PAC-method), [51](#)
- overview, [16](#), [52](#)
- overview, (overview), [52](#)
- overview, reanno-method (overview), [52](#)
- overview<- (overview), [52](#)
- overview<-, reanno, data.frame-method
(overview), [52](#)
- PAC (PAC-class), [54](#)
- PAC-class, [54](#)
- PAC_analyze, [55](#)
- PAC_annotate, [5](#), [10](#), [18](#), [21](#), [33](#), [43](#), [56](#), [102](#)
- PAC_check, [25](#), [27](#), [28](#), [30](#), [38](#), [46](#), [58](#), [62](#)
- PAC_covplot, [9](#), [14](#), [40](#), [59](#), [65](#), [67](#), [69](#), [71](#), [73](#), [76](#), [78](#), [79](#), [81](#), [83](#), [85](#), [86](#), [88](#), [91](#), [93](#), [106](#)
- PAC_create, [25](#), [27](#), [28](#), [30](#), [38](#), [46](#), [59](#), [62](#)
- PAC_deseq, [9](#), [14](#), [40](#), [60](#), [63](#), [67](#), [69](#), [71](#), [73](#), [76](#), [78](#), [79](#), [81](#), [83](#), [85](#), [86](#), [88](#), [91](#), [93](#), [106](#)
- PAC_filter, [9](#), [14](#), [40](#), [60](#), [65](#), [65](#), [69](#), [71](#), [73](#), [76](#), [78](#), [79](#), [81](#), [83](#), [85](#), [86](#), [88](#), [91](#), [93](#), [106](#)
- PAC_filtsep, [9](#), [14](#), [40](#), [60](#), [65](#), [67](#), [68](#), [71](#), [73](#), [76](#), [78](#), [79](#), [81](#), [83](#), [85](#), [86](#), [88](#), [91](#), [93](#), [106](#)
- PAC_gtf, [9](#), [14](#), [40](#), [60](#), [65](#), [67](#), [69](#), [69](#), [73](#), [76](#), [78](#), [79](#), [81](#), [83](#), [85](#), [86](#), [88](#), [91](#), [93](#), [106](#)
- PAC_jitter, [9](#), [14](#), [40](#), [60](#), [65](#), [67](#), [69](#), [71](#), [72](#), [76](#), [78](#), [79](#), [81](#), [83](#), [85](#), [86](#), [88](#), [91](#), [93](#), [106](#)
- PAC_mapper, [9](#), [14](#), [39](#), [40](#), [60](#), [65](#), [67](#), [69](#), [71](#), [73](#), [74](#), [78](#), [79](#), [81](#), [83](#), [85](#), [86](#), [88](#), [91–93](#), [105](#), [106](#)
- PAC_nbias, [9](#), [14](#), [40](#), [60](#), [65](#), [67](#), [69](#), [71](#), [73](#), [76](#), [76](#), [79](#), [81](#), [83](#), [85](#), [86](#), [88](#), [91](#), [93](#), [106](#)
- PAC_norm, [9](#), [14](#), [40](#), [60](#), [65](#), [67](#), [69](#), [71](#), [73](#), [76](#), [78](#), [78](#), [81](#), [83](#), [85](#), [86](#), [88](#), [90](#), [91](#), [93](#), [106](#)
- PAC_pca, [9](#), [14](#), [40](#), [60](#), [65](#), [67](#), [69](#), [71](#), [73](#), [76](#), [78](#), [79](#), [80](#), [83](#), [85](#), [86](#), [88](#), [91](#), [93](#), [106](#)
- PAC_pie, [9](#), [14](#), [40](#), [60](#), [65](#), [67](#), [69](#), [71](#), [73](#), [76](#), [78](#), [79](#), [81](#), [82](#), [85](#), [86](#), [88](#), [91](#), [93](#), [106](#)
- PAC_saturation, [9](#), [14](#), [40](#), [60](#), [65](#), [67](#), [69](#), [71](#), [73](#), [76](#), [78](#), [79](#), [81](#), [83](#), [84](#), [86](#), [88](#), [91](#), [93](#), [106](#)
- PAC_sizedist, [9](#), [14](#), [40](#), [60](#), [65](#), [67](#), [69](#), [71](#), [73](#), [76](#), [78](#), [79](#), [81](#), [83](#), [85](#), [85](#), [89](#), [91](#), [93](#), [106](#)
- PAC_stackbar, [9](#), [14](#), [40](#), [60](#), [65](#), [67](#), [69](#), [71](#), [73](#), [76](#), [78](#), [79](#), [81](#), [83](#), [85](#), [86](#), [87](#), [91](#), [93](#), [106](#)
- PAC_summary, [9](#), [14](#), [40](#), [60](#), [65](#), [67](#), [69](#), [71](#), [73](#), [76](#), [78](#), [79](#), [81](#), [83](#), [85](#), [86](#), [89](#), [89](#), [93](#), [106](#)
- PAC_trna, [9](#), [14](#), [40](#), [60](#), [65](#), [67](#), [69](#), [71](#), [73](#), [76](#), [78](#), [79](#), [81](#), [83](#), [85](#), [86](#), [89](#), [91](#), [91](#), [106](#)
- pheno, [8](#), [13](#), [48](#), [51](#), [96](#), [100](#), [105](#)
- pheno, (pheno), [96](#)

pheno,PAC-method (pheno), [96](#)
pheno<- (pheno), [96](#)
pheno<- ,PAC,data.frame-method (pheno),
 [96](#)

reanno (reanno-class), [97](#)
reanno-class, [97](#)
rlog, [79](#)
rownames,PAC-method, [99](#)
rownames,reanno-method
 (rownames,PAC-method), [99](#)

show.PAC, [8](#), [13](#), [48](#), [51](#), [96](#), [100](#), [105](#)
simplify_reanno, [4](#), [5](#), [10](#), [18](#), [21](#), [33](#), [43](#), [57](#),
 [101](#)
summary, [8](#), [13](#), [48](#), [51](#), [96](#), [100](#), [104](#)
summary, (summary), [104](#)
summary,PAC-method (summary), [104](#)
summary<- (summary), [104](#)
summary<- ,PAC,list-method (summary), [104](#)

tRNA_class, [9](#), [14](#), [40](#), [60](#), [65](#), [67](#), [69](#), [71](#), [73](#),
 [76](#), [78](#), [79](#), [81](#), [83](#), [85](#), [86](#), [89](#), [91](#), [93](#),
 [105](#)

varianceStabilizingTransformation, [79](#)