Package 'QuasR'

November 12, 2026

Type Package

Title Quantify and Annotate Short Reads in R

Version 1.52.0

Date 2025-06-23

Depends R (>= 4.4), parallel, GenomicRanges, Rbowtie

Imports methods, grDevices, graphics, utils, stats, tools, BiocGenerics, S4Vectors, IRanges, Biobase, Biostrings, BSgenome, Rsamtools (>= 2.13.1), GenomicFeatures, txdbmaker, ShortRead, BiocParallel, Seqinfo, rtracklayer, GenomicFiles, AnnotationDbi

Suggests Gviz, BiocStyle, GenomeInfoDbData, GenomicAlignments, Rhisat2, knitr, rmarkdown, covr, testthat

LinkingTo Rhtslib (>= 1.99.1)

Encoding UTF-8

SystemRequirements GNU make

Description This package provides a framework for the quantification and analysis of Short Reads. It covers a complete workflow starting from raw sequence reads, over creation of alignments and quality control plots, to the quantification of genomic regions of interest. Read alignments are either generated through Rbowtie (data from DNA/ChIP/ATAC/Bis-seq experiments) or Rhisat2 (data from RNA-seq experiments that require spliced alignments), or can be provided in the form of bam files.

URL https://bioconductor.org/packages/QuasR

License GPL-2

biocViews Genetics, Preprocessing, Sequencing, ChIPSeq, RNASeq, MethylSeq, Coverage, Alignment, QualityControl, ImmunoOncology

Archs x64

VignetteBuilder knitr

RoxygenNote 7.3.2

BugReports https://github.com/fmicompbio/QuasR/issues

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

git_branch RELEASE_3_23

git_last_commit 162f60c

git_last_commit_date 2026-04-28

Repository Bioconductor 3.23

Date/Publication 2026-11-11

Author Anita Lerch [aut],
Adam Alexander Thil SMITH [aut] (ORCID:
<https://orcid.org/0000-0003-4593-3042>),
Charlotte Soneson [aut] (ORCID:
<https://orcid.org/0000-0003-3833-2169>),
Dimos Gaidatzis [aut],
Michael Stadler [aut, cre] (ORCID:
<https://orcid.org/0000-0002-2269-4934>)

Maintainer Michael Stadler <michael.stadler@fmi.ch>

Contents

QuasR-package 2
.get_alnstats_for_bam 3
alignmentStats 4
preprocessReads 5
qAlign 8
qCount 12
qExportWig 17
qMeth 20
qProfile 24
qProject-class 27
qProjectUpdate 29
qQCReport 30
worker_message 33

Index **34**

`QuasR-package` *Quantify and Annotate Short Reads in R*

Description

This package provides a pipeline for the quantification and analysis of short reads.

Details

See packageDescription('QuasR') for package details.

Author(s)

Anita Lerch, Charlotte Soneson, Dimos Gaidatzis and Michael Stadler

See Also

qAlign, qCount, qProfile, qMeth, qQCReport

Examples

```
## Not run:
# see qCount, qMeth and qProfile manual pages for examples
example(qCount)
example(qMeth)
example(qProfile)

## End(Not run)
```

.get_alnstats_for_bam *Internal function to get seqlengths, mapped and unmapped counts for a bam file*

Description

Internal function to get seqlengths, mapped and unmapped counts for a bam file

Usage

```
.get_alnstats_for_bam(bf)
```

Arguments

bf	Character scalar with the path and filename to a bam file (index is assumed to be already existing)

Value

a named numeric vector with elements 'seqlength', 'mapped' and 'unmapped'

`alignmentStats` *Get statistics on alignments*

Description

Get statistics on alignments from bam file or `qProject` object.

Usage

```
alignmentStats(x, collapseBySample = TRUE)
```

Arguments

`x`	the source of alignment bam files, one of: • a `character` vector with bam files • a `qProject` object
`collapseBySample`	If `TRUE` and `x` is a `qProject` object, sum counts for bam files with identical sample names.

Details

Internally, `alignmentStats` queries the bam index files similar to 'idxstats' from samtools. Please note that this does not discriminate for example between primary and secondary alignments. If you need more statistics, see for example `quickBamFlagSummary` from package **Rsamtools**.

If `x` is a `qProject` object, the auxiliary bam files will not contain any unmapped reads, and the corresponding unmapped counts are calculated by subtracting auxiliary mapped counts from the total reads. The latter correspond to the unmapped counts from the corresponding genome bam files.

Value

A `matrix` with one row per bam file and three columns ("seqlength", "mapped" and "unmapped").

Author(s)

Anita Lerch, Dimos Gaidatzis and Michael Stadler

See Also

`qProject`, `quickBamFlagSummary` from package **Rsamtools**

Examples

```
## Not run:
# see qProject manual page for an example
example(qProject)

## End(Not run)
```

`preprocessReads` *Preprocess Short Reads*

Description

Truncate sequences, remove parts matching to adapters and filter out low quality or low complexity sequences from (compressed) 'fasta' or 'fastq' files.

Usage

```
preprocessReads(
  filename,
  outputFilename = NULL,
  filenameMate = NULL,
  outputFilenameMate = NULL,
  truncateStartBases = NULL,
  truncateEndBases = NULL,
  Lpattern = "",
  Rpattern = "",
  max.Lmismatch = rep(0:2, c(6, 3, 100)),
  max.Rmismatch = rep(0:2, c(6, 3, 100)),
  with.Lindels = FALSE,
  with.Rindels = FALSE,
  minLength = 14L,
  nBases = 2L,
  complexity = NULL,
  nrec = 1000000L,
  clObj = NULL
)
```

Arguments

`filename`	the name(s) of the input sequence file(s).
`outputFilename`	the name(s) of the output sequence file(s).
`filenameMate`	for paired-end experiments, the name(s) of the input sequence file(s) containing the second read (mate) of each pair.
`outputFilenameMate`	for paired-end experiments, the name(s) of the output sequence file(s) containing the second read (mate) of each pair.

`truncateStartBases`
: integer(1): the number of bases to be truncated (removed) from the beginning of each sequence.

`truncateEndBases`
: integer(1): the number of bases to be truncated (removed) from the end of each sequence.

`Lpattern`
: character(1): the left (5'-end) adapter sequence.

`Rpattern`
: character(1): the right (3'-end) adapter sequence.

`max.Lmismatch`
: mismatch tolerance when searching for matches of `Lpattern` (see 'Details').

`max.Rmismatch`
: mismatch tolerance when searching for matches of `Rpattern` (see 'Details').

`with.Lindels`
: if `TRUE`, indels are allowed in the alignments of the suffixes of `Lpattern` with the subject, at its beginning (see 'Details').

`with.Rindels`
: same as `with.Lindels` but for alignments of the prefixes of `Rpattern` with the subject, at its end (see 'Details').

`minLength`
: integer(1): the minimal allowed sequence length.

`nBases`
: integer(1): the maximal number of Ns allowed per sequence.

`complexity`
: `NULL` (default) or numeric(1): If not `NULL`, the minimal sequence complexity, as a fraction of the average complexity in the human genome (~3.9bits). For example, `complexity = 0.5` will filter out sequences that do not have at least half the complexity of the human genome. See 'Details' on how the complexity is calculated.

`nrec`
: integer(1): the number of sequence records to read at a time.

`clObj`
: a cluster object to be used for parallel processing of multiple files (see 'Details').

Details

Sequence files can be in fasta or fastq format, and can be compressed by either gzip, bzip2 or xz (extensions .gz, .bz2 or .xz). Multiple files can be processed by a single call to `preprocessReads`; in that case all sequence file vectors must have identical lengths.

`nrec` can be used to limit the memory usage when processing large input files. `preprocessReads` iteratively loads chunks of `nrec` sequences from the input until all data been processed.

Sequence pairs from paired-end experiments can be processed by specifying pairs of input and output files (`filenameMate` and `outputFilenameMate` arguments). In that case, it is assumed that pairs appear in the same order in the two input files, and only pairs in which both reads pass all filtering criteria are written to the output files, maintaining the consistent ordering.

If output files are compressed, the processed sequences are first written to temporary files (created in the same directory as the final output file), and the output files are generated at the end by compressing the temporary files.

For the trimming of left and/or right flanking sequences (adapters) from sequence reads, the `trimLRPatterns` function from package **Biostrings** is used, and the arguments `Lpattern`, `Rpattern`, `max.Lmismatch`, `max.Rmismatch`, `with.Lindels` and `with.Rindels` are used in the call to `trimLRPatterns`. `Lfixed` and `Rfixed` arguments of `trimLRPatterns` are set to `TRUE`, thus only fixed patterns (without IUPAC codes for ambigous bases) can be used. Currently, trimming of adapters is only supported for single read experiments.

Sequence complexity (H) is calculated based on the dinucleotide composition using the formula (Shannon entropy):

$$H = -\sum_i f_i \log_2 f_i,$$

where f_i is the fraction of dinucleotide i from all dinucleotides in the sequence. Sequence reads that fulfill the condition $H/H_r \geq c$ are retained (not filtered out), where $H_r = 3.908$ is the reference complexity in bits obtained from the human genome, and c is the value given to the argument `complexity`.

If an object that inherits from class `cluster` is provided to the `clObj` argument, for example an object returned by `makeCluster` from package **parallel**, multiple files will be processed in parallel using `clusterMap` from package **parallel**.

Value

A matrix with summary statistics on the processed sequences, containing:

- One column per input file (or pair of input files for paired-end experiments).
- The number of sequences or sequence pairs in rows:

  `totalSequences` - the total number in the input

  `matchTo5pAdaptor` - matching to `Lpattern`

  `matchTo3pAdaptor` - matching to `Rpattern`

  `tooShort` - shorter than `minLength`

  `tooManyN` - more than `nBases` Ns

  `lowComplexity` - relative complexity below `complexity`

  `totalPassed` - the number of sequences/sequence pairs that pass all filtering criteria and were written to the output file(s).

Author(s)

Anita Lerch, Dimos Gaidatzis and Michael Stadler

See Also

`trimLRPatterns` from package **Biostrings**, `makeCluster` from package **parallel**

Examples

```
# sample files
infiles <- system.file(package="QuasR", "extdata",
                       c("rna_1_1.fq.bz2","rna_1_2.fq.bz2"))
outfiles <- paste(tempfile(pattern=c("output_1_","output_2_")),".fastq",sep="")
# single read example
preprocessReads(infiles, outfiles, nBases=0, complexity=0.6)
unlink(outfiles)
# paired-end example
preprocessReads(filename=infiles[1],
                outputFilename=outfiles[1],
                filenameMate=infiles[2],
```

```
                outputFilenameMate=outfiles[2],
                nBases=0, complexity=0.6)
unlink(outfiles)
```

---

qAlign *Align reads*

Description

Create read alignments against reference genome and optional auxiliary targets if not yet existing. If necessary, also build target indices for the aligner.

Usage

```
qAlign(
  sampleFile,
  genome,
  auxiliaryFile = NULL,
  aligner = "Rbowtie",
  maxHits = 1,
  paired = NULL,
  splicedAlignment = FALSE,
  snpFile = NULL,
  bisulfite = "no",
  alignmentParameter = NULL,
  projectName = "qProject",
  alignmentsDir = NULL,
  lib.loc = NULL,
  cacheDir = NULL,
  clObj = NULL,
  checkOnly = FALSE,
  geneAnnotation = NULL
)
```

Arguments

sampleFile	The name of a text file listing input sequence files and sample names (see 'Details').
genome	The reference genome for primary alignments, one of: • a string referring to a "BSgenome" package (e.g. ""BSgenome.Hsapiens.UCSC.hg19""), which will be downloaded automatically from Bioconductor if not present • the name of a fasta sequence file containing one or several sequences (chromosomes) to be used as a reference. The aligner index will be created when necessary and stored in a default location (see 'Details').
auxiliaryFile	The name of a text file listing sequences to be used as additional targets for alignment of reads not mapping to the reference genome (see 'Details').

`aligner` selects the aligner program to be used for aligning the reads. Currently, only "Rbowtie" and "Rhisat2" are supported, which are R wrapper packages for 'bowtie' / 'SpliceMap' and 'hisat2', respectively (see `Rbowtie-package` and `Rhisat2-package` packages).

`maxHits` sets the maximal number of allowed mapping positions per read (default: 1). If a read produces more than `maxHits` alignments, no alignments will be reported for it. In case of a multi-mapping read, a single alignment is randomly selected.

`paired` defines the type of paired-end library and can be set to one of `no` (single read experiment, default), `fr` (fw/rev), `ff` (fw/fw) or `rf` (rev/fw).

`splicedAlignment`
If `TRUE`, reads will be aligned using a spliced aligner, depending on the value of `aligner` described above:

`aligner="Rhisat2"` : This is the recommended setting for spliced alignments and will use hisat2 from the `Rhisat2-package`. See also the `geneAnnotation` argument below for providing known exon-exon junctions.

`aligner="Rbowtie"` : This is not recommended and only available for legacy reasons. It will use SpliceMap to produce spliced alignments (without using a database of known exon-exon junctions). Compared to the alternative alignment modes (non-spliced or spliced using `Rhisat2` as aligner), this alignment mode is about ten-fold slower and also less sensitive. Furthermore, SpliceMap can only be used for reads with a minimal length of 50nt; SpliceMap ignores reads that are shorter, and these reads will not be contained in the BAM file, neither as mapped or unmapped reads.

`snpFile` The name of a text file listing single nucleotide polymorphisms to be used for allele-specific alignment and quantification (see 'Details').

`bisulfite` For bisulfite-converted samples (Bis-seq), the type of bisulfite library ("dir" for directional libraries, "undir" for undirectional libraries).

`alignmentParameter`
An optional string containing command line parameters to be used for the aligner, to overrule the default alignment parameters used by `QuasR`. Please use with caution; some alignment parameters may break assumptions made by `QuasR`. Default parameters are listed in 'Details'.

`projectName` An optional name for the alignment project.

`alignmentsDir` The directory to be used for storing alignments (bam files). If set to `NULL` (default), bam files will be generated at the location of the input sequence files.

`lib.loc` can be used to change the default library path of R. The library path is used by `QuasR` to store aligner index packages created from `BSgenome` reference genomes.

`cacheDir` specifies the location to store (potentially huge) temporary files. If set to `NULL` (default), the temporary directory of the current R session as returned by `tempdir()` will be used.

`clObj` A cluster object, created by the package **parallel**, to enable parallel processing and speed up the alignment process.

checkOnly — If TRUE, prevents the automatic creation of alignments or aligner indices. This allows to quickly check for missing alignment files without starting the potentially long process of their creation. In the case of missing alignments or indices, an exception is thrown.

geneAnnotation — Only used if aligner is "Rhisat2". The path to either a gtf file or a sqlite database generated by exporting a TxDb object. This file is used to generate a splice site file for Rhisat2, that will be used to guide the spliced alignment. Please note that if using an sqlite database file, do not use the one contained in the installed package folder of a TxDb package. QuasR (through Rhisat2) creates additional files in the folder which would interfere with the loading of the TxDb package.

Details

Before generating new alignments, qAlign looks for previously generated alignments as well as for an aligner index. If no aligner index exists, it will be automatically created and stored in the same directory as the provided fasta file, or as an R package in the case of a BSgenome reference. The name of this R package will be the same as the BSgenome package name, with an additional suffix from the aligner (e.g. BSgenome.Hsapiens.UCSC.hg19.Rbowtie). The generated bam files contain both aligned und unaligned reads. For paired-end samples, by default no alignments will be reported for read pairs where only one of the reads could be aligned.

sampleFile is a tab-delimited text file listing all the input sequences to be included in a given analysis. The file has either two (single-end) or three columns (paired-end). The first row contains the column names, and additional rows contain relative or absolute path and name of input sequence file(s), as well as the according sample name. Three input file formats are supported (fastq, fasta and bam). All input files in one sampleFile need to be in the same format, and are recognized by their extension (.fq, .fastq, .fa, .fasta, .fna, .bam), in raw or compressed form (e.g. .fastq.gz). If bam files are provided, then no alignments are generated by qAlign, and the alignments contained in the bam files will be used instead.

The column names in sampleFile have to match to the ones in the examples below, for a single-read experiment:

FileName	SampleName
chip_1_1.fq.bz2	Sample1
chip_2_1.fq.bz2	Sample2

and for a paired-end experiment:

FileName1	FileName2	SampleName
rna_1_1.fq.bz2	rna_1_2.fq.bz2	Sample1
rna_2_1.fq.bz2	rna_2_2.fq.bz2	Sample2

The "SampleName" column is the human-readable name for each sample that will be used as sample labels. Multiple sequence files may be associated to the same sample name, which instructs QuasR to combine those files.

auxiliaryFile is a tab-delimited text file listing one or several additional target sequence files in fasta format. Reads that do not map against the reference genome will be aligned against each of

these target sequence files. The first row contains the column names which have to match to the ones in the example below:

FileName	AuxName
NC_001422.1.fa	phiX174

`snpFile` is a tab-delimited text file without a header and contains four columns with chromosome name, position, reference allele and alternative allele, as in the example below:

chr1	8596	G	A
chr1	18443	G	A
chr1	18981	C	T
chr1	19341	G	A

The reference and alternative alleles will be injected into the reference genome, resulting in two separate genomes. All reads will be aligned separately to both of these genomes, and the alignments will be combined, only retaining the best alignment for each read. In the final alignment, each read will be marked with a tag that classifies it into reference (`R`), alternative (`A`) or unknown (`U`), if the reads maps equally well to both genomes.

If `bisulfite` is set to "dir" or "undir", reads will be C-to-T converted and aligned to a similarly converted genome.

If `alignmentParameter` is `NULL` (recommended), `qAlign` will select default parameters that are suitable for the experiment type. Please note that for bisulfite or allele-specific experiments, each read is aligned multiple times, and resulting alignments need to be combined. This requires special settings for the alignment parameters that are not recommended to be changed. For 'simple' experiments (neither bisulfite, allele-specific, nor spliced), alignments are generated using the parameters `-m maxHits --best --strata`. This will align reads with up to "maxHits" best hits in the genome and selects one of them randomly.

Value

A `qProject` object.

Author(s)

Anita Lerch, Dimos Gaidatzis, Charlotte Soneson and Michael Stadler

See Also

`qProject`, `makeCluster` from package **parallel**, `Rbowtie-package` package, `Rhisat2-package` package

Examples

```
## Not run:
# see qCount, qMeth and qProfile manual pages for examples
example(qCount)
example(qMeth)
example(qProfile)
```

```
## End(Not run)
```

---

qCount *Quantify alignments*

Description

Quantify alignments from sequencing data.

Usage

```
qCount(
  proj,
  query,
  reportLevel = c(NULL, "gene", "exon", "promoter", "junction"),
  selectReadPosition = c("start", "end"),
  shift = 0L,
  orientation = c("any", "same", "opposite"),
  useRead = c("any", "first", "last"),
  auxiliaryName = NULL,
  mask = NULL,
  collapseBySample = TRUE,
  includeSpliced = TRUE,
  includeSecondary = TRUE,
  mapqMin = 0L,
  mapqMax = 255L,
  absIsizeMin = NULL,
  absIsizeMax = NULL,
  maxInsertSize = 500L,
  clObj = NULL
)
```

Arguments

proj	A qProject object representing a sequencing experiment as returned by qAlign
query	An object of type GRanges, GRangesList or TxDb with the regions to be quantified. The type of query will determine the mode of quantification (see 'Details'). For reportLevel="junction", query is ignored and can also be NULL.
reportLevel	Level of quantification (query of type TxDb or NULL), one of gene **(default)** : one value per gene exon : one value per exon promoter : one value per promoter junction : one count per detected exon-exon junction (query will be ignored in this case)

`selectReadPosition`
: defines the part of the alignment that has to be contained within a query region to produce an overlap (see 'Details'). Possible values are:

  `start` **(default)** : start of the alignment

  `end` : end of the alignment

`shift`
: controls the shifting alignments towards their 3'-end before quantification. `shift` can be one of:

  - an "integer" vector of the same length as the number of alignment files
  - a single "integer" value
  - the character string `"halfInsert"` (only available for paired-end experiments)

  The default of `0` will not shift any alignments.

`orientation`
: sets the required orientation of the alignments relative to the query region in order to be counted, one of:

  `any` **(default)** : count alignment on the same and opposite strand

  `same` : count only alignment on the same strand

  `opposite` : count only alignment on the opposite strand

`useRead`
: For paired-end experiments, selects the read mate whose alignments should be counted, one of:

  `any` **(default)** : count all alignments

  `first` : count only alignments from the first read

  `last` : count only alignments from the last read

`auxiliaryName`
: Which bam files to use in an experiments with auxiliary alignments (see 'Details').

`mask`
: If not `NULL`, a `GRanges` object with reference regions to be masked, i.e. excluded from the quantification, such as unmappable or highly repetitive regions (see 'Details').

`collapseBySample`
: If `TRUE` (the default), sum alignment counts from bam files with the same sample name.

`includeSpliced`
: If `TRUE` (the default), include spliced alignments when counting. A spliced alignment is defined as an alignment with a gap in the read of at least 60 bases.

`includeSecondary`
: If `TRUE` (the default), include alignments with the secondary bit (0x0100) set in the `FLAG` when counting.

`mapqMin`
: Minimal mapping quality of alignments to be included when counting (mapping quality must be greater than or equal to `mapqMin`). Valid values are between 0 and 255. The default (0) will include all alignments.

`mapqMax`
: Maximal mapping quality of alignments to be included when counting (mapping quality must be less than or equal to `mapqMax`). Valid values are between 0 and 255. The default (255) will include all alignments.

`absIsizeMin`
: For paired-end experiments, minimal absolute insert size (TLEN field in SAM Spec v1.4) of alignments to be included when counting. Valid values are greater than 0 or `NULL` (default), which will not apply any minimum insert size filtering.

absIsizeMax	For paired-end experiments, maximal absolute insert size (TLEN field in SAM Spec v1.4) of alignments to be included when counting. Valid values are greater than 0 or NULL (default), which will not apply any maximum insert size filtering.
maxInsertSize	Maximal fragment size of the paired-end experiment. This parameter is used if shift="halfInsert" and will ensure that query regions are made wide enough to emcompass all alignment pairs whose mid falls into the query region. The default value is 500 bases.
clObj	A cluster object to be used for parallel processing (see 'Details').

Details

qCount is used to count alignments in each sample from a qProject object. The features to be quantified, together with the mode of quantification, are specified by the query argument, which is one of:

GRanges : Overlapping alignments are counted separately for each coordinate region. If multiple regions have identical names, their counts will be summed, counting each alignment only once even if it overlaps more than one of these regions. Alignments may be counted more than once if they overlap multiple regions that have different names. This mode is for example used to quantify ChIP-seq alignments in promoter regions, or gene expression levels in an RNA-seq experiment (using a query with exon regions named by gene).

GRangesList : Alignments are counted and summed for each list element in query if they overlap with any of the regions contained in the list element. The order of the list elements defines a hierarchy for quantification: Alignment will only be counted for the first element (the one with the lowest index in query) that they overlap, but not for any potential further list elements containing overlapping regions. This mode can be used to hierarchically and uniquely count (assign) each alignment to a one of several groups of regions (the elements in query), for example to estimate the fractions of different classes of RNA in an RNA-seq experiment (rRNA, tRNA, snRNA, snoRNA, mRNA, etc.)

TxDb : Used to extract regions from annotation and report alignment counts depending on the value of reportLevel. If reportLevel="exon", alignments overlapping each exon in query are counted. If reportLevel="gene", alignment counts for all exons of a gene will be summed, counting each alignment only once even if it overlaps multiple annotated exons of a gene. These are useful to calculate exon or gene expression levels in RNA-seq experiments based on the annotation in a TxDb object. If reportLevel="promoter", the promoters function from package **GenomicFeatures** is used with default arguments to extract promoter regions around transcript start sites, e.g. to quantify alignments inf a ChIP-seq experiment.

any of the above or NULL **for** reportLevel="junction" : The query argument is ignored if reportLevel is set to "junction", and qCount will count the number of alignments supporting each exon-exon junction detected in any of the samples in proj. The arguments selectReadPosition, shift, orientation, useRead and mask will have no effect in this quantification mode.

The additional arguments allow to fine-tune the quantification:

selectReadPosition defines the part of the alignment that has to be contained within a query region for an overlap. The values start (default) and end refer to the biological start (5'-end) and end (3'-end) of the alignment. For example, the start of an alignment on the plus strand is its leftmost (lowest) base, and the end of an alignment on the minus strand is also the leftmost base.

`shift` allows on-the-fly shifting of alignments towards their 3'-end prior to overlap determination and counting. This can be helpful to increase resolution of ChIP-seq experiments by moving alignments by half the immuno-precipitated fragment size towards the middle of fragments. `shift` is either an "integer" vector with one value per alignment file in `proj`, or a single "integer" value, in which case all alignment files will be shifted by the same value. For paired-end experiments, it can be alternatively set to "halfInsert", which will estimate the true fragment size from the distance between aligned read pairs and shift the alignments accordingly.

`orientation` controls the interpretation of alignment strand when counting, relative to the strand of the query region. `any` will count all overlapping alignments, irrespective of the alignment strand (e.g. used in an unstranded RNA-seq experiment). `same` will only count the alignments on the same strand as the query region (e.g. in a stranded RNA-seq experiment that generates reads from the sense strand), and `opposite` will only count the alignments on the opposite strand from the query region (e.g. to quantify anti-sense transcription in a stranded RNA-seq experiment).

`includeSpliced` and `includeSecondary` can be used to include or exclude spliced or secondary alignments, respectively. `mapqMin` and `mapqMax` allow to select alignments based on their mapping qualities. `mapqMin` and `mapqMax` can take integer values between 0 and 255 and equal to $-10log_{10}Pr(\text{mapping position is wrong})$, rounded to the nearest integer. A value 255 indicates that the mapping quality is not available.

In paired-end experiments, `useRead` allows to quantify either all alignments (`useRead="any"`), or only the first (`useRead="first"`) or last (`useRead="last"`) read from a read pair or read group. Note that for `useRead="any"` (the default), an alignment pair that is fully contained within a query region will contribute two counts to the value of that region. `absIsizeMin` and `absIsizeMax` can be used to select alignments based on their insert size (TLEN field in SAM Spec v1.4).

`auxiliaryName` selects the reference sequence for which alignments should be quantified. `NULL` (the default) will select alignments against the genome. If set to a character string that matches one of the auxiliary target names (as specified in the `auxiliaryFile` argument of `qAlign`), the corresponding alignments will be counted.

`mask` can be used to specify a `GRanges` object with regions in the reference sequence to be excluded from quantification. The regions will be considered unstranded (`strand="*"`). Alignments that overlap with a region in `mask` will not be counted. Masking may reduce the effective width of query regions reported by `qCount`, even down to zero for regions that are fully contained in `mask`.

If `clObj` is set to an object that inherits from class `cluster`, for example an object returned by `makeCluster` from package **parallel**, the quantification task is split into multiple chunks and processed in parallel using `clusterMap`. Currently, not all tasks will be efficiently parallelized: For example, a single query region and a single (group of) bam files will not be split into multiple chunks.

Value

A `matrix` with effective query regions width in the first column, and alignment counts in subsequent columns, or a `GRanges` object if `reportLevel="junction"`.

The effective query region width returned as first column in the matrix is calculated by the number of unique, non-masked bases in the reference sequence that contributed to the count of this query name (irrespective if the bases were covered by alignments or not). An effective width of zero indicates that the region was fully masked and will have zero counts in all samples.

The alignment counts in the matrix are contained from column two onwards. For projects with allele-specific quantification, i.e. if a file with single nucleotide polymorphisms was supplied to the

snpFile argument of qAlign, there will be three columns per bam file (number of alignments for Reference, Unknown and Alternative genotypes, with suffixed _R, _U and _A). Otherwise there is a single columns per bam file.

If collapseBySample=TRUE, groups of bam files with identical sample name are combined by summing their alignment counts.

For reportLevel="junction", the return value is a GRanges object. The start and end coordinates correspond to the first and last base in each detected intron. Plus- and minus-strand alignments are quantified separately, so that in an unstranded RNA-seq experiment, the same intron may be represented twice; once for each strand. The counts for each sample are contained in the mcols of the GRanges object.

Author(s)

Anita Lerch, Dimos Gaidatzis and Michael Stadler

See Also

qAlign, qProject, makeCluster from package **parallel**

Examples

```
library(GenomicRanges)
library(Biostrings)
library(Rsamtools)

# copy example data to current working directory
file.copy(system.file(package="QuasR", "extdata"), ".", recursive=TRUE)

# load genome sequence
genomeFile <- "extdata/hg19sub.fa"
gseq <- readDNAStringSet(genomeFile)
chrRegions <- GRanges(names(gseq), IRanges(start=1,width=width(gseq),names=names(gseq)))

# create alignments (paired-end experiment)
sampleFile <- "extdata/samples_rna_paired.txt"
proj <- qAlign(sampleFile, genomeFile, splicedAlignment=TRUE)

# count reads using a "GRanges" query
qCount(proj, query=chrRegions)
qCount(proj, query=chrRegions, useRead="first")

# hierarchical counting using a "GRangesList" query
library(rtracklayer)
annotationFile <- "extdata/hg19sub_annotation.gtf"
gtfRegions <- import.gff(annotationFile, format="gtf", feature.type="exon")
names(gtfRegions) <- mcols(gtfRegions)$source
gtfRegionList <- split(gtfRegions, names(gtfRegions))
names(gtfRegionList)

res3 <- qCount(proj, gtfRegionList)
res3
```

```
# gene expression levels using a "TxDb" query
library("txdbmaker")
genomeRegion <- scanFaIndex(genomeFile)
chrominfo <- data.frame(chrom=as.character(seqnames(genomeRegion)),
                        length=end(genomeRegion),
                        is_circular=rep(FALSE, length(genomeRegion)))
txdb <- makeTxDbFromGFF(annotationFile,
                        format="gtf",
                        chrominfo=chrominfo,
                        dataSource="Ensembl modified",
                        organism="Homo sapiens")

res4 <- qCount(proj, txdb, reportLevel="gene")
res4

# exon-exon junctions
res5 <- qCount(proj, NULL, reportLevel="junction")
res5
```

---

qExportWig *QuasR wig file export*

Description

Create a fixed-step wig file from the alignments in the genomic bam files of the 'QuasR' project.

Usage

```
qExportWig(
  proj,
  file = NULL,
  collapseBySample = TRUE,
  binsize = 100L,
  shift = 0L,
  strand = c("*", "+", "-"),
  scaling = TRUE,
  tracknames = NULL,
  log2p1 = FALSE,
 colors = c("#1B9E77", "#D95F02", "#7570B3", "#E7298A", "#66A61E", "#E6AB02", "#A6761D",
    "#666666"),
  includeSecondary = TRUE,
  mapqMin = 0L,
  mapqMax = 255L,
  absIsizeMin = NULL,
  absIsizeMax = NULL,
  createBigWig = FALSE,
  useRead = c("any", "first", "last"),
```

```
  pairedAsSingle = FALSE,
  clObj = NULL
)
```

Arguments

proj	A qProject object as returned by qAlign.
file	A character vector with the name(s) for the wig or bigWig file(s) to be generated. Either NULL or a vector of the same length as the number of bam files (for collapseBySample=FALSE) or the number of unique sample names (for collapseBySample=TRUE) in proj. If NULL, the wig or bigWig file names are generated from the names of the genomic bam files or unique sample names with an added ".wig.gz" or ".bw" extension.
collapseBySample	If TRUE, genomic bam files with identical sample name will be combined (summed) into a single track.
binsize	A numerical value defining the bin and step size for the wig or bigWig file(s). binsize will be coerced to integer().
shift	Either a vector or a scalar value defining the read shift (e.g. half of fragment length, see 'Details'). If length(shift)>1, the length must match the number of bam files in 'proj', and the i-th sample will be converted to wig or bigWig using the value in shift[i]. shift will be coerced to integer(). For paired-end alignments, shift will be ignored, and a warning will be issued if it is set to a non-zero value (see 'Details').
strand	Only count alignments of strand. The default ("*") will count all alignments.
scaling	If TRUE or a numerical value, the output values in the wig or bigWig file(s) will be linearly scaled by the total number of aligned reads per sample to improve comparability (see 'Details').
tracknames	A character vector with the names of the tracks to appear in the track header. If NULL, the sample names in proj will be used.
log2p1	If TRUE, the number of alignments x per bin will be transformed using the formula log2(x+1).
colors	A character vector with R color names to be used for the tracks.
includeSecondary	If TRUE (the default), include alignments with the secondary bit (0x0100) set in the FLAG.
mapqMin	Minimal mapping quality of alignments to be included (mapping quality must be greater than or equal to mapqMin). Valid values are between 0 and 255. The default (0) will include all alignments.
mapqMax	Maximal mapping quality of alignments to be included (mapping quality must be less than or equal to mapqMax). Valid values are between 0 and 255. The default (255) will include all alignments.
absIsizeMin	For paired-end experiments, minimal absolute insert size (TLEN field in SAM Spec v1.4) of alignments to be included. Valid values are greater than 0 or NULL (default), which will not apply any minimum insert size filtering.

absIsizeMax — For paired-end experiments, maximal absolute insert size (TLEN field in SAM Spec v1.4) of alignments to be included. Valid values are greater than 0 or NULL (default), which will not apply any maximum insert size filtering.

createBigWig — If TRUE, first a temporary wig file will be created and then converted to BigWig format (file extension ".bw") using the wigToBigWig function from package **rtracklayer**.

useRead — For paired-end experiments, selects the read mate whose alignments should be counted, one of:

any **(default)** : count all alignments

first : count only alignments from the first read

last : count only alignments from the last read

For single-read alignments, this argument will be ignored. For paired-end alignments, setting this argument to a value different from the default (any) will cause qExportWig not to automatically use the mid of fragments, but to treat the selected read as if it would come from a single-read experiment (see 'Details').

pairedAsSingle — If TRUE, treat paired-end data single read data, which means that instead of calculating fragment mid-points for each read pair, the 5-prime ends of the reads is used. This is for example useful when analyzing paired-end DNAse-seq or ATAC-seq data, in which the read starts are informative for chromatin accessibility.

clObj — A cluster object to be used for parallel processing of multiple samples.

Details

qExportWig() uses the genome bam files in proj as input to create wig or bigWig files with the number of alignments (pairs) per window of binsize nucleotides. By default (collapseBySample=TRUE), one file per unique sample will be created. If collapseBySample=FALSE, one file per genomic bam file will be created. See http://genome.ucsc.edu/goldenPath/help/wiggle.html for the definition of the wig format, and http://genome.ucsc.edu/goldenPath/help/bigWig.html for the definition of the bigWig format.

The genome is tiled with sequential windows of length binsize, and alignments in the bam file are assigned to these windows: Single read alignments are assigned according to their 5'-end coordinate shifted by shift towards the 3'-end (assuming that the 5'-end is the leftmost coordinate for plus-strand alignments, and the rightmost coordinate for minus-strand alignments). Paired-end alignments are assigned according to the base in the middle between the leftmost and rightmost coordinates of the aligned pair of reads, unless pairedAsSingle = TRUE is used. Each pair of reads is only counted once, and not properly paired alignments are ignored. If useRead is set to select only the first or last read in a paired-end experiment, the selected read will be treated as reads from a single read experiment. Secondary alignments can be excluded by setting includeSecondary=FALSE. In paired-end experiments, absIsizeMin and absIsizeMax can be used to select alignments based on their insert size (TLEN field in SAM Spec v1.4).

For scaling=TRUE, the number of alignments per bin n for the sample i are linearly scaled to the mean total number of alignments over all samples in proj according to: $n_s = n/N[i] * mean(N)$ where n_s is the scaled number of alignments in the bin and N is a vector with the total number of alignments for each sample. Alternatively, if scaling is set to a positive numerical value s, this value is used instead of mean(N), and values are scaled according to: $n_s = n/N[i] * s$.

mapqMin and mapqMax allow to select alignments based on their mapping qualities. mapqMin and mapqMax can take integer values between 0 and 255 and equal to $-10log_{10}Pr(\text{mapping position is wrong})$, rounded to the nearest integer. A value 255 indicates that the mapping quality is not available.

If createBigWig=FALSE and file ends with '.gz', the resulting wig file will be compressed using gzip and is suitable for uploading as a custom track to your favorite genome browser (e.g. UCSC or Ensembl).

Value

(invisible) The file name of the generated wig or bigWig file(s).

Author(s)

Anita Lerch, Dimos Gaidatzis and Michael Stadler

See Also

qProject, qAlign, wigToBigWig

Examples

```
# copy example data to current working directory
file.copy(system.file(package="QuasR", "extdata"), ".", recursive=TRUE)

# create alignments
sampleFile <- "extdata/samples_chip_single.txt"
genomeFile <- "extdata/hg19sub.fa"
proj <- qAlign(sampleFile, genomeFile)

# export wiggle file
qExportWig(proj, binsize=100L, shift=0L, scaling=TRUE)
```

---

qMeth *Quantify DNA methylation*

Description

Quantify methylation of cytosines from bisulfite sequencing data.

Usage

```
qMeth(
  proj,
  query = NULL,
  reportLevel = c("C", "alignment"),
  mode = c("CpGcomb", "CpG", "allC", "var"),
  collapseBySample = TRUE,
```

```
  collapseByQueryRegion = FALSE,
  asGRanges = TRUE,
  mask = NULL,
  reference = "genome",
  keepZero = TRUE,
  mapqMin = 0L,
  mapqMax = 255L,
  clObj = NULL
)
```

Arguments

proj	A qProject object from a bisulfite sequencing experiment
query	A GRanges object with the regions to be quantified. If NULL, all available target sequences (e.g. the whole genome) will be analyzed. Available target sequences are extracted from the header of the first bam file.
reportLevel	Report results combined for C's (reportLevel="C"), the default) or individually for single alignments (reportLevel="alignment"). The latter imposes further restrictions on some arguments (see 'Details').
mode	Cytosine quantification mode, one of: CpGcomb : only C's in CpG context (strands combined) CpG : only C's in CpG context (strands separate) allC : all C's (strands separate) var : variant detection (all C's, strands separate) CpGcomb is the default.
collapseBySample	If TRUE, combine (sum) counts from bamfiles with the same sample name.
collapseByQueryRegion	If TRUE, combine (sum) counts for all cytosines contained in the same query region.
asGRanges	If TRUE, return results as a GRanges object; if FALSE, the results are returned as a data.frame.
mask	An optional GRanges object with genomic regions to be masked, i.e. excluded from the analysis (e.g. unmappable regions).
reference	Source of bam files; can be either "genome" (then the alignments against the genome are used) or the name of an auxiliary target sequence (then alignments against this target sequence will be used). The auxiliary name must correspond to the name contained in the auxiliary file refered by the auxiliaryFile argument of qAlign.
keepZero	If FALSE, only cytosines covered by at least one alignment will be returned; keepZero must be TRUE if multiple samples have the same sample name and collapseBySample is TRUE.
mapqMin	Minimal mapping quality of alignments to be included when counting (mapping quality must be greater than or equal to mapqMin). Valid values are between 0 and 255. The default (0) will include all alignments.

mapqMax	Maximal mapping quality of alignments to be included when counting (mapping quality must be less than or equal to mapqMax). Valid values are between 0 and 255. The default (255) will include all alignments.
clObj	A cluster object to be used for parallel processing of multiple files (see 'Details').

Details

qMeth can be used on a qProject object from a bisulfite sequencing experiment (sequencing of bisulfite-converted DNA), such as the one returned by qAlign when its parameter bisulfite is set to a different value than "no".

qMeth quantifies DNA methylation by counting total and methylated events for individual cytosines, using the alignments that have been generated in converted (three-letter) sequence space for example by qAlign. A methylated event corresponds to a C/C match in the alignment, an unmethylated event to a T/C mismatch (or G/G matches and A/G mismatches on the opposite strand). For paired-end samples, the part of the left fragment alignment that overlaps with the right fragment alignment is ignored, preventing the use of redundant information coming from the same molecule.

Both directed (bisulfite="dir") and undirected (bisulfite="undir") experimental protocols are supported by qAlign and qMeth.

By default, results are returned per C nucleotide. If reportLevel="alignment", results are reported separately for individual alignments. In that case, query has to be a GRanges object with exactly one region, mode has to be either "CpG" or "allC", the arguments collapseByQueryRegion, asGRanges, mask and keepZero have no effect and allele-specific projects are treated in the same way as normal (non-allele specific) projects.

Using the parameter mode, quantification can be limited to cytosines in CpG context, and counts obtained for the two cytosines on opposite strands within a single CpG can be combined (summed).

The quantification of methylation for all cytosines in the query region(s) (mode="allC") should be done with care, especially for large query regions, as the return value may require a large amount of memory.

If mode is set to "var", qMeth only counts reads from the strand opposite of the cytosine and reports total and matching alignments. For a position identical to the reference sequence, only matches (and very few sequencing errors) are expected, independent on the methylation state of the cytosine. A reduced fraction of alignments matching the reference are indicative of sequence variations in the sequenced sample.

mapqMin and mapqMax allow to select alignments based on their mapping qualities. mapqMin and mapqMax can take integer values between 0 and 255 and equal to $-10log_{10}Pr(\text{mapping position is wrong})$, rounded to the nearest integer. A value 255 indicates that the mapping quality is not available.

If an object that inherits from class cluster is provided to the clObj argument, for example an object returned by makeCluster from package **parallel**, the quantification task is split into multiple chunks and processed in parallel using clusterApplyLB from package **parallel**. Not all tasks will be efficiently parallelized: For example, a single query region and a single (group of) bam files will not be split into multiple chunks.

Value

For reportLevel="C", a GRanges object if asGRanges=TRUE, otherwise a data.frame.

Each row contains the coordinates of individual cytosines for `collapseByQueryRegion=FALSE` or query regions for `collapseByQueryRegion=TRUE`.

In addition to the coordinates columns (or `seqnames`, `ranges` and `strand` slots for `GRanges` objects), each row contains per bam file:

Two values (total and methylated events, with suffixes _T and _M), or if the `qProject` object was created including a SNP table, six values (total and methylated events for Reference, Unknown and Alternative genotypes, with suffixed _TR, _TU, _TA, _MR, _MU and _MA). In the latter case, C's or CpG's that overlap with SNPs in the table are removed.

If `collapseBySample=TRUE`, groups of bam files with identical sample name are combined (summed) and will be represented by a single set of total and methylated count columns.

If `mode="var"`, the _T and _M columns correspond to total and matching alignments overlapping the guanine paired to the cytosine.

For `reportLevel="alignment"`, a `list` with one element per bam file or sample (depending on `collapseBySample`). Each list element is another list with the elements:

`aid` : character vector with unique alignment identifiers

`Cid` : integer vector with genomic coordinate of C base

`strand` : character vector with the strand of the C base

`meth` : integer vector with methylation state for alignment and C defined by `aid` and `Cid`. The values are 1 for methylated or 0 for unmethylated states.

Author(s)

Anita Lerch, Dimos Gaidatzis and Michael Stadler

See Also

`qAlign`, `makeCluster` from package **parallel**

Examples

```
# copy example data to current working directory
file.copy(system.file(package="QuasR", "extdata"), ".", recursive=TRUE)

# create alignments
sampleFile <- "extdata/samples_bis_single.txt"
genomeFile <- "extdata/hg19sub.fa"
proj <- qAlign(sampleFile, genomeFile, bisulfite="dir")
proj

# calculate methylation states
meth <- qMeth(proj, mode="CpGcomb")
meth
```

qProfile *Quantify alignments by relative position*

Description

Quantify alignments from sequencing data, relative to their position in query regions.

Usage

```
qProfile(
  proj,
  query,
  upstream = 1000,
  downstream = upstream,
  selectReadPosition = c("start", "end"),
  shift = 0L,
  orientation = c("any", "same", "opposite"),
  useRead = c("any", "first", "last"),
  auxiliaryName = NULL,
  mask = NULL,
  collapseBySample = TRUE,
  includeSpliced = TRUE,
  includeSecondary = TRUE,
  mapqMin = 0L,
  mapqMax = 255L,
  absIsizeMin = NULL,
  absIsizeMax = NULL,
  maxInsertSize = 500L,
  binSize = 1L,
  clObj = NULL
)
```

Arguments

proj	A qProject object representing a sequencing experiment as returned by qAlign
query	An object of type GRanges with the regions to be profiled. All regions in query will be anchored at their biological start position (start(query) for regions on strand "+" or "*", end(query) for regions on strand "-"). This position will become position zero in the return value.
upstream	An "integer" vector of length one or the same length as query indicating the number of bases upstream of the anchor position to include in the profile.
downstream	An "integer" vector of length one or the same length as query indicating the number of bases downstream of the anchor position to include in the profile.
selectReadPosition	defines the part of the alignment that has to be contained within a query region to produce an overlap (see Details), and that is used to calculate the relative position within the query region. Possible values are:

`start` **(default)** : start of the alignment

`end` : end of the alignment

`shift` controls the shifting alignments towards their 3'-end before quantification. `shift` can be one of:

- an "integer" vector of the same length as the number of alignment files
- a single "integer" value
- the character string `"halfInsert"` (only available for paired-end experiments)

The default of `0` will not shift any alignments.

`orientation` sets the required orientation of the alignments relative to the query region in order to be counted, one of:

`any` **(default)** : count alignment on the same and opposite strand

`same` : count only alignment on the same strand

`opposite` : count only alignment on the opposite strand

`useRead` For paired-end experiments, selects the read mate whose alignments should be counted, one of:

`any` **(default)** : count all alignments

`first` : count only alignments from the first read

`last` : count only alignments from the last read

`auxiliaryName` Which bam files to use in an experiments with auxiliary alignments (see Details).

`mask` If not `NULL`, a `GRanges` object with reference regions to be masked, i.e. excluded from the quantification, such as unmappable or highly repetitive regions (see Details).

`collapseBySample`
If `TRUE` (the default), sum alignment counts from bam files with the same sample name.

`includeSpliced` If `TRUE` (the default), include spliced alignments when counting. A spliced alignment is defined as an alignment with a gap in the read of at least 60 bases.

`includeSecondary`
If `TRUE` (the default), include alignments with the secondary bit (0x0100) set in the `FLAG` when counting.

`mapqMin` Minimal mapping quality of alignments to be included when counting (mapping quality must be greater than or equal to `mapqMin`). Valid values are between 0 and 255. The default (0) will include all alignments.

`mapqMax` Maximal mapping quality of alignments to be included when counting (mapping quality must be less than or equal to `mapqMax`). Valid values are between 0 and 255. The default (255) will include all alignments.

`absIsizeMin` For paired-end experiments, minimal absolute insert size (TLEN field in SAM Spec v1.4) of alignments to be included when counting. Valid values are greater than 0 or `NULL` (default), which will not apply any minimum insert size filtering.

`absIsizeMax` For paired-end experiments, maximal absolute insert size (TLEN field in SAM Spec v1.4) of alignments to be included when counting. Valid values are greater than 0 or `NULL` (default), which will not apply any maximum insert size filtering.

maxInsertSize	Maximal fragment size of the paired-end experiment. This parameter is used if shift="halfInsert" and will ensure that query regions are made wide enough to emcompass all alignment pairs whose mid falls into the query region. The default value is 500 bases.
binSize	Numeric scalar giving the size of bins (must be an odd number). The default value (1) gives back counts for single bases. Otherwise, alignments are counted in adjacent, non-overlapping windows of size binSize that tile the interval defined by upstream and downstream.
clObj	A cluster object to be used for parallel processing (see 'Details').

Details

qProfile is used to count alignments in each sample from a qProject object, relative to their position in query regions.

Most arguments are identical to the ones of qCount.

The query argument is a GRanges object that defines the regions for the profile. All regions in query will be aligned to one another at their anchor position, which corresponds to their biological start position (start(query) for regions on strand "+" or "*", end(query) for regions on strand "-").

This anchor position will be extended (with regard to strand) by the number of bases specified by upstream and downstream. In the return value, the anchor position will be at position zero.

If binSize is greater than one, upstream and downstream will be slightly increased in order to include the complete first and last bins of binSize bases.

Regions with identical names in names{query} will be summed, and profiles will be padded with zeros to accomodate the length of all profiles.

Value

A list of matrices with length(unique(names(query))) rows with profile names, and max(upstream)+max(downstream)
columns indicating relative position (for binsize=1).

For binSize values greater than 1, the number of columns corresponds to the number of bins (tiles), namely ceiling(max(upstream)/binSize)+ceiling(max(downstream)/binSize). A middle bin of size binSize is always positioned centered at the anchor of each region. Additional bins are positioned upstream and downstream, adjacent to that middle bin, in order to include at least upstream and downstream bases, respectively (potentially more in order to fill the first and last bins).

The relative positions are given as column names (for binSize > 1 they refer to the bin mid). In that case, the bins are "right-open". For example, if binSize = 10, the bin with the midpoint "-50" contains counts for the alignments in [-55,-45).

The first list element is called "coverage" and contains, for each profile and relative position, the number of overlapping regions that contributed to the profile.

Subsequent list elements contain the alignment counts for individual sequence files (collapseBySample=FALSE) or samples (collapseBySample=TRUE) in proj.

For projects with allele-specific quantification, i.e. if a file with single nucleotide polymorphisms was supplied to the snpFile argument of qAlign, there will be three rows instead of one row

with counts per unique region name, with numbers of alignments for Reference, Unknown and Alternative genotypes (suffixed _R, _U and _A).

Author(s)

Anita Lerch, Dimos Gaidatzis and Michael Stadler

See Also

qCount, qAlign, qProject, makeCluster from package **parallel**

Examples

```
# copy example data to current working directory
file.copy(system.file(package="QuasR", "extdata"), ".", recursive=TRUE)

# create alignments (single-end experiment)
genomeFile <- "extdata/hg19sub.fa"
sampleFile <- "extdata/samples_chip_single.txt"
proj <- qAlign(sampleFile, genomeFile)

# load transcript start site coordinates
library(rtracklayer)
annotationFile <- "extdata/hg19sub_annotation.gtf"
tssRegions <- import.gff(annotationFile, format="gtf",
                         feature.type="start_codon")

# obtain a combined TSS profile
pr1 <- qProfile(proj, tssRegions)
lapply(pr1, dim)
lapply(pr1, "[", , 1:5)

prComb <- do.call("+", lapply(pr1[-1], function(x) x/pr1[[1]]))
barplot(prComb, xlab="Position", ylab="Mean no. of alignments")

# obtain TSS profiles for individual regions
names(tssRegions) <- mcols(tssRegions)$transcript_id
pr2 <- qProfile(proj, tssRegions)
lapply(pr2, dim)
lapply(pr2, "[", 1:3, 1:5)
```

---

qProject-class *qProject objects*

Description

The qProject class is a container for the meta-data (e.g. sample names, paths and names of sequence and alignment files) associated with a high-throughput sequencing experiment analyzed with QuasR.

Details

The qProject class is returned by qAlign and stores all information on a high-throughput sequencing experiment analyzed with QuasR. qProject objects can be conveniently passed to 'q'-functions (function name starting with the letter 'q'). The information is stored in the following slots:

`reads` a 'data.frame' with sequence read files.

`reads_md5subsum` a 'data.frame' with fingerprints for sequence read files.

`alignments` a 'data.frame' with alignment files.

`samplesFormat` a 'character(1)' specifying the format of input files.

`genome` a 'character(1)' specifying the reference genome.

`genomeFormat` a 'character(1)' specifying the format of the reference genome.

`aux` a 'data.frame' with auxiliary reference sequence files.

`auxAlignments` a 'data.frame' with alignment files for auxiliary reference sequence files.

`aligner` a 'character(1)' specifying the aligner.

`maxHits` a 'numeric(1)' specifying the maximum number of alignments per sequence.

`paired` a 'character(1)' specifying the paired-type; one of "no", "fr", "rf", "ff".

`splicedAlignment` a 'logical(1)'; `TRUE` when performing spliced-alignments.

`snpFile` a 'character(1)' with a file name containing SNP information.

`bisulfite` a 'character(1)' defining the bisulfite type; one of "no", "dir", "undir".

`alignmentParameter` a 'character(1)' with aligner command line parameters.

`projectName` a 'character(1)' with the project name.

`alignmentsDir` a 'character(1)' with the directory to be used to store alignment files.

`lib.loc` a 'character(1)' with the library directory to use for installing of alignment index packages.

`cacheDir` a 'character(1)' with a directory to use for temporary files.

`alnModeID` a 'character(1)' used internally to indicate the alignment mode.

Accessors

In the following code snippets, `x` is a qProject object.

`length(x)` : Gets the number of input files.

`genome(x)` : Gets the reference genome as a 'character(1)'. The type of genome is stored as an attribute in `attr(genome(x),"genomeFormat")`: "BSgenome" indicates that `genome(x)` refers to the name of a BSgenome package, "file" indicates that it contains the path and filename of a genome in FASTA format.

`auxiliaries(x)` : Gets a `data.frame` with auxiliary target sequences, with one row per auxiliary target, and columns "FileName" and "AuxName".

`alignments(x)` : Gets a list with two elements "genome" and "aux". `alignments(x)$genome` contains a `data.frame` with `length(x)` rows and the columns "FileName" (containing the path to bam files with genomic alignments) and "SampleName". `alignments(x)$aux` contains a `data.frame` with one row per auxiliary target sequence (with auxiliary names as row names), and `length(x)` columns.

Subsetting

In the following code snippets, x is a qProject object.

x[i] : Get qProject object instance with i input files, where i can be an NA-free logical, numeric, or character vector.

Author(s)

Anita Lerch, Dimos Gaidatzis and Michael Stadler

See Also

qAlign

Examples

```
# copy example data to current working directory
file.copy(system.file(package="QuasR", "extdata"), ".", recursive=TRUE)

# create alignments
sampleFile <- "extdata/samples_chip_single.txt"
genomeFile <- "extdata/hg19sub.fa"
auxFile <- "extdata/auxiliaries.txt"

proj <- qAlign(sampleFile, genomeFile, auxiliaryFile=auxFile)
proj

# alignment statistics using a qProject
alignmentStats(proj)

# alignment statistics using bam files
alignmentStats(alignments(proj)$genome$FileName)
alignmentStats(unlist(alignments(proj)$aux))
```

---

qProjectUpdate *Update qProject*

Description

Check if a qProject is compatible with the current class definition, and update it if necessary.

Usage

```
qProjectUpdate(proj, quiet = TRUE)
```

Arguments

proj	a qProject object.
quiet	logical, whether to suppress update messages.

Value

A qProject object.

Author(s)

Charlotte Soneson

See Also

qProject

---

qQCReport *QuasR Quality Control Report*

Description

Generate quality control plots for a qProject object or a vector of fasta/fastq/bam files. The available plots vary depending on the types of available input (fasta, fastq, bam files or qProject object; paired-end or single-end).

Usage

```
qQCReport(
  input,
  pdfFilename = NULL,
  chunkSize = 1000000L,
  useSampleNames = FALSE,
  clObj = NULL,
  a4layout = TRUE,
  ...
)
```

Arguments

input	A vector of files or a qProject object as returned by qAlign.
pdfFilename	The path and name of a pdf file to store the report. If NULL, the quality control plots will be generated in separate plotting windows on the standard graphical device.
chunkSize	The number of sequences, sequence pairs (for paired-end data) or alignments that will be sampled from each data file to collect quality statistics.
useSampleNames	If TRUE, the plots will be labelled using the sample names instead of the file names. Sample names are obtained from the qProject object, or from names(input) if input is a named vector of file names. Please not that if there are multiple files for the same sample, the sample names will not be unique.
clObj	A cluster object to be used for parallel processing of multiple input files.

`a4layout` A logical scalar. If TRUE, the output of mapping rate and uniqueness plots will be adjusted for a4 format devices.

`...` Additional arguments that will be passed to the functions generating the individual quality control plots, see 'Details'.

Details

This function generates quality control plots for all input files or the sequence and alignment files contained in a `qProject` object, allowing assessment of the quality of a sequencing experiment. `qQCReport` uses functionality from the **ShortRead** package to collect quality data, and visualizes the results similarly as the 'FastQC' quality control tool from Simon Andrews (see 'References' below). It is recommended to create PDF reports (`pdfFilename` argument), for which the plot layouts have been optimised.

Some plots will only be generated if the necessary information is available (e.g. base qualities in fastq sequence files).

The currently available plot types are:

Quality score boxplot shows the distribution of base quality values as a box plot for each position in the input sequence. The background color (green, orange or red) indicates ranges of high, intermediate and low qualities.

Nucleotide frequency plot shows the frequency of A, C, G, T and N bases by position in the read.

Duplication level plot shows for each sample the fraction of reads observed at different duplication levels (e.g. once, two-times, three-times, etc.). In addition, the most frequent sequences are listed.

Mapping statistics shows fractions of reads that were (un)mappable to the reference genome.

Library complexity shows fractions of unique read(-pair) alignment positions, as a measure of the complexity in the sequencing library. Please note that this measure is not independent from the total number of reads in a library, and is best compared between libraries of similar sizes.

Mismatch frequency shows the frequency and position (relative to the read sequence) of mismatches in the alignments against the reference genome.

Mismatch types shows the frequency of read bases that caused mismatches in the alignments to the reference genome, separately for each genome base.

Fragment size shows the distribution of fragment sizes inferred from aligned read pairs.

One approach to assess the quality of a sample is to compare its control plots to the ones from other samples and search for relative differences. Special quality measures are expected for certain types of experiments: A genomic re-sequencing sample with an overrepresentation of T bases may be suspicious, while such a nucleotide bias is normal for a directed bisulfite-sequencing sample.

Additional arguments can be passed to the internal functions that generate the individual quality control plots using `...{}`:

`lmat`**:** a matrix (e.g. `matrix(1:12, ncol=2)`) used by an internal call to the `layout` function to specify the positioning of multiple plot panels on a device page. Individual panels correspond to different samples.

`breaks`**:** a numerical vector (e.g. `c(1:10)`) defining the bins used by the 'Duplication level' plot.

Value

The function is called for its side effect of generating quality control plots. It invisibly returns a list with components that contain the data used to generate each of the QC plots. Available components are (depending on input data, see 'Details'):

qualByCycle : quality score boxplot

nuclByCycle : nucleotide frequency plot

duplicated : duplication level plot

mappings : mapping statistics barplot

uniqueness : library complexity barplot

errorsByCycle : mismatch frequency plot

mismatchTypes : mismatch type plot

fragDistribution : fragment size distribution plot

Author(s)

Anita Lerch, Dimos Gaidatzis and Michael Stadler

References

FastQC quality control tool at http://www.bioinformatics.bbsrc.ac.uk/projects/fastqc/

See Also

qProject, qAlign, ShortRead package

Examples

```
# copy example data to current working directory
file.copy(system.file(package="QuasR", "extdata"), ".", recursive=TRUE)

# create alignments
sampleFile <- "extdata/samples_chip_single.txt"
genomeFile <- "extdata/hg19sub.fa"

proj <- qAlign(sampleFile, genomeFile)

# create quality control report
qQCReport(proj, pdfFilename="qc_report.pdf")
```

worker_message *Build a formatted message for outputting to log files*

Description

Build a formatted message for outputting to log files

Usage

```
worker_message(..., sep = "", appendLF = TRUE)
```

Index

∗ **internal**
.get_alnstats_for_bam, 3
QuasR-package, 2
worker_message, 33
∗ **methods**
qQCReport, 30
∗ **misc**
alignmentStats, 4
preprocessReads, 5
qAlign, 8
qCount, 12
qMeth, 20
qProfile, 24
qProjectUpdate, 29
∗ **package**
QuasR-package, 2
∗ **utilities**
alignmentStats, 4
preprocessReads, 5
qAlign, 8
qCount, 12
qExportWig, 17
qMeth, 20
qProfile, 24
qProjectUpdate, 29
.get_alnstats_for_bam, 3
[,qProject,ANY,missing,missing-method *(*qProject-class*)*, 27

alignments *(*qProject-class*)*, 27
alignments,qProject-method *(*qProject-class*)*, 27
alignmentStats, 4
auxiliaries *(*qProject-class*)*, 27
auxiliaries,qProject-method *(*qProject-class*)*, 27

class:qProject *(*qProject-class*)*, 27
clusterApplyLB, *22*
clusterMap, *7*, *15*

genome,qProject-method *(*qProject-class*)*, 27
GRanges, *12–15*, *24–26*
GRangesList, *12*, *14*

length,qProject-method *(*qProject-class*)*, 27

makeCluster, *7*, *11*, *15*, *16*, *22*, *23*, *27*

preprocessReads, 5

qAlign, *3*, 8, *12*, *15*, *16*, *20–24*, *26–29*, *32*
qCount, *3*, 12, *26*, *27*
qExportWig, 17
qMeth, *3*, 20
qProfile, *3*, 24
qProject, *4*, *11*, *12*, *16*, *20*, *24*, *27*, *29*, *30*, *32*
qProject *(*qProject-class*)*, 27
qProject-class, 27
qProjectUpdate, 29
qQCReport, *3*, 30
QuasR *(*QuasR-package*)*, 2
QuasR-package, 2
quickBamFlagSummary, *4*

ShortRead, *32*
show,qProject-method *(*qProject-class*)*, 27

trimLRPatterns, *6*, *7*
TxDb, *12*, *14*

wigToBigWig, *19*, *20*
worker_message, 33