Package 'Rsubread'

October 18, 2022

Version 2.10.5

Date 2022-08-09

Title Mapping, quantification and variant analysis of sequencing data

Author Wei Shi, Yang Liao and Gordon K Smyth with contributions from Jenny Dai

Maintainer Wei Shi <wei.shi@onjcri.org.au>, Yang Liao <yang.liao@onjcri.org.au> and Gordon K Smyth <smyth@wehi.edu.au>

Description Alignment, quantification and analysis of RNA sequencing data (including both bulk RNA-seq and scRNA-seq) and DNA sequenicng data (including ATAC-seq, ChIP-seq, WGS, WES etc). Includes functionality for read mapping, read counting, SNP calling, structural variant detection and gene fusion discovery. Can be applied to all major sequencing techologies and to both short and long sequence reads.

URL http://bioconductor.org/packages/Rsubread

License GPL (>=3)

Imports grDevices, stats, utils, Matrix

biocViews Sequencing, Alignment, SequenceMatching, RNASeq, ChIPSeq, SingleCell, GeneExpression, GeneRegulation, Genetics, ImmunoOncology, SNP, GeneticVariability, Preprocessing, QualityControl, GenomeAnnotation, GeneFusionDetection, IndelDetection, VariantAnnotation, VariantDetection, MultipleSequenceAlignment

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

git_branch RELEASE_3_15

git_last_commit 5c2d59e

git_last_commit_date 2022-08-12

Date/Publication 2022-10-18

R topics documented:

`align` *Align Sequence Reads to a Reference Genome via Seed-and-Vote*

Description

The `align` function can align both DNA and RNA sequencing reads. `Subjunc` is an RNA-seq aligner and it reports full alignment of each read (`align` reports partial alignment for exon spanning reads).

Usage

```
align(

    # index for reference sequences
    index,

    # input reads and output
    readfile1,
    readfile2 = NULL,
    type = "rna",
    input_format = "gzFASTQ",
    output_format = "BAM",
    output_file = paste(readfile1,"subread",output_format,sep="."),
```

```
# offset value added to Phred quality scores of read bases
phredOffset = 33,

# thresholds for mapping
nsubreads = 10,
TH1 = 3,
TH2 = 1,
maxMismatches = 3,

# unique mapping and multi-mapping
unique = FALSE,
nBestLocations = 1,

# indel detection
indels = 5,
complexIndels = FALSE,

# read trimming
nTrim5 = 0,
nTrim3 = 0,

# distance and orientation of paired end reads
minFragLength = 50,
maxFragLength = 600,
PE_orientation = "fr",

# number of CPU threads
nthreads = 1,

# read group
readGroupID = NULL,
readGroup = NULL,

# read order
keepReadOrder = FALSE,
sortReadsByCoordinates = FALSE,

# color space reads
color2base = FALSE,

# dynamic programming
DP_GapOpenPenalty = -1,
DP_GapExtPenalty = 0,
DP_MismatchPenalty = 0,
DP_MatchScore = 2,

# detect structural variants
detectSV = FALSE,
```

```
    # gene annotation
    useAnnotation = FALSE,
    annot.inbuilt = "mm39",
    annot.ext = NULL,
    isGTF = FALSE,
    GTF.featureType = "exon",
    GTF.attrType = "gene_id",
    chrAliases = NULL)

subjunc(

    # index for reference sequences
    index,

    # input reads and output
    readfile1,
    readfile2 = NULL,
    input_format = "gzFASTQ",
    output_format = "BAM",
    output_file = paste(readfile1,"subjunc",output_format,sep = "."),

    # offset value added to Phred quality scores of read bases
    phredOffset = 33,

    # thresholds for mapping
    nsubreads = 14,
    TH1 = 1,
    TH2 = 1,
    maxMismatches = 3,

    # unique mapping and multi-mapping
    unique = FALSE,
    nBestLocations = 1,

    # indel detection
    indels = 5,
    complexIndels = FALSE,

    # read trimming
    nTrim5 = 0,
    nTrim3 = 0,

    # distance and orientation of paired end reads
    minFragLength = 50,
    maxFragLength = 600,
    PE_orientation = "fr",
```

```
    # number of CPU threads
    nthreads = 1,

    # read group
    readGroupID = NULL,
    readGroup = NULL,

    # read order
    keepReadOrder = FALSE,
    sortReadsByCoordinates = FALSE,

    # color space reads
    color2base = FALSE,

    # dynamic programming
    DP_GapOpenPenalty = -1,
    DP_GapExtPenalty = 0,
    DP_MismatchPenalty = 0,
    DP_MatchScore = 2,

    # detect all junctions including gene fusions
    reportAllJunctions = FALSE,

    # gene annotation
    useAnnotation = FALSE,
    annot.inbuilt = "mm39",
    annot.ext = NULL,
    isGTF = FALSE,
    GTF.featureType = "exon",
    GTF.attrType = "gene_id",
    chrAliases = NULL)
```

Arguments

index	character string giving the basename of index file. Index files should be located in the current directory.
readfile1	a character vector including names of files that include sequence reads to be aligned. For paired-end reads, this gives the list of files including first reads in each library. File format is FASTQ/FASTA by default. See input_format option for more supported formats.
readfile2	a character vector giving names of files that include second reads in paired-end read data. Files included in readfile2 should be in the same order as their mate files included in readfile1 .NULL by default.
type	a character string or an integer giving the type of sequencing data. Possible values include rna (or 0; RNA-seq data) and dna (or 1; genomic DNA-seq data such as WGS, WES, ChIP-seq data etc.). Character strings are case insensitive.
input_format	character string specifying format of read input files. gzFASTQ by default (this also includes FASTQ, FASTA, gzipped FASTQ and gzipped FASTA formats).

Other supported formats include SAM and BAM. Character values are case insensitive.

output_format — character string specifying format of output file. BAM by default. Acceptable formats include SAM and BAM.

output_file — a character vector specifying names of output files. By default, names of output files are set as the file names provided in readfile1 added with an suffix string.

phredOffset — numeric value added to base-calling Phred scores to make quality scores (represented as ASCII letters). Possible values include 33 and 64. By default, 33 is used.

nsubreads — numeric value giving the number of subreads extracted from each read.

TH1 — numeric value giving the consensus threshold for reporting a hit. This is the threshold for the first reads if paired-end read data are provided.

TH2 — numeric value giving the consensus threhold for the second reads in paired-end data.

maxMismatches — numeric value giving the maximum number of mis-matched bases allowed in the alignment. 3 by default. Mis-matches found in soft-clipped bases are not counted.

unique — logical indicating if only uniquely mapped reads should be reported. A uniquely mapped read has one single mapping location that has less mis-matched bases than any other candidate locations. By default, multi-mapping reads will be reported in addition to uniquely mapped reads. Number of alignments reported for each multi-mapping read is determined by the nBestLocations parameter.

nBestLocations — numeric value specifying the maximal number of equally-best mapping locations that will be reported for a multi-mapping read. 1 by default. The allowed value is between 1 to 16 (inclusive). In the mapping output, 'NH' tag is used to indicate how many alignments are reported for the read and 'HI' tag is used for numbering the alignments reported for the same read. This argument is only applicable when unique option is set to FALSE.

indels — numeric value giving the maximum number of insertions/deletions allowed during the mapping. 5 by default.

complexIndels — logical indicating if complex indels will be detected. If TRUE, the program will try to detect multiple short indels that occurs concurrently in a small genomic region (indels could be as close as 1bp apart).

nTrim5 — numeric value giving the number of bases trimmed off from 5' end of each read. 0 by default.

nTrim3 — numeric value giving the number of bases trimmed off from 3' end of each read. 0 by default.

minFragLength — numeric value giving the minimum fragment length. 50 by default.

maxFragLength — numeric value giving the maximum fragment length. 600 by default.

PE_orientation — character string giving the orientation of the two reads from the same pair. It has three possible values including fr, ff and rf. Letter f denotes the forward strand and letter r the reverse strand. fr by default (ie. the first read in the pair is on the forward strand and the second read on the reverse strand).

`nthreads` numeric value giving the number of threads used for mapping. `1` by default.

`readGroupID` a character string giving the read group ID. The specified string is added to the read group header field and also be added to each read in the mapping output. `NULL` by default.

`readGroup` a character string in the format of `tag:value`. This string will be added to the read group (RG) header in the mapping output. `NULL` by default.

`keepReadOrder` logical indicating if the order of reads in the BAM output is kept the same as that in the input file. (Reads from the same pair are always placed next to each other regardless of this option).

`sortReadsByCoordinates` logical indicating if reads will be sorted by their mapping locations in the mapping output. This option is applicable for BAM output only. A BAI index file will also be generated for each BAM file so the generated BAM files can be directly loaded into a genome browser such as IGB or IGV. `FALSE` by default.

`color2base` logical. If `TRUE`, color-space read bases will be converted to base-space bases in the mapping output. Note that the mapping itself will still be performed at color-space. `FALSE` by default.

`DP_GapOpenPenalty` a numeric value giving the penalty for opening a gap when using the Smith-Waterman dynamic programming algorithm to detect insertions and deletions. The Smith-Waterman algorithm is only applied for those reads which are found to contain insertions or deletions. `-1` by default.

`DP_GapExtPenalty` a numeric value giving the penalty for extending the gap, used by the Smith-Waterman algorithm. `0` by default.

`DP_MismatchPenalty` a numeric value giving the penalty for mismatches, used by the Smith-Waterman algorithm. `0` by default.

`DP_MatchScore` a numeric value giving the score for matches used by the Smith-Waterman algorithm. `2` by default.

`detectSV` logical indicating if structural variants (SVs) will be detected during read mapping. See below for more details.

`reportAllJunctions` logical indicating if all discovered junctions will be reported. This includes exon-exon junctions and also gene fusions. Presence of donor/receptor sites is not required when `reportAllJunctions` is `TRUE`. This option should only be used for RNA-seq data.

`useAnnotation` logical indicating if gene annotation information will be used in the mapping. `FALSE` by default.

`annot.inbuilt` a character string specifying an in-built annotation used for read summarization. It has five possible values `"mm39"`, `"mm10"`, `"mm9"`, `"hg38"` and `"hg19"`, corresponding to the NCBI RefSeq annotations for genomes 'mm39', 'mm10', 'mm9', 'hg38' and 'hg19', respectively. `"mm39"` by default. See `featureCounts` function for more details on the in-built annotations.

annot.ext — A character string giving name of a user-provided annotation file or a data frame including user-provided annotation data. If the annotation is in GTF format, it can only be provided as a file. If it is in SAF format, it can be provided as a file or a data frame. The provided annotation file can be a uncompressed or gzip compressed file. If an annotation is provided via annot.ext, the annot.inbuilt parameter will be ignored. See featureCounts function for more details about this parameter.

isGTF — logical indicating if the annotation provided via the annot.ext argument is in GTF format or not. FALSE by default. This option is only applicable when annot.ext is not NULL.

GTF.featureType — a character string denoting the type of features that will be extracted from a GTF annotation. "exon" by default. This argument is only applicable when isGTF is TRUE.

GTF.attrType — a character string denoting the type of attributes in a GTF annotation that will be used to group features. "gene_id" by default. The grouped features are called meta-features. For instance, a feature can be an exon and several exons can be grouped into a gene (meta-feature). This argument is only applicable when isGTF is TRUE.

chrAliases — a character string providing the name of a comma-delimited text file that includes aliases of chromosome names. This file should contain two columns. First column contains names of chromosomes included in the SAF or GTF annotation and second column contains corresponding names of chromosomes in the reference genome. No column headers should be provided. Also note that chromosome names are case sensitive. This file can be used to match chromosome names between the annotation and the reference genome.

Details

align and subjunc are R versions of programs in the Subread suite of programs (http://subread.sourceforge.net) that implement fast, accurate sequence read alignment based on the "seed-and-vote" mapping paradigm (Liao et al. 2013; Liao et al. 2019). align is the R version of the command-line program subread-align and subjunc is the R version of the command-line program subjunc.

align is a general-purpose aligner that can be used to align both genomic DNA-seq reads and RNA-seq reads. subjunc is designed for mapping RNA-seq reads only. The main difference between subjunc and align for RNA-seq alignment is that subjunc reports discovered exon-exon junctions and it also performs full alignments for every read including exon-spanning reads. align instead reports the largest mappable regions for each read and soft-clips the remainder of the read. subjunc requires the presence of donor and receptor sites when calling exon-exon junctions. It can detect up to four junction locations in a junction read.

align and subjunc can be run with either a full index or a gapped index (see buildindex for more details). Maximum mapping speed is achieved when the full index is used. With a full index built for human/mouse genome, align maps ~14 million reads per minute with 10 threads. subjunc is slightly slower than align. Both align and subjunc require 17.8GB of memory for mapping. With a gapped index built for human/mouse genome, align maps ~9 million reads per minute with 10 threads. align requires 8.2GB of memory for mapping and subjunc requires 8.8GB of memory.

The detectSV argument of align can be used to detect structural variants (SVs) in genomic DNA sequencing data. The reportAllJunctions argument of subjunc function can be used for the detection of SVs in RNA-seq data, as well as the detection of non-canonical junctions that do not have the canonical donor/receptor sites. For each library, breakpoints detected from SV events are saved to a file with suffix name '.breakpoints.txt', which includes chromosomal coordinates of SV breakpoints and numbers of supporting reads. The BAM/SAM output includes extra fields to describe the complete alignments of breakpoint-containing reads. For a breakpoint-containing read, mapping of its major sequence segment is described in the main fields of BAM/SAM output whereas mapping of its minor sequence segment, which does not map along with the major segment due to the presence of a breakpoint, is described in the extra fields including 'CC'(Chr), 'CP'(Position),'CG'(CIGAR) and 'CT'(strand). Note that each breakpoint-containing read occupies only one row in the BAM/SAM output.

Value

For each library, mapping results including both aligned and unaligned reads are saved to a file. Indels discovered during mapping are saved to a VCF format file (.indel.vcf). subjunc also outputs a BED format file that contains list of discovered exon-exon junctions (.junction.bed). If detectSV or reportAllJunctions options are set to TRUE, structural variants discovered during read mapping will be reported and saved to a file (.breakpoints.txt), which includes chromosomal coordinates of breakpoints and number of supporting reads for each breakpoint.

Other than outputting data to files, align and subjunc also return a data.frame that includes mapping statistics for each library, such as total number of reads, number of mapped reads, number of uniquely mapped reads, number of indels and number of junctions (subjunc only).

Author(s)

Wei Shi and Yang Liao

References

Yang Liao, Gordon K Smyth and Wei Shi (2019). The R package Rsubread is easier, faster, cheaper and better for alignment and quantification of RNA sequencing reads. *Nucleic Acids Research*, 47(8):e47. http://www.ncbi.nlm.nih.gov/pubmed/30783653

Yang Liao, Gordon K Smyth and Wei Shi (2013). The Subread aligner: fast, accurate and scalable read mapping by seed-and-vote. *Nucleic Acids Research*, 41(10):e108. http://www.ncbi.nlm.nih.gov/pubmed/23558742

See Also

buildindex

Examples

```
# Build an index for the artificial sequence included in file 'reference.fa'.
ref <- system.file("extdata","reference.fa",package="Rsubread")

buildindex(basename="./reference_index",reference=ref)
```

```
# align a sample read dataset ('reads.txt') to the sample reference
reads <- system.file("extdata","reads.txt.gz",package="Rsubread")

align.stat <- align(index = "./reference_index", readfile1 = reads,
output_file = "./Rsubread_alignment.BAM", phredOffset = 64)

align.stat
```

`atgcContent` *Calculate Percentages of Nucleotides in Reads*

Description

Calculate percentages of nucleotides A, T, G and C in the input FASTQ or FASTQ file.

Usage

```
atgcContent(

    filename,
    basewise = FALSE)
```

Arguments

`filename`	a character string giving the name of input FASTQ/FASTA file
`basewise`	logical specifying how the percentages are calculated. If `TRUE`, nucleotide percentages will be calculated for each base position in the read across all the reads. By default, percentages are calculated for the entire dataset.

Details

Sequencing reads could contain letter "N" besides "A", "T", "G" and "C". Percentage of "N" in the read dataset is calcuated as well.

The `basewise` calculation is useful for examining the GC bias towards the base position in the read. By default, the percentages of nucleotides in the entire dataset will be reported.

Value

A named vector containing percentages for each nucleotide type if `basewise` is `FALSE`. Otherwise, a data matrix containing nucleotide percentages for each base position of the reads.

Author(s)

Zhiyin Dai and Wei Shi

buildindex *Build Index for a Reference Genome*

Description

An index needs to be built before read mapping can be performed. This function creates a hash table index for the reference genome, which can then be used by Subread and Subjunc aligners for read alignment.

Usage

```
buildindex(

    # basic input/output options
    basename,
    reference,

    # options for the details of the index
    gappedIndex = FALSE,
    indexSplit = FALSE,
    memory = 8000,
    TH_subread = 100,
    colorspace = FALSE)
```

Arguments

basename	a character string giving the basename of created index files.
reference	a character string giving the name of a FASTA or gzipped FASTA file containing the sequences of all chromosomes and contigs.
gappedIndex	logical indicating whether a gapped index or a full index should be built. A gapped index contains 16mers (subreads) that are extracted every three bases from a reference genome, whereas a full index contains subreads extracted from every chromosomal location of a genome. The index provides a hash table mapping the sequences of all subreads to their corresponding chromosomal locations. Default value is FALSE (i.e., a full index is built).
indexSplit	logical indicating whether the index can be split into multiple blocks. The block size is determined by the value of memory. FALSE by default (ie. a single-block index is generated).
memory	a numeric value specifying the amount of memory (in megabytes) used for storing the index during read mapping. 8000 MB by default. Note that this option is ignored when indexSplit is FALSE.
TH_subread	a numeric value specifying the threshold for removing highly repetitive subreads (16bp mers). 100 by default. Subreads will be excluded from the index if they occur more than threshold number of times in the genome.
colorspace	logical specifying the mode of the index. If TRUE, a color space index will be built. Otherwise, a base space index will be built.

Details

This function generates a hash-table index for a reference genome, in which keys are subreads (16mers) and values are their chromosomal locations in the reference genome. The built index can then be used by Subread (align) and subjunc aligners to map reads (Liao et al. 2019; Liao et al. 2013). Index building is an one-off operation.

Highly repetitive subreads (or uninformative subreads) are excluded from the hash table so as to reduce mapping ambiguity. TH_subread specifies the maximal number of times a subread is allowed to occur in the reference genome to be included in hash table.

Maximum mapping speed can be achieved by building a full index for the reference genome. By default buildindex builds a full index. Building a gapped index will significantly reduce the memory use, at a modest cost to read mapping time. It is recommended to use a gapped index on a personal computer due to the limited amount of computer memory available. Memory use can be further reduced by splitting an index to multiple blocks. The amount of memory to be used in read mapping is determined at the index building stage.

To build a full index for human/mouse genome, buildindex function requires 15GB memory. When using a full index to map reads to human/mouse genome, align and subjunc requires 17.8GB memory. To build a gapped index for human/mouse genome, buildindex function only requires 5.7GB memory. When using a gapped index to map reads to human/mouse genome, align requires 8.2GB memory and subjunc requires 8.8GB memory.

Sequences of reference genomes can be downloaded from public databases. For instance, primary assembly of human genome GRCh38/hg38 or mouse genome GRCm38/mm10 can be downloaded from the GENCODE database via the following links:

ftp://ftp.ebi.ac.uk/pub/databases/gencode/Gencode_human/release_28/GRCh38.primary_assembly.genome.fa.gz

ftp://ftp.ebi.ac.uk/pub/databases/gencode/Gencode_mouse/release_M18/GRCm38.primary_assembly.genome.fa.gz

Value

No value is produced but index files are written to the current working directory.

Author(s)

Wei Shi and Yang Liao

References

Yang Liao, Gordon K Smyth and Wei Shi (2019). The R package Rsubread is easier, faster, cheaper and better for alignment and quantification of RNA sequencing reads. *Nucleic Acids Research*, 47(8):e47. http://www.ncbi.nlm.nih.gov/pubmed/30783653

Yang Liao, Gordon K Smyth and Wei Shi (2013). The Subread aligner: fast, accurate and scalable read mapping by seed-and-vote. *Nucleic Acids Research*, 41(10):e108. http://www.ncbi.nlm.nih.gov/pubmed/23558742

See Also

align

Examples

```
# Build an index for the artifical sequence included in file 'reference.fa'
ref <- system.file("extdata","reference.fa",package="Rsubread")
buildindex(basename="./reference_index",reference=ref)
```

`cellCounts` *Map and quantify single cell RNA-seq data generated by 10X Genomics*

Description

Process raw 10X scRNA-seq data and generate UMI counts for each gene in each cell.

Usage

```
cellCounts(

    # input data
    index,
    sample,
    input.mode = "BCL",
    cell.barcode = NULL,

    # parameters used for mapping reads
    nsubreads = 15,
    minVotes = 1,
    maxMismatches = 10,
    minMappedLength = 1,

    # parameters used for assigning and counting UMIs
    annot.inbuilt = "mm39",
    annot.ext = NULL,
    isGTFAnnotationFile = FALSE,
    GTF.featureType = "exon",
    GTF.attrType = "gene_id",
    useMetaFeatures = TRUE,

    # user provided UMI cutoff for cell calling
    umi.cutoff = NULL,

    # number of threads
    nthreads = 10,

    # dealing with multi-mapping reads in the alignment step
    nBestLocations = 1,
    uniqueMapping = FALSE

    )
```

Arguments

index	A character string providing the base name of index files created for a reference genome by the buildindex function.
sample	A data frame or a character string providing sample-related information, including location of the data, sample names and index set names. See the Details section below for more details.
input.mode	A character string specifying the input mode. The supported input modes include BCL, FASTQ, FASTQ-dir and BAM. BCL is the BCL format of raw reads generated by the sequencers such as Illumina sequencers. FASTQ is the FASTQ format of sequencing reads. FASTQ-dir is a directory where FASTQ-format reads are saved. FASTQ-dir is useful for providing cellCounts the FASTQ data generated by bcl2fastq program or bamtofastq program (developed by 10X). BAM is the BAM format of mapped read data with cell barcodes and UMI sequences included (See the Details section below for more details). BCL by default.
cell.barcode	A character string giving the name of a text file (can be gzipped) that contains the set of cell barcodes used in sample preparation. If NULL, a cell barcode set will be determined for the input data by cellCounts based on the matching of cell barcodes sequences of the first 100,000 reads in the data with the three cell barcode sets used by 10X Genomics. NULL by default.
nsubreads	numeric value giving the number of subreads extracted from each read. 15 by default.
minVotes	numeric value giving the consensus threshold for reporting a hit. 1 by default.
maxMismatches	numeric value giving the maximum number of mis-matched bases allowed in the alignment. 10 by default. Mis-matches found in soft-clipped bases are not counted.
minMappedLength	numeric value giving the minimum mappable length in the read for reporting a hit. 1 by default.
annot.inbuilt	Specify an inbuilt annotation for UMI counting. See featureCounts for more details. mm39 by default.
annot.ext	Specify an external annotation for UMI counting. See featureCounts for more details. NULL by default.
isGTFAnnotationFile	See featureCounts for more details. FALSE by default.
GTF.featureType	See featureCounts for more details. exon by default.
GTF.attrType	See featureCounts for more details. gene_id by default.
useMetaFeatures	Specify if UMI counting should be carried out at the meta-feature level (eg. gene level). See featureCounts for more details. TRUE by default.
umi.cutoff	Specify a UMI count cutoff for cell calling. All the cells with a total UMI count greater than this cutoff will be called. If NULL, a bootstrapping procedure will be performed to determine this cutoff. NULL by default.

nthreads	A numeric value giving the number of threads used for read mapping and counting. 10 by default.
nBestLocations	A numeric value giving the maximum number of reported alignments for each multi-mapping read. 1 by default.
uniqueMapping	A logical value specifying if the multi-mapping reads should not be reported as mapped (i.e. reporting uniquely mapped reads only). FALSE by default.

Details

This function takes as input scRNA-seq reads generated by the 10X platform, maps them to the reference genome and then produces UMI (Unique Molecular Identifier) counts for each gene in each cell.

Sample demultiplexing, cell barcode demultiplexing and read deduplication are carried out before generating the UMI counts. cellCounts can process multiple datasets at the same time.

The sample information should be provided to cellCounts via the sample parameter. If the input format is BCL (ie. input.mode="BCL"), the provided sample information should include the location where the read data are stored, flowcell lanes used for sequencing, sample names and names of index sets used for indexing samples. These information should be saved to a data.frame object and then provided to the sample parameter. Below shows an example of this data frame:

```
InputDirectory Lane SampleName IndexSetName
/path/to/dataset1 1 Sample1 SI-GA-E1
/path/to/dataset1 1 Sample2 SI-GA-E2
/path/to/dataset1 2 Sample1 SI-GA-E1
/path/to/dataset1 2 Sample2 SI-GA-E2
/path/to/dataset2 1 Sample3 SI-GA-E3
/path/to/dataset2 1 Sample4 SI-GA-E4
/path/to/dataset2 2 Sample3 SI-GA-E3
/path/to/dataset2 2 Sample4 SI-GA-E4
...
```

It is compulsory to have the four column headers shown in the example above when generating this data frame for a 10X dataset. If more than one datasets are provided for analysis, the InputDirectory column should include more than one distinct directory. Note that this data frame is different from the Sample Sheet generated by the Illumina sequencer. The cellCounts function uses the index set names included in this data frame to generate an Illumina Sample Sheet and then uses it to demultiplex all the samples.

If the input format is FASTQ, a data.frame object containing the following three columns, BarcodeUMIFile, ReadFile and SampleName, should be provided to the sample parameter. Each row in the data frame represents a sample. The BarcodeUMIFile column should include names of FASTQ files containing cell barcode and UMI sequences for each read, and the ReadFile column should include names of FASTQ files containing genomic sequences for corresponding reads.

If the input format is FASTQ-dir, a character string, which includes the path to the directory where the FASTQ-format read data are stored, should be provided to the sample parameter. The data in this directory are expected to be generated by the bcl2fastq program or the bamtofastq program (a program developed by 10X).

Finally, if the input format is BAM, a data.frame object containing the following two columns, BAMFile and SampleName, should be provided to the sample parameter. Each row in the data frame represents a sample. The BAMFile column includes names of BAM files that contain read data with associated cell barcode and UMI sequences for the samples. The cell-barcodes and UMI sequences should be provided in the "CR" and "UR" tags in the BAM file. The Phred-encoded quality strings of the cell-barcodes and UMIs should be provided in the "CY" and "UY" tags. These tags can be found in BAM files generated by cellCounts and CellRanger.

Value

The cellCounts function returns a List object to R. It also outputs one BAM file for each sample. The BAM file includes location-sorted read mapping results. If the input mode is neither FASTQ nor FASTQ-dir, it also outputs three or four gzipped FASTQ files, which include cell barcode and UMI sequences (R1), sample index sequences (I1, and optionally I2 for dual-indexed samples) and the actual genomic sequences of the reads (R2), respectively.

The returned List object contains the following components:

counts
: a List object including UMI counts for each sample. Each component in this object is a matrix that contains UMI counts for a sample. Rows in the matrix are genes and columns are cells.

annotation
: a data.frame object containing a gene annotation. This is the annotation that was used for the assignment of UMIs to genes during quantification. Rows in the annotation are genes. Columns of the annotation include GeneID, Chr, Start, End and Length.

sample.info
: a data.frame object containing sample information and quantification statistics. It includes the following columns: SampleName, InputDirectory (if the input format is BCL), TotalCells, HighConfidenceCells (if umi.cutoff is NULL), RescuedCells (if umi.cutoff is NULL), TotalUMI, MinUMI, MedianUMI, MaxUMI, MeanUMI, TotalReads, MappedReads and AssignedReads. Each row in the data frame is a sample.

cell.confidence
: a List object indicating if a cell is a high-confidence cell or a rescued cell (low confidence). Each component in the object is a logical vector indicating which cells in a sample are high-confidence cells. cell.confidence is included in the output only if umi.cutoff is NULL.

Author(s)

Yang Liao and Wei Shi

See Also

buildindex, align, featureCounts

Examples

```
# Data used in this example (71MB) can be downloaded from https://cloudstor.aarnet.edu.au/plus/s/BhXfdhUXlR34nuZ
## Not run:
library(Rsubread)
```

```
buildindex("chr1", "hg38_chr1.fa.gz")
sample.sheet <- data.frame(BarcodeUMIFile="reads_R1.fastq.gz", ReadFile="reads_R2.fastq.gz", SampleName="Example
counts <- cellCounts("chr1", sample.sheet, input.mode="FASTQ", annot.inbuilt="hg38")

## End(Not run)
```

`detectionCall` *Determine Detection P Values for Each Gene in an RNA-seq Dataset*

Description

Use GC content adjusted background read counts to determine the detection p values for each gene

Usage

```
detectionCall(

    dataset,
    species = "hg",
    plot = FALSE)
```

Arguments

`dataset`	a character string giving the filename of a SAM format file, which is the output of read alignment.
`species`	a character string specifing the species. Options are hg and `mm`.
`plot`	logical indicating whether a density plot of detection p values will be generated.

Value

A data frame which includes detection p values and annotation information for each gene.

Author(s)

Zhiyin Dai and Wei Shi

detectionCallAnnotation

Generate Annotation Data Used for Calculating Detection P Values

Description

This is for internal use only. This function generates the annotation files for calculating p values in the detectionCall function.

Usage

```
detectionCallAnnotation(

    species = "hg",
    binsize = 2000)
```

Arguments

species	a character string specifying the species to analyase
binsize	an integer value specifying the bin-size of integenic region

Details

This is an internal function and should not be called by users directly.

It takes as input the annotation files produced by processExons function, calculates GC percentages for each exon of genes and also for intergenic regions and add GC info into the annotations. The new annotation data are then saved to files which can be used by detectionCall function for calling absolutely expressed genes.

Value

Two annotation files, which contain GC content for exons of genes and for intergenic regions respectively, are written to the current working directory. This function returns a NULL object.

Author(s)

Zhiyin Dai and Wei Shi

exactSNP *Accurately and Efficiently call SNPs*

Description

Measure background noises and perform Fisher's Exact tests to detect SNPs.

Usage

```
exactSNP(

    # basic input/output options
    readFile,
    isBAM = FALSE,
    refGenomeFile,
    SNPAnnotationFile = NULL,
    outputFile = paste0(readFile, ".exactSNP.VCF"),

    # fine tuning parameters
    qvalueCutoff = 12,
    minAllelicFraction = 0,
    minAllelicBases = 1,
    minReads = 1,
    maxReads = 1000000,
    minBaseQuality = 13,
    nTrimmedBases = 3,
    nthreads = 1)
```

Arguments

readFile	a character string giving the name of a file including read mapping results. This function takes as input a SAM file by default. If a BAM file is provided, the isBAM argument should be set to TRUE.
isBAM	logical indicating if the file provided via readFile is a BAM file. FALSE by default.
refGenomeFile	a character string giving the name of a file that includes reference sequences (FASTA format).
SNPAnnotationFile	a character string giving the name of a VCF-format file that includes annotated SNPs (the file can be uncompressed or gzip compressed). Such annotation can be downloaded from public databases such as dbSNP. Incorporating known SNPs into SNP calling has been found to be helpful. However note that the annotated SNPs may or may not be called for the sample being analyzed.
outputFile	a character string giving the name of the output file to be generated by this function. The output file includes all the reported SNPs (in VCF format). It includes discovered indels as well.

`qvalueCutoff` a numeric value giving the q-value cutoff for SNP calling at sequencing depth of 50X. `12` by default. The q-value is calcuated as -log10(p), where p is the p-value yielded from the Fisher's Exact test. Note that this function automatically adjusts the q-value cutoff for each chromosomal location according to its sequencing depth, based on this cutoff.

`minAllelicFraction` a numeric value giving the minimum fraction of allelic bases out of all read bases included at a chromosomal location required for SNP calling. Its value must be within `0` and `1`. `0` by default.

`minAllelicBases` an integer giving the minimum number of allelic (mis-matched) bases a SNP must have at a chromosomal location. `1` by default.

`minReads` an integer giving the minimum number of mapped reads a SNP-containing location must have (ie. the minimum coverage). `1` by default.

`maxReads` an integer specifying the maximum depth a SNP-containing location is allowed to have. `1000000` by default. Any location having number of mapped reads higher than this threshold will not be considered for SNP calling. This option is useful for removing PCR artefacts.

`minBaseQuality` a numeric value giving the minimum base quality score (Phred score) read bases should satisfy before being used for SNP calling. `13` by default(corresponding to base calling p value of 0.05). Read bases with quality scores less than 13 will be excluded from analysis.

`nTrimmedBases` a numeric value giving the number of bases trimmed off from each end of the read. `3` by default.

`nthreads` a numeric value giving the number of threads/CPUs used. `1` by default.

Details

This function takes as input a SAM/BAM format file, measures local background noise for each chromosomal location and then performs Fisher's exact tests to find statistically significant SNPs .

This function implements a novel algorithm for discovering SNPs. This algorithm is comparable with or better than existing SNP callers, but it is fast more efficient. It can be used to call SNPs for individual samples (ie. no control samples are required). Detail of the algorithm is described in a manuscript which is currently under preparation.

Value

No value is produced but but a VCF format file is written to the current working directory. This file contains detailed information for discovered SNPs including chromosomal locations, reference bases, alternative bases, read coverages, allele frequencies and p values.

Author(s)

Yang Liao and Wei Shi

featureCounts *Count Reads by Genomic Features*

Description

Assign mapped sequencing reads to specified genomic features.

Usage

```
featureCounts(files,

    # annotation
    annot.inbuilt = "mm39",
    annot.ext = NULL,
    isGTFAnnotationFile = FALSE,
    GTF.featureType = "exon",
    GTF.attrType = "gene_id",
    GTF.attrType.extra = NULL,
    chrAliases = NULL,

    # level of summarization
    useMetaFeatures = TRUE,

    # overlap between reads and features
    allowMultiOverlap = FALSE,
    minOverlap = 1,
    fracOverlap = 0,
    fracOverlapFeature = 0,
    largestOverlap = FALSE,
    nonOverlap = NULL,
    nonOverlapFeature = NULL,

    # Read shift, extension and reduction
    readShiftType = "upstream",
    readShiftSize = 0,
    readExtension5 = 0,
    readExtension3 = 0,
    read2pos = NULL,

    # multi-mapping reads
    countMultiMappingReads = TRUE,

    # fractional counting
    fraction = FALSE,

    # long reads
    isLongRead = FALSE,
```

```
    # read filtering
    minMQS = 0,
    splitOnly = FALSE,
    nonSplitOnly = FALSE,
    primaryOnly = FALSE,
    ignoreDup = FALSE,

    # strandness
    strandSpecific = 0,

    # exon-exon junctions
    juncCounts = FALSE,
    genome = NULL,

    # parameters specific to paired end reads
    isPairedEnd = FALSE,
    countReadPairs = TRUE,
    requireBothEndsMapped = FALSE,
    checkFragLength = FALSE,
    minFragLength = 50,
    maxFragLength = 600,
    countChimericFragments = TRUE,
    autosort = TRUE,

    # number of CPU threads
    nthreads = 1,

    # read group
    byReadGroup = FALSE,

    # report assignment result for each read
    reportReads = NULL,
    reportReadsPath = NULL,

    # miscellaneous
    maxMOp = 10,
    tmpDir = ".",
    verbose = FALSE)
```

Arguments

files	a character vector giving names of input files containing read mapping results. The files can be in either SAM format or BAM format. The file format is automatically detected by the function.
annot.inbuilt	a character string specifying an in-built annotation used for read summarization. It has five possible values "mm39", "mm10", "mm9", "hg38" and "hg19", corresponding to the NCBI RefSeq annotations for genomes 'mm39', 'mm10',

'mm9', 'hg38' and 'hg19', respectively. "mm39" by default. The in-built annotation has a SAF format (see below).

annot.ext A character string giving name of a user-provided annotation file or a data frame including user-provided annotation data. If the annotation is in GTF format, it can only be provided as a file. If it is in SAF format, it can be provided as a file or a data frame. See below for more details about SAF format annotation. If an annotation file is provided, it can be uncompressed or gzip compressed. Note that annot.ext will override annot.inbuilt if both provided.

isGTFAnnotationFile
logical indicating whether the annotation provided via the annot.ext argument is in GTF format or not. FALSE by default. This option is only applicable when annot.ext is not NULL.

GTF.featureType
a character string or a vector of character strings giving the feature type or types used to select rows in the GTF annotation which will be used for read summarization. "exon" by default. This argument is only applicable when isGTFAnnotationFile is TRUE. Feature types can be found in the third column of a GTF annotation.

GTF.attrType a character string giving the attribute type in the GTF annotation which will be used to group features (eg. exons) into meta-features (eg. genes). "gene_id" by default. This argument is only applicable when isGTFAnnotationFile is TRUE. Attributes can be found in the ninth column of a GTF annotation.

GTF.attrType.extra
a character vector specifying extra GTF attribute types that will also be included in the counting output. These attribute types will not be used to group features. NULL by default.

chrAliases a character string giving the name of a chromosome name alias file. This should be a two-column comma-delimited text file. Chromosome name aliases included in this file are used to match chr names in annotation with those in the reads. First column in the file should include chr names in the annotation and second column should include chr names in the reads. Chr names are case sensitive. No column header should be included in the file.

useMetaFeatures
logical indicating whether the read summarization should be performed at the feature level (eg. exons) or meta-feature level (eg. genes). If TRUE, features in the annotation (each row is a feature) will be grouped into meta-features, using their values in the GeneID column in the SAF-format annotation file or using the GTF.attrType attribute in the GTF-format annotation file, and then reads will be assiged to the meta-features instead of the features. See below for more details.

allowMultiOverlap
logical indicating if a read is allowed to be assigned to more than one feature (or meta-feature) if it is found to overlap with more than one feature (or meta-feature). FALSE by default.

minOverlap integer giving the minimum number of overlapped bases required for assigning a read to a feature (or a meta-feature). For assignment of read pairs (fragments), number of overlapping bases from each read in the same pair will be summed.

If a negative value is provided, then a gap of up to specified size will be allowed between read and the feature that the read is assigned to. 1 by default.

`fracOverlap` numeric giving minimum fraction of overlapping bases in a read that is required for read assignment. Value should be within range [0,1]. 0 by default. Number of overlapping bases is counted from both reads if paired end. Both this option and `minOverlap` option need to be satisfied before a read can be assigned.

`fracOverlapFeature`
numeric giving minimum fraction of bases included in a feature that is required for overlapping with a read or a read pair. Value should be within range [0,1]. 0 by default.

`largestOverlap` If TRUE, a read (or read pair) will be assigned to the feature (or meta-feature) that has the largest number of overlapping bases, if the read (or read pair) overlaps with multiple features (or meta-features).

`nonOverlap` integer giving the maximum number of bases in a read (or a read pair) that are allowed not to overlap the assigned feature. NULL by default (ie. no limit is set).

`nonOverlapFeature`
integer giving the maximum number of non-overlapping bases allowed in a feature during read assignment. NULL by default (ie. no limit is set).

`readShiftType` a character string indicating how reads should be shifted. It has four possible values including `upstream`, `downstream`, `left` and `right`. Read shifting is performed before read extension (`readExtension5` and `readExtension3`) and reduction (`read2pos`).

`readShiftSize` integer specifying the number of bases the reads will be shifted by. 0 by default. Negative value is not allowed.

`readExtension5` integer giving the number of bases extended upstream from 5' end of each read. 0 by default. Read extension is performed after read shifting but before read reduction. Negative value is not allowed.

`readExtension3` integer giving the number of bases extended downstream from 3' end of each read. 0 by default. Negative value is not allowed.

`read2pos` Specifying whether each read should be reduced to its 5' most base or 3' most base. It has three possible values: NULL, 5 (denoting 5' most base) and 3 (denoting 3' most base). Default value is NULL, ie. no read reduction will be performed. If a read is reduced to a single base, only that base will be considered for the read assignment. Read reduction is performed after read shifting and extension.

`countMultiMappingReads`
logical indicating if multi-mapping reads/fragments should be counted, TRUE by default. 'NH' tag is used to located multi-mapping reads in the input BAM/SAM files.

`fraction` logical indicating if fractional counts are produced for multi-mapping reads and/or multi-overlapping reads. FALSE by default. See below for more details.

`isLongRead` logical indicating if input data contain long reads. This option should be set to TRUE if counting Nanopore or PacBio long reads.

`minMQS` integer giving the minimum mapping quality score a read must satisfy in order to be counted. For paired-end reads, at least one end should satisfy this criteria. 0 by default.

splitOnly — logical indicating whether only split alignments (their CIGAR strings contain letter 'N') should be included for summarization. FALSE by default. Example split alignments are exon-spanning reads from RNA-seq data. useMetaFeatures should be set to FALSE and allowMultiOverlap should be set to TRUE, if the purpose of summarization is to assign exon-spanning reads to all their overlapping exons.

nonSplitOnly — logical indicating whether only non-split alignments (their CIGAR strings do not contain letter 'N') should be included for summarization. FALSE by default.

primaryOnly — logical indicating if only primary alignments should be counted. Primary and secondary alignments are identified using bit 0x100 in the Flag field of SAM/BAM files. If TRUE, all primary alignments in a dataset will be counted no matter they are from multi-mapping reads or not (ie. countMultiMappingReads is ignored).

ignoreDup — logical indicating whether reads marked as duplicates should be ignored. FALSE by default. Read duplicates are identified using bit Ox400 in the FLAG field in SAM/BAM files. The whole fragment (read pair) will be ignored if paired end.

strandSpecific — an integer vector indicating if strand-specific read counting should be performed. Length of the vector should be either 1 (meaning that the value is applied to all input files), or equal to the total number of input files provided. Each vector element should have one of the following three values: 0 (unstranded), 1 (stranded) and 2 (reversely stranded). Default value of this parameter is 0 (ie. unstranded read counting is performed for all input files).

juncCounts — logical indicating if number of reads supporting each exon-exon junction will be reported. Junctions are identified from those exon-spanning reads in input data. FALSE by default.

genome — a character string giving the name of a FASTA-format file that includes the reference sequences used in read mapping that produced the provided SAM/BAM files. NULL by default. This argument should only be used when juncCounts is TRUE. Note that providing reference sequences is optional when juncCounts is set to TRUE.

isPairedEnd — A logical scalar or a logical vector, indicating whether libraries contain paired-end reads or not. FALSE by default. For any library that contains paired-end reads, the 'countReadPairs' parameter controls if read pairs or reads should be counted.

countReadPairs — A logical scalar or a logical vector, indicating if read pairs, instead of reads, should be counted for paired-end read data. TRUE by default. This parameter is only applicable for paired-end data. For single-end data, it is ignored.

requireBothEndsMapped — logical indicating if both ends from the same fragment are required to be successfully aligned before the fragment can be assigned to a feature or meta-feature. This parameter is only appliable when isPairedEnd is TRUE.

checkFragLength — logical indicating if the two ends from the same fragment are required to satisify the fragment length criteria before the fragment can be assigned to a feature or meta-feature. This parameter is only appliable when isPairedEnd is TRUE. The fragment length criteria are specified via minFragLength and maxFragLength.

`minFragLength` integer giving the minimum fragment length for paired-end reads. 50 by default.

`maxFragLength` integer giving the maximum fragment length for paired-end reads. 600 by default. minFragLength and maxFragLength are only applicable when isPairedEnd is TRUE. Note that when a fragment spans two or more exons, the observed fragment length might be much bigger than the nominal fragment length.

`countChimericFragments` logical indicating whether a chimeric fragment, which has its two reads mapped to different chromosomes, should be counted or not. TRUE by default.

`autosort` logical specifying if the automatic read sorting is enabled. This option is only applicable for paired-end reads. If TRUE, reads will be automatically sorted by their names if reads from the same pair are found not to be located next to each other in the input. No read sorting will be performed if there are no such reads found.

`nthreads` integer giving the number of threads used for running this function. 1 by default.

`byReadGroup` logical indicating if read counting will be performed for each individual read group. FALSE by default.

`reportReads` output detailed read assignment results for each read (or fragment if paired end). The detailed assignment results can be saved in three different formats including CORE, SAM and BAM (note that these values are case sensitive). Default value of this option is NULL, indicating not to report assignment results for reads. CORE format represents a tab-delimited text file that contains four columns including read name, status (assigned or the reason if not assigned), number of targets and target list. A target is a feature or a meta-feature. Items in the target lists is separated by comma. If a read is not assigned, its number of targets will be set as -1. For each input file, a text file is generated and its name is the input file name added with '.featureCounts'. When SAM or BAM format is specified, the detailed assignment results will be saved to SAM and BAM format files. Names of generated files are the input file names added with '.featureCounts.sam' or '.featureCounts.bam'. Three tags are used to describe read assignment results: XS, XN and XT. Tag XS gives the assignment status. Tag XN gives number of targets. Tag XT gives comma separated target list.

`reportReadsPath` a character string specifying the directory where files including detailed assignment results are saved. If NULL, the results will be saved to the current working directory.

`maxMOp` integer giving the maximum number of 'M' operations (matches or mis-matches) allowed in a CIGAR string. 10 by default. Both 'X' and '=' operations are treated as 'M' and adjacent 'M' operations are merged in the CIGAR string.

`tmpDir` a character string specifying the directory under which intermediate files are saved (later removed). By default, current working directory is used.

`verbose` logical indicating if verbose information for debugging will be generated. This may include information such as unmatched chromosomes/contigs between reads and annotation.

Details

featureCounts is a general-purpose read summarization function that can assign mapped reads from genomic DNA and RNA sequencing to genomic features or meta-features.

The function takes as input a set of SAM or BAM files containing read mapping results. The files might be generated by align or subjunc or any suitable aligner.

featureCounts accepts two annotation formats to specify the genomic features: SAF (Simplified Annotation Format) or GTF/GFF. The GTF/GFF format is specified at https://genome.ucsc.edu/FAQ/FAQformat.html. SAF is a Simplified Annotation Format with five columns. It can be either a data.frame or a tab-delimited text file in the following style:

```
GeneID Chr Start End Strand
497097 chr1 3204563 3207049 -
497097 chr1 3411783 3411982 -
497097 chr1 3660633 3661579 -
100503874 chr1 3637390 3640590 -
100503874 chr1 3648928 3648985 -
100038431 chr1 3670236 3671869 -
...
```

SAF annotation includes the following five required columns: GeneID, Chr, Start, End and Strand. The GeneID column includes identifiers of features. These identifiers can be integer or character string. The Chr column includes chromosome names of features and these names should match the chromosome names includes in the provided SAM/BAM files. The Start and End columns include start and end coordinates of features, respectively. Both start and end coordinates are inclusive. The Strand column indicates the strand of features ("+" or "-"). Column names in a SAF annotation should be the same as those shown in the above example (case insensitive). Columns can be in any order. Extra columns are allowed to be added into the annotation.

In-built annotations, which were generated based on NCBI RefSeq gene annotations, are provided to faciliate convenient read summarization. We provide in-built annotations for the following genomes: "hg38", "hg19", "mm39", "mm10" and "mm9". The content of in-built annotations can be accessed via the getInBuiltAnnotation function. These annotations have a SAF format.

The in-built annotations are a modified version of NCBI RefSeq gene annotations. For in-built annotations "hg38", "hg19", "mm10" and "mm9", their original RefSeq annotations were downloaded from NCBI RefSeq MapView.

We then used these downloaded annotations to create in-built annotations. For each gene, we used its CDS (coding DNA sequence) and UTR (untranslated) regions provided in the original annotation to construct a list of exons, by merging and concatenating all CDSs and UTRs belonging to the same gene. Exons within each gene include all chromosomal bases included in the original CDS and UTR regions, but they include each base only once (they might be included multiple times in the original CDSs and UTRs). Also, exons within the same gene do not overlap with each other.

For the in-built annotation "mm39", its original RefSeq annotation was downloaded from https://ftp.ncbi.nlm.nih.gov/genomes/all/annotation_releases/10090/109/GCF_000001635.27_GRCm39/GCF_000001635.27_GRCm39_genomic.gtf.gz. This is a GTF-format annotation. We changed the format of chromosome names from "NC_xxxxxx" to "chr" followed by a number (eg. "NC_000067.7" changed to "chr1"). The flattenGTF function was then used to merge overlapping exons within the same gene to generate the inbuilt annotation. Entrez gene identifiers were included in this annotation.

Users may provide an external annotation for read summarization via the annot.ext argument. If the external annotation is in SAF format, it can be provdied as either a data.frame or a tab-delimited text file with proper column names included. If it is in GTF/GFF format, it should be provided as a file only (and isGTFAnnotationFile should be set to TRUE).

featureCounts function uses the GTF.attrType attribute in a GTF/GFF annotation to group features to form meta-features when performing read summarization at meta-feature level.

The argument useMetaFeatures specifies whether read summarization should be performed at feature level or at meta-feature level. A feature represents a continuous genomic region and a meta-feature is a group of features. For instance, an exon is a feature and a gene comprising one or more exons is a meta-feature. To assign reads to meta-features, featureCounts firstly groups into meta-features the features that have the same gene identifiers. featureCounts looks for gene identifiers in GeneID column of a SAF annotation or by using GTF.attrType attribute in a GTF/GFF annotation. Then for each read featureCounts searches for meta-features that have at least one feature that overlaps with the read. A read might be found to overlap with more than one feature within the same meta-feature (eg. an exon-spanning read overlaps with more than one exon from the same gene), however this read will still be counted only once for the meta-feature.

RNA-seq reads are often summarized to meta-features to produce read counts for genes. Further downstream analysis can then be carried out to discover differentially expressed genes. Feature-level summarization of RNA-seq data often yields exon-level read counts, which is useful for investigating alternative splicing of genes.

featureCounts provides multiple options to count multi-mapping reads (reads mapping to more than one location in the reference genome). Users can choose to ignore such reads by setting countMultiMappingReads to FALSE, or fully count every alignment reported for a multi-mapping read by setting countMultiMappingReads to TRUE (each alignment carries 1 count), or count each alignment fractionally by setting both countMultiMappingReads and fraction to TRUE (each alignment carries 1/x count where x is the total number of alignments reported for the read).

featureCounts also provides multiple options to count multi-overlapping reads (reads overlapping with more than one meta-feature when conducting meta-feature-level summarization or overlapping with more than one feature when conducting feature-level summarization). Users can choose to ignore such reads by setting allowMultiOverlap to FALSE, or fully count them for each overlapping meta-feature/feature by setting allowMultiOverlap to TRUE (each overlapping meta-feature/feature receives a count of 1 from a read), or assign a fractional count to each overlapping meta-feature/feature by setting both allowMultiOverlap and fraction to TRUE (each overlapping meta-feature/feature receives a count of 1/y from a read where y is the total number of meta-features/features overlapping with the read).

If a read is both multi-mapping and multi-overlapping, then each overlapping meta-feature/feature will receive a fractional count of 1/(x*y) if countMultiMappingReads, allowMultiOverlap and fraction are all set to TRUE.

When isPairedEnd is TRUE, fragments (pairs of reads) instead of reads will be counted. featureCounts function checks if reads from the same pair are adjacent to each other (this could happen when reads were for example sorted by their mapping locations), and it automatically reorders those reads that belong to the same pair but are not adjacent to each other in the input read file.

Value

A list with the following components:

`counts` a data matrix containing read counts for each feature or meta-feature for each library.

`counts_junction (optional)`
a data frame including primary and secondary genes overlapping an exon junction, position information for the left splice site ('Site1') and the right splice site ('Site2') of a junction (including chromosome name, coordinate and strand) and number of supporting reads for each junction in each library. Both primary and secondary genes overlap at least one of the two splice sites of a junction. Secondary genes do not overlap more splice sites than the primary gene. When the primary and secondary genes overlap same number of splice sites, the gene with the smallest leftmost base position is selected as the primary gene. For each junction, no more than one primary gene is reported but there might be more than one secondary genes reported. If a junction does not overlap any genes, it has a missing value in the fields of primary gene and secondary gene. Note that this data frame is only generated when `juncCounts` is `TRUE`.

`annotation` a data frame with six columns including `GeneID`, `Chr`, `Start`, `End` and `Length`. When read summarization was performed at feature level, each row in the data frame is a feature and columns in the data frame give the annotation information for the features. When read summarization was performed at meta-feature level, each row in the data frame is a meta-feature and columns in the data frame give the annotation information for the features included in each meta feature except the `Length` column. For each meta-feature, the `Length` column gives the total length of genomic regions covered by features included in that meta-feature. Note that this length will be less than the sum of lengths of features included in the meta-feature when there are features overlapping with each other. Also note the `GeneID` column gives Entrez gene identifiers when the in-built annotations are used.

`targets` a character vector giving sample information.

`stat` a data frame giving number of successfully assigned reads in each library and also number of unassigned reads falling in each filtering category (unmapped, read type, singleton, mapping quality, chimera, fragment length, duplicate, multi-mapping, secondary alignment, split alignment, no features, overlapping length and assignment ambiguity). See Users Guide for more details.

Author(s)

Wei Shi and Yang Liao

References

Yang Liao, Gordon K Smyth and Wei Shi (2019). The R package Rsubread is easier, faster, cheaper and better for alignment and quantification of RNA sequencing reads. *Nucleic Acids Research*, 47(8):e47. http://www.ncbi.nlm.nih.gov/pubmed/30783653

Yang Liao, Gordon K Smyth and Wei Shi (2014). featureCounts: an efficient general-purpose program for assigning sequence reads to genomic features. *Bioinformatics*, 30(7):923-30. http://www.ncbi.nlm.nih.gov/pubmed/24227677

See Also

getInBuiltAnnotation

Examples

```
## Not run:
# Summarize SAM format single-end reads using built-in RefSeq annotation for mouse genome mm39:
featureCounts(files="mapping_results_SE.sam",annot.inbuilt="mm39")

# Summarize single-end reads using a user-provided GTF annotation file:
featureCounts(files="mapping_results_SE.sam",annot.ext="annotation.gtf",
isGTFAnnotationFile=TRUE,GTF.featureType="exon",GTF.attrType="gene_id")

# Summarize single-end reads using 5 threads:
featureCounts(files="mapping_results_SE.sam",nthreads=5)

# Summarize BAM format single-end read data:
featureCounts(files="mapping_results_SE.bam")

# Perform strand-specific read counting (strandSpecific=2 if reversely stranded):
featureCounts(files="mapping_results_SE.bam",strandSpecific=1)

# Summarize paired-end reads and counting fragments (instead of reads):
featureCounts(files="mapping_results_PE.bam",isPairedEnd=TRUE)

# Count fragments satisfying the fragment length criteria, eg. [50bp, 600bp]:
featureCounts(files="mapping_results_PE.bam",isPairedEnd=TRUE,
checkFragLength=TRUE,minFragLength=50,maxFragLength=600)

# Count fragments that have both ends successfully aligned without checking the fragment length:
featureCounts(files="mapping_results_PE.bam",isPairedEnd=TRUE,requireBothEndsMapped=TRUE)

# Exclude chimeric fragments from fragment counting:
featureCounts(files="mapping_results_PE.bam",isPairedEnd=TRUE,countChimericFragments=FALSE)

## End(Not run)
```

---

findCommonVariants *Finding the Common Variants Among All Input VCF Files*

Description

The common variants (inc. SNPs and indels) among all the input files are found. A data frame containing these common variants is returned. The data frame has a similar format as VCF files.

Usage

```
findCommonVariants(VCF_files)
```

Arguments

VCF_files	a character vector giving the names of VCF format files.

Details

This function loads all variants (SNPs and indels) from the input VCF files, and find the common variants that are reported in all the VCF files. If a variant record in a input VCF file has multiple alternative sequences (split by ','), each alternative sequence is treated as a single variant. Two variants in two VCF files are the same only if their genomic locations, their reference sequences, their alternative sequences and their variant types are identical.

This function currently does not support other types of variants other than SNPs and indels.

There are eight columns in the returned data frame: chromosome name, position, identity, reference sequence, alternative sequence, quality, filter and extra information. The input may have more columns; these columns are not included in the data frame. If the identity, the quality, the filter and the extra information for the same variant are different among the input VCF files, those information associated with the lowest quality value of this variant among the VCF files is reported in the resulted data frame. For example, if an SNP on chrX:12345 (A=>G) is reported in all the three input VCF files, and the quality scores in the three VCF files are 100, 10, 50 respectively, the identity, the quality, the filter and the extra information in the second VCF file are reported in the resulted data frame for this SNP.

Value

A data frame containing the common variants among all the input VCF files is returned. The first eight columns are: chromosome name, position, identity, reference sequence, alternative sequence, quality, filter and extra information.

If there are not any common variants, this function returns an NA value.

Author(s)

Yang Liao and Wei Shi

Examples

```
## Not run:
# finding the common variants between to input VCF files: a.vcf and b.vcf
library(Rsubread)
findCommonVariants(c('a.vcf','b.vcf'))

## End(Not run)
```

flattenGTF *Flatten Features in GTF or GFF Annotation Files*

Description

Convert a GTF/GFF annotation to SAF format and flatten overlapping features.

Usage

```
flattenGTF(

    # basic input/output options
    GTFfile,
    GTF.featureType = "exon",
    GTF.attrType = "gene_id",

    # the option specifying the merging algorithm
    method = "merge")
```

Arguments

GTFfile	a character string giving the name of a GTF file as input.
GTF.featureType	a character string giving the feature type used to select rows in a GTF annotation. "exon" by default. Feature types can be found in the third column of a GTF annotation.
GTF.attrType	a character string giving the attribute type in a GTF annotation which will be used to group features. "gene_id" by default. Attributes can be found in the ninth column of a GTF annotation.
method	a character string specifying the method for how to flatten the GTF file. The method can be either "merge" or "chop". "merge" by default. See the details section for more information.

Details

This function locates features in a GTF annotation via GTF.featureType and then groups them into meta-features via GTF.attrType.

When method="merge", the overlapping features found in a meta-feature will be merged to form a single continuous feature encompassing all the overlapping features. If method="chop", overlapping features will be chopped into multiple non-overlapping bins. Here is an example to illustrate the differences between the two methods. Say there are three exons belonging to the same gene and the coordinates of these exons are [100, 200], [150, 500] and [250, 400]. When running on "merge" mode, a single exon will be returned for this gene and the coordinates of this exon are [100, 500], representing the union of the three original exons. When running on "chop" mode, five non-overlapping bins will be returned, including [100, 149], [150, 200], [201, 249], [250, 400] and

[401, 500]. Intervals of these bins are determined by start and end coordinates of the three original exons (100, 150, 200, 250, 400 and 500).

Output of this function is a SAF format annotation which can be fed to featureCounts function for read counting. Description to SAF format annotation can also be found in featureCounts.

Value

A data.frame including a SAF format annotation in which features included in each meta-feature are all distinct.

Author(s)

Yang Liao and Wei Shi

See Also

featureCounts

---

getInBuiltAnnotation *Retrieve In-Built Annotations*

Description

Retrieve an in-built annotation and save it to a data frame.

Usage

```
getInBuiltAnnotation(annotation = "mm39")
```

Arguments

annotation	a character string specifying the in-built annotation to be retrieved. It has five possible values mm39, mm10, mm9, hg38 and hg19, corresponding to the NCBI RefSeq annotations for genomes 'mm39', 'mm10', 'mm9', 'hg38' and 'hg19', respectively. mm39 by default.

Details

The featureCounts read summarization function provides in-built annotations for conveniently summarizing reads to genes or exons, and this function allows users to have access to those in-built annotations.

For more information about these annotations, please refer to the help page for featureCounts function.

Value

A data frame with five columns including GeneID, Chr, Start, End and Strand.

Author(s)

Wei Shi

See Also

featureCounts

Examples

```
x <- getInBuiltAnnotation("hg38")
head(x)
```

---

processExons *Obtain Chromosomal Coordiates of Each Exon Using NCBI Annotation*

Description

This is for internal use. It converts the NCBI exon annotations into data files for use in the detectionCall function.

Usage

```
processExons(

    filename = "human_seq_gene.md",
    species = "hg")
```

Arguments

filename	a character string giving the name of input .md file (NCBI annotation file)
species	a character string specifying the species

Details

This is an internal function and should not be called by users directly.

It processes annotation of genes in human genome GRCh37/hg19 or mouse genome GRCm37/mm9. The annotation is available for download from the following links (these annotations include chromosomal coordinates of UTR and CDS regions of genes):

ftp://ftp.ncbi.nlm.nih.gov/genomes/H_sapiens/ARCHIVE/BUILD.37.2/mapview/seq_gene.md.gz

ftp://ftp.ncbi.nlm.nih.gov/genomes/M_musculus/ARCHIVE/BUILD.37.2/mapview/seq_gene.md.gz

This function finds the chromosomal coordinates of intergenic regions (regions between neighbouring genes) and then outputs them to a file. It also outputs to a file chromosomal coordinates of exons of genes by concatenating UTRs with CDSs and merging overlapping CDSs within each gene. The generated annotation files will then be used by detectionCallAnnotation function to produce annotation data required by detectionCall function.

Value

Two annotation files are written to the current working directory. This function returns a NULL object.

Author(s)

Zhiyin Dai and Wei Shi

---

promoterRegions *Generate Annotation for Promoter Regions of Genes*

Description

Create a SAF data-frame of genewise promoter regions.

Usage

```
promoterRegions(

    annotation = "mm39",
    upstream = 3000L,
    downstream = 2000L)
```

Arguments

annotation	a data.frame containing gene annotation in SAF format or a character string giving the name of a genome with built-in annotation. If using built-in annotation, the character string should be one of the following: mm10, mm9, hg38 or hg19 corresponding to the NCBI RefSeq annotations for the genomes 'mm10', 'mm9', 'hg38' and 'hg19', respectively.
upstream	an integer giving the number of upstream bases that will be inclued in the promoter region generated for each gene. These bases are taken immediately upstream (5' end) from transcriptional start site of each gene.
downstream	an integer giving the number of downstream bases that will be inclued in the promoter region generated for each gene. These bases are taken immediately downstream (3' end) from transcriptional start site of each gene.

Details

This function takes as input a SAF format gene annotation and produces a SAF format data.frame containing the chromosomal coordinates of the specified promoter region for each gene. See featureCounts for definition of the SAF format.

Regardless of the upstream or downstream values, the downstream end of the region never extends past the end of the gene and the upstream end never extends outside the relevant chromosome. Setting downstream to an infinite or large value will cause the body of each gene to be included.

Value

A SAF format data.frame with columns GeneID, Chr, Start, End and Strand.

Author(s)

Gordon K Smyth

See Also

featureCounts, getInBuiltAnnotation

Examples

```
# To get whole gene bodies (from TSS to TES) for the latest mouse genome:
x <- promoterRegions("mm39", upstream = 0, downstream = Inf)
head(x)
```

---

propmapped *Calculate the Proportion of Mapped Reads or Fragments in SAM/BAM Files*

Description

Number of mapped reads/fragments will be counted and fraction of such reads/fragments will be calculated.

Usage

```
propmapped(

    files,
    countFragments = TRUE,
    properlyPaired = FALSE,
    verbose = FALSE)
```

Arguments

files	a character vector giving the names of SAM/BAM format files. Format of input files is automatically determined by the function.
countFragments	logical indicating whether reads or fragments (read pairs) should be counted. If TRUE, fragments will be counted when paired-end read data are provided. This function automatically detects if the data are single end or paired end. For single end data, each read is treated as a fragment and therefore the value of this parameter should be set to TRUE.
properlyPaired	logical indicating if only properly paired reads will be counted. This is only applicable for paired end data. FALSE by default.
verbose	logical indicating if verbose information should be displayed.

Details

This function uses the FLAG field in the SAM/BAM to look for mapped reads and count them. Reads/fragments, which have more than one reported location, will be reported only once.

When counting single end reads, counting reads has the same meaning as counting fragments (the results are identical).

Note that align and subjunc return the same mapping statistics as propmapped as a by-product of the alignment process, so saving the output from align or subjunc will be faster and more informative than running propmapped separately.

Value

A data frame containing the total number of reads, number of mapped reads and proportion of mapped reads for each library.

Author(s)

Wei Shi and Yang Liao

Examples

```
# build an index using the sample reference sequence provided in the package
# and save it to the current directory
ref <- system.file("extdata","reference.fa",package="Rsubread")
buildindex(basename="./reference_index",reference=ref)

# align the sample read data provided in this package to the sample reference
# and save the mapping results to the current directory
reads <- system.file("extdata","reads.txt.gz",package="Rsubread")
stat <- align(index="./reference_index",readfile1=reads,output_file="./Rsubread_alignment.BAM")
stat

# get the percentage of successfully mapped reads
propmapped("./Rsubread_alignment.BAM")
```

---

qualityScores *Extract Quality Score Data in a Sequencing Read Dataset*

Description

Extract quality strings and convert them to Phred scores for generating boxplots.

Usage

```
qualityScores(

    # basic input/output options
    filename,
```

```
    input_format = "gzFASTQ",
    offset = 33,
    nreads = 10000)
```

Arguments

filename	a character string giving the name of an input file containing sequence reads.
input_format	a character string specifying format of the input file. gzFASTQ (gzipped FASTQ) by default. Acceptable formats include gzFASTQ, FASTQ, SAM and BAM. Character string is case insensitive.
offset	a numeric value giving the offset added to the base-calling Phred scores. Possible values include 33 and 64. By default, 33 is used.
nreads	a numeric value giving the number of reads from which quality scores are extracted. 10000 by default. A value of -1 indicates that quality scores will be extracted from all the reads.

Details

Quality scores of read bases are represented by ASCII characters in next-gen sequencing data. This function extracts the quality characters from each base in each read and then converts them to Phred scores using the provided offset value (offset).

If the total number of reads in a dataset is n, then every n/nreads read is extracted from the input data.

Value

A data matrix containing Phred scores for read bases. Rows in the matrix are reads and columns are base positions in each read.

Author(s)

Wei Shi, Yang Liao and Zhiyin Dai

Examples

```
reads <- system.file("extdata","reads.txt.gz",package="Rsubread")
x <- qualityScores(filename=reads,offset=64,nreads=1000)
x[1:10,1:10]
```

---

removeDupReads *Remove Sequencing Reads Mapped to Identical Locations*

Description

Remove reads which are mapped to identical locations, using mapping location of the first base of each read.

Usage

```
removeDupReads(

    # basic input/output options
    inputFile,
    threshold = 50,
    outputFile,
    outputFormat = "BAM")
```

Arguments

inputFile	a character string giving the name of a SAM or BAM format input file.
threshold	a numeric value giving the threshold for removing duplicated reads, 50 by default. Reads will be removed if they are found to be duplicated equal to or more than threshold times.
outputFile	a character string giving the base name of output files.
outputFormat	a character string giving the format of the output file. BAM and SAM are accepted. BAM by default.

Details

This function uses the mapping location of first base of each read to find duplicated reads. Reads are removed if they are duplicated more than threshold number of times.

Value

This function generates a BAM or SAM file including the remaining reads after duplicate removal.

Author(s)

Yang Liao and Wei Shi

---

repair *Re-Order Paired-End Reads to Place Reads*

Description

Fast re-ordering of paired-end reads using read names and mapping locations.

Usage

```
repair(

    inFiles,
    inFormat = "BAM",
    outFiles = paste(inFiles,"repair",sep="."),
```

```
addDummy = TRUE,
fullData = TRUE,
compress = FALSE,
nthreads = 8)
```

Arguments

inFiles	a character vector giving names of input files. These files are typically location-sorted BAM files.
inFormat	a character string specifying format of input files. Supported formats include BAM and SAM.
outFiles	a character string giving names of output files. Re-ordered reads are saved to BAM-format files.
addDummy	logical indicating if a dummy read will be added to each singleton read which has a missing pair in the input. TRUE by default.
fullData	logical indicating if sequences and base-calling quality scores of reads will be included in the output. TRUE by default.
compress	logical indicating if compression should be turned on when generating BAM output. FALSE by default.
nthreads	a numeric value giving number of CPU threads. 8 by default.

Details

This function takes as input paired-end BAM or SAM files, re-orders reads to make reads from the same pair be adjacent to each other and outputs the re-ordered reads into new BAM files.

The function makes use of both read names and mapping information of reads (eg. mapping coordinates) to identify reads belonging to the same pair. This makes sure that all paired-end reads are correctly re-ordered, especially those multi-mapping read pairs that include more than one reported alignment in the input.

The BAM files produced by this function are compatible with featureCounts, meaning that no further re-ordering needs to be performed by featureCounts.

Value

No value is produced but BAM files with re-ordered reads are written to the current working directory.

Author(s)

Wei Shi and Yang Liao

RsubreadUsersGuide *View Rsubread Users Guide*

Description

Users Guide for Rsubread and Subread

Usage

```
RsubreadUsersGuide()
```

Details

The Subread/Rsubread Users Guide provides detailed description to the functions and programs included in the Subread and Rsubread software packages. It also includes case studies for analyzing next-gen sequencing data.

The Subread package is written in C and it can be downloaded from http://subread.sourceforge.net. The Rsubread package provides R wrappers functions for many of the programs included in Subread package.

Value

Character string giving the file location.

Author(s)

Wei Shi

See Also

vignette

sam2bed *Convert a SAM Format File to a BED File*

Description

This function performs SAM to BED conversion. Each read in the SAM input is converted into an interval in the BED file.

Usage

```
sam2bed(

    samfile,
    bedfile,
    readlen)
```

Arguments

samfile	a character string giving the name of input file. Input format should be in SAM format.
bedfile	a character string giving the name of output file. Output file is in BED format.
readlen	a numeric value giving the length of reads included in the input file.

Details

This function converts a SAM format file to a BED format file, which can then be displayed in a genome browser like UCSC genome browser, IGB, IGV. This function does not find the read length from the SAM file, but requires the user to specify the read length.

Value

No value is produced but a BED format file is written to the current working directory. This file contains six columns including chromosomal name, start position, end position, name('.'), mapping quality score and strandness.

Author(s)

Wei Shi

---

simReads *Generate Simulated Reads from a Set of Transcripts*

Description

The simReads function generates simulated reads from a set of transcripts. Each transcript has a predefined abundance in the output.

Usage

```
simReads(

    # the transcript database and the wanted abundances
    transcript.file,
    expression.levels,

    # the name of the output
    output.prefix,

    # options on the output
    library.size = 1000000,
    read.length = 75,
    truth.in.read.names = FALSE,
```

```
    # simulating sequencing errors
    simulate.sequencing.error = TRUE,
    quality.reference = NULL,

    # parameters for generating paired-end reads.
    paired.end = FALSE,
    fragment.length.min = 100,
    fragment.length.max = 500,
    fragment.length.mean = 180,
    fragment.length.sd = 40,

    # manipulating transcript names
    simplify.transcript.names = FALSE)

scanFasta(

    # the file containing the transcript database
    transcript.file,

    # manipulating transcript names
    simplify.transcript.names = FALSE,

    # miscellaneous options
    quiet = FALSE)
```

Arguments

`transcript.file`
: a character string giving the name of a file that contains the transcript names and sequences. The format can be FASTA or gzipped-FASTA. No duplicate sequence name is allowed in the file, and duplicate sequences will trigger warning messages.

`expression.levels`
: a numeric vector giving the relative expression levels of the transcripts, in the order of transcripts in `transcript.file`. The sum of the values must be positive, and no negative value is allowed.

`output.prefix`
: a character string giving the basename of all the output files.

`library.size`
: a numeric value giving the number of reads or read-pairs to be generated. One million by default.

`read.length`
: a anumeric value giving the length of each read in the output. Maximum length is 250bp. Transcripts that are shorter than the read length will not be used for generating simulated reads. 75 by default.

`truth.in.read.names`
: logical indicating if the true mapping location of reads or read-pairs should be encoded into the read names. `FALSE` by default.

`simulate.sequencing.error`
: logical indicating if sequencing errors should be simulated in the output reads. If `TRUE`, the `quality.reference` parameter must be specified unless the output

read length is 100-bp or 75-bp. If the output read length is 100-bp or 75-bp, the `quality.reference` parameter can be optionally omitted, and the function will use its inbuilt quality strings.

`quality.reference`
: a character string giving the name of a file that contains one or multiple sequencing quality strings in the Phred+33 format. The sequencing quality strings must have the same length as `read.length`.

`paired.end`
: logical indicating if paired-end reads should be generated. `FALSE` by default.

`fragment.length.min`
: a numeric value giving the minimum fragment length. The minimum fragment length must be equal to or greater than the output read length. 100 by default.

`fragment.length.max`
: a numeric value giving the maximum fragment length. 500 by default.

`fragment.length.mean`
: a numeric value giving the mean of fragment lengths. 180 by default.

`fragment.length.sd`
: a numeric value giving the standard deviation of fragment lengths. The fragment lengths are drawn for a truncated gamma distribution that is defined by `fragment.length.min`, `fragment.length.max`, `fragment.length.mean` and `fragment.length.sd`. 40 by default.

`simplify.transcript.names`
: logical indicating if transcript names should be simplified. If `TRUE`, the transcript names are truncated to the first `|` or space. `FALSE` by default.

`quiet`
: logical indicating if the warning message for repeated sequences should be suppressed in the `scanFasta` function. `FALSE` by default.

Details

`simReads` generates simulated reads from a set of transcript sequences at specified abundances. The input includes a transcript file in FASTA or gzipped-FASTA format, and a numeric vector that describes the wanted abundance for each transcript. The output of this function is one or two gzipped FASTQ files that contain the simulated reads or read-pairs. To have reads generated from it, a transcript must have a length equal to or greater than the output read length, and also equal to or greater than the minimum fragment length in case of paired-end reads.

When generating paired-end reads, the fragment lengths are drawn from a truncated normal distribution with the mean and standard deviation specified in `fragment.length.mean` and `fragment.length.sd`; the minimum and maximum fragment lengths are specified in `fragment.length.min` and `fragment.length.max`.

Substitution sequencing errors can be simulated in the reads by emulating the sequencing quality of a real High-Throughput Sequencing sample. When `simulate.sequencing.error = TRUE` and a set of Phred+33 encoded quality strings are provided to `simReads`, it randomly chooses a quality string for each output read, and substitutes the read bases with random base values at the probabilities described in the quality string. This function has inbuilt quality strings for generating 100-bp and 75-bp long reads, hence the `quality.reference` can be optionally omitted when `read.length` is 100 or 75.

The `scanFasta` function checks and processes the FASTA or gzipped-FASTA file. It scans through the file that defines the transcript sequences and returns a data.frame of transcript names and sequence lengths. It additionally checks the transcript sequences for uniqueness.

Value

simReads writes a FASTQ file, or a pair of FASTQ files if paired.end, containing the simulated reads. It also returns a data.frame with three columns:

TranscriptID	character	transcript name
Length	integer	length of transcript sequence
Count	integer	simulated read count

scanFasta returns a data.frame with six columns:

TranscriptID	character	transcript name
Length	integer	length of transcript sequence
MD5	character	MD5 digest of the transcript sequence
Unique	logical	is this transcript's sequence unique in the FASTA file?
Occurrence	integer	number of times this transcript's sequence was observed
Duplicate	logical	this transcript's sequence is a duplicate of a previous sequence.

Note that selecting transcripts with Duplicate == FALSE will ensure unique sequences, i.e., any sequence that was observed multiple times in the FASTQ file will be only included only once in the selection.

Author(s)

Yang Liao, Gordon K Smyth and Wei Shi

Examples

```
## Not run:
# Scan through the fasta file to get transcript names and lengths
transcripts <- scanFasta("GENCODE-Human-transcripts.fa.gz")
nsequences <- nrow(transcripts) - sum(transcripts$Duplicate)

# Assign a random TPM value to each non-duplicated transcript sequence
TPMs <- rep(0, nrow(transcripts))
TPMs[!transcripts$Duplicate] <- rexp(nsequences)

# Generate actual reads.
# The output read file is my-simulated-sample_R1.fastq.gz
# The true read counts are returned.
true.counts <- simReads("GENCODE-Human-transcripts.fa.gz", TPMs, "my-simulated-sample")
print(true.counts[1:10,])

## End(Not run)
```

sublong *Align Long Sequence Reads to a Reference Genome via Seed-and-Vote*

Description

This function aligns DNA-seq reads, generated by long-read sequencing technologies such as Nanopore and PacBio sequencers, to a reference genome.

Usage

```
sublong(

    # basic input/output options
    index,
    readFiles,
    outputFiles,
    outputFormat = "BAM",
    nthreads = 1)
```

Arguments

index	a character vector giving the basename of index files. Index files should be located in the current directory. The provided index should be a full index and also it should have only one block. See buildindex for index building options.
readFiles	a character vector giving the names of input files that contain long sequence reads. FASTQ and gzipped FASTQ formats are both accepted.
outputFiles	a character vector specifying the names of output files that contain read mapping results.
outputFormat	a character string specifying the format of output files. BAM by default. Acceptable formats include SAM and BAM.
nthreads	an integer giving the number of threads used for mapping. 1 by default. Note that when more than one thread is used, the order of reads might be changed in the output.

Details

sublong is designed for the mapping of long reads. It performs full alignment of reads by performing seed-and-vote mapping followed by a bounded dynamic programming procedure. sublong is able to map reads as long as millions of bases.

sublong is extremely fast. It takes less than 10 minutes to complete the mapping of more than 100,000 long reads generated from Nanopore MinION ultra-long sequencing protocol.

The number of CIGAR operations (eg. insertion and deletion) reported for a long read may exceed the limit on the total number of operations allowed in a CIGAR string (up to 65,535 operations in a CIGAR string in BAM output and up to 99,900 operations in a CIGAR string in SAM output). If this limited is exceeded, the read will be soft clipped.

Author(s)

Yang Liao and Wei Shi

Examples

```
ref <- system.file("extdata","reference.fa",package="Rsubread")
buildindex(basename="./full_index",reference=ref,gappedIndex=FALSE, indexSplit=FALSE)
reads <- system.file("extdata","longreads.txt.gz",package="Rsubread")
sublong("./full_index",reads,"./Long_alignment.BAM",nthreads=4)
```

---

txUnique *Count Number of Bases Unique to Each Transcript*

Description

For each transcript, number of unique bases and total number of bases will be reported.

Usage

```
txUnique(

    # basic input/output options
    GTF_Annotation_File,
    Feature_Type = "exon",
    Gene_ID_Attribute = "gene_id",
    Transcript_ID_Attribute = "transcript_id")
```

Arguments

`GTF_Annotation_File`	a character string giving the name of a GTF file containing the transcript-level annotations.
`Feature_Type`	a character string specifying the type of annotations. Only the annotations of the given type will be loaded from the GTF file. `exon` by default. The third column in the GTF file contains the type of the annotation in each line
`Gene_ID_Attribute`	a character string specifying the attribute name in the 9-th column in the GTF file for the gene identifier. `gene_id` by default.
`Transcript_ID_Attribute`	a character string specifying the attribute name in the 9-th column in the GTF file for the transcript identifier. `transcript_id` by default.

Details

This function compares the transcripts belonging to the same gene and then counts the number of bases unique to each transcript. It also reports the total number of bases for each transcript. When a transcript is found to contain overlapping exons, the overlapping exons will be merged and each overlapping base will be counted only once.

Value

A data matrix in which each row represents a transcript. The matrix includes four columns: gene identifier, transcript identifier, number of bases unique to each transcript and total number of bases each transcript has.

Author(s)

Yang Liao and Wei Shi

Index