# Package 'NxtIRFcore'

June 6, 2023

**Title** Core Engine for NxtIRF: a User-Friendly Intron Retention and Alternative Splicing Analysis using the IRFinder Engine

**Version** 1.6.0

**Date** 2022-10-25

**Description** Interactively analyses Intron Retention and Alternative Splicing Events (ASE) in RNA-seq data. NxtIRF quantifies ASE events in BAM files aligned to the genome using a splice-aware aligner such as STAR. The core quantitation algorithm relies on the IRFinder/C++ engine ported via Rcpp for multi-platform compatibility. In addition, NxtIRF provides convenient pipelines for downstream analysis and publication-ready visualisation tools. Note that NxtIRFcore is now replaced by SpliceWiz in Bioconductor 3.16 onwards.

**License** MIT + file LICENSE

**Depends** NxtIRFdata

**Imports** methods, stats, utils, tools, parallel, magrittr, Rcpp (>= 1.0.5), data.table, fst, ggplot2, AnnotationHub, BiocFileCache, BiocGenerics, BiocParallel, Biostrings, BSgenome, DelayedArray, DelayedMatrixStats, genefilter, GenomeInfoDb, GenomicRanges, HDF5Array, IRanges, plotly, R.utils, rhdf5, rtracklayer, SummarizedExperiment, S4Vectors

**LinkingTo** Rcpp, zlibbioc, RcppProgress

**Suggests** knitr, rmarkdown, pheatmap, shiny, openssl, crayon, egg, DESeq2, limma, DoubleExpSeq, Rsubread, testthat (>= 3.0.0)

**VignetteBuilder** knitr

**biocViews** Software, Transcriptomics, RNASeq, AlternativeSplicing, Coverage, DifferentialSplicing

**SystemRequirements** C++11

**Collate** AllImports.R RcppExports.R AllClasses.R AllGenerics.R NxtFilter-methods.R NxtSE-methods.R globals.R ggplot_themes.R example_data.R wrappers.R make_plot_data.R Coverage.R utils.R File_finders.R BuildRef.R STAR_utils.R Mappability.R IRFinder.R CollateData.R MakeSE.R Filters.R ASE-methods.R NxtIRFcore-package.R zzz.R

**Encoding** UTF-8

**RoxygenNote** 7.1.2

**URL** https://github.com/alexchwong/NxtIRFcore

**BugReports** https://support.bioconductor.org/

**Config/testthat/edition** 3

**PackageStatus** Deprecated

**git_url** https://git.bioconductor.org/packages/NxtIRFcore

**git_branch** RELEASE_3_17

**git_last_commit** 2829efa

**git_last_commit_date** 2023-04-25

**Date/Publication** 2023-06-06

**Author** Alex Chit Hei Wong [aut, cre, cph],
William Ritchie [cph],
Ulf Schmitz [ctb]

**Maintainer** Alex Chit Hei Wong <a.wong@centenary.org.au>

# R topics documented:

NxtIRFcore-package *NxtIRFcore: a command line interface for NxtIRF - IRFinder-based differential Alternative Splicing and Intron Retention analysis*

## Description

(Important!) NxtIRFcore will be replaced by SpliceWiz from Bioconductor version 3.16 onwards. SpliceWiz replaces the full functionality of NxtIRFcore, plus heaps more! NxtIRF is a computationally efficient and user friendly workflow that analyses aligned short-read RNA sequencing for differential intron retention and alternative splicing. It utilises an improved IRFinder-based OpenMP/C++ algorithm. A streamlined downstream analysis pipeline allows for GLM-based differential IR and splicing analysis, suited for large datasets of up to hundreds of samples. Additionally NxtIRF provides a novel visualisation of per-nucleotide mean and variations of alignment coverage across splice and IR events, grouped by user-defined experimental conditions.

## Details

IRFinder is a well-established bioinformatic tool that measures intron retention (IR) in annotated and novel retained introns in short-read RNA sequencing samples. It is a computationally-efficient algorithm that measures alignment coverage across introns, accounting for regions of low-mappable intronic regions. Unlike other algorithms that measure exon-intron spanning reads, IRFinder considers the alignment coverage across the whole intron, allowing it to distinguish between full-length and partial IR. This distinction is important as partial IR is often confounded with novel alternate splice site usage, alternate transcription start site and intronic polyadenylation events.

NxtIRF is a R/Bioconductor package that provides a user-friendly workflow using the IRFinder algorithm to perform both IR and alternative splicing analysis in large datasets. By incorporating the core C++ based IRFinder algorithm using Rcpp, NxtIRF is multi-platform and further improves computational efficiency using OpenMP-based multi-threading. Besides analysing IR, NxtIRF analyses other forms of alternative splicing events that depend on alternate splice site selection, including skipped exons, mutually exclusive exons, alternate 5'- and 3'- splice sites, alternate first exons and alternate last exons.

Downstream, NxtIRF provides functions to collate individual NxtIRF/IRFinder outputs of multiple samples in an experiment / dataset, and assembles these into a specialised NxtSE object that inherits the SummarizedExperiment class. Users can easily define experimental conditions, perform differential analysis and filter out lowly-expressed splice events.

Finally, NxtIRF provides visualisation tools to illustrate alternative splicing using coverage plots, including a novel method to normalise RNA-seq coverage grouped by experimental condition. This approach accounts for variations introduced by sequenced library size and gene expression. NxtIRF efficiently computes and visualises means and variations in per-nucleotide coverage depth across alternate exons in genomic loci.

NxtIRFcore is the command line interface for R/Bioconductor. NxtIRF (coming soon) will feature an interactive graphical user interface with additional functions.

**Features include:**

- Reference generation from user-supplied local and web resources, as well as connectivity to the AnnotationHub repository for Ensembl-based genomes and gene annotations;

- OpenMP and BiocParallel-based multi-threaded support to process short-read BAM files using the IRFinder algorithm written in native C++;
- Stores alignment coverage using the *COV* format, which is a binary compressed and indexed format for rapid recall of RNA-seq coverage. In contrast to the *BigWig* format, *COV* files store coverage of unstranded as well as stranded alignment coverage, and is much more space-efficient, allowing for better portability;
- Memory-efficient collation of hundreds of samples using on-disk memory approaches and H5-based assay storage;
- Streamlined user-friendly functions to construct multi-factor complex experimental designs, and perform differential IR and alternative splicing analysis using well-established statistical methods including limma and DESeq2;
- Advanced RNA-seq coverage visualisation, including the ability to combine RNA-seq coverage of multiple samples using advanced library normalisation methods across samples grouped by conditions;

The main functions are:

- BuildReference - Prepares genome and gene annotation references from FASTA and GTF files, and synthesises the NxtIRF reference for the IRFinder engine and NxtIRF-based downstream analysis.
- STAR-methods - (Optional) Provides wrapper functions to build the STAR genome reference and alignment of short-read FASTQ raw sequencing files. This functionality is only available on systems with STAR installed.
- IRFinder - OpenMP/C++ based IRFinder algorithm to analyse single or multiple BAM files using the NxtIRF/IRFinder reference.
- CollateData - Collates an experiment based on multiple IRFinder outputs for individual samples, into one unified H5-based data structure.
- MakeSE - Constructs a NxtSE (H5-based SummarizedExperiment) object, specialised to house measurements of retained introns and junction counts of alternative splice events.
- apply_filters - Use default or custom filters to remove alternative splicing or IR events pertaining to low-abundance genes and transcripts.
- ASE-methods - one-step method to perform differential alternate splice event (ASE) analysis on a NxtSE object using limma or DESeq2.
- make_plot_data: Functions that compile individual and group-mean percent spliced in (PSI) values of IR and alternative splice events; useful to produce scatter plots or heatmaps.
- Plot_Coverage: Generate RNA-seq coverage plots of individual samples or across samples grouped by user-specified conditions

See the NxtIRF vignette for worked examples on how to use NxtIRF

## Author(s)

Alex Wong

## References

Middleton R, Gao D, Thomas A, Singh B, Au A, Wong JJ, Bomane A, Cosson B, Eyras E, Rasko JE, Ritchie W. IRFinder: assessing the impact of intron retention on mammalian gene expression. Genome Biol. 2017 Mar 15;18(1):51. https://doi.org/10.1186/s13059-017-1184-4

---

`ASE-methods` *Use Limma, DESeq2 or DoubleExpSeq to test for differential Alternative Splice Events*

---

## Description

Use Limma, DESeq2 or DoubleExpSeq to test for differential Alternative Splice Events

## Usage

```
limma_ASE(
  se,
  test_factor,
  test_nom,
  test_denom,
  batch1 = "",
  batch2 = "",
  filter_antiover = TRUE,
  filter_antinear = FALSE
)

DESeq_ASE(
  se,
  test_factor,
  test_nom,
  test_denom,
  batch1 = "",
  batch2 = "",
  n_threads = 1,
  filter_antiover = TRUE,
  filter_antinear = FALSE
)

DoubleExpSeq_ASE(
  se,
  test_factor,
  test_nom,
  test_denom,
  filter_antiover = TRUE,
  filter_antinear = FALSE
)
```

## Arguments

| | |
|---|---|
| `se` | The NxtSE object created by `MakeSE()`. To reduce runtime and avoid excessive multiple testing, consider filtering the object using apply_filters |
| `test_factor` | The condition type which contains the contrasting variable |
| `test_nom` | The nominator condition to test for differential ASE. Usually the "treatment" condition |
| `test_denom` | The denominator condition to test against for differential ASE. Usually the "control" condition |
| `batch1, batch2` | (Optional, limma and DESeq2 only) One or two condition types containing batch information to account for. |
| `filter_antiover, filter_antinear` | Whether to remove novel IR events that overlap over or near anti-sense genes. Default will exclude antiover but not antinear introns. These are ignored if stranded RNA-seq protocols are used. |
| `n_threads` | (DESeq2 only) How many threads to use for DESeq2 based analysis. |

## Details

Using **limma**, NxtIRF models included and excluded counts as log-normal distributed, whereas using **DESeq2**, NxtIRF models included and excluded counts as negative binomial distributed with dispersion shrinkage according to their mean count expressions. For **limma** and **DESeq2**, differential ASE are considered as the "interaction" between included and excluded splice counts for each sample. See this vignette for an explanation of how this is done.

Using **DoubleExpSeq**, included and excluded counts are modelled using the generalized beta prime distribution, using empirical Bayes shrinkage to estimate dispersion.

**EventType** are as follow:

- `IR` = (novel) intron retention
- `MXE` = mutually exclusive exons
- `SE` = skipped exons
- `AFE` = alternate first exon
- `ALE` = alternate last exon
- `A5SS` = alternate 5'-splice site
- `A3SS` = alternate 3'-splice site
- `RI` = (known / annotated) intron retention.

NB: NxtIRF separately considers known "RI" and novel "IR" events separately:

- **IR** novel events are calculated using the IRFinder method, whereby spliced transcripts are **all** isoforms that do not retain the intron, as estimated via the `SpliceMax` and `SpliceOverMax` methods
- see CollateData.

- **RI** known retained introns are those that lie completely within a single exon of another transcript. (NB: in NxtIRFcore v1.1.1 and later, this encompasses exons from any transcript, including `retained_intron` and `sense_intronic` transcripts). RI's are calculated by considering the specific spliced intron as a binary event paired with its retention. The spliced abundance is calculated exclusively by splice reads mapped to the specific intron boundaries. Known retained introns are those where the intron retaining transcript is an **annotated** transcript. In NxtIRFcore `version < 1.1.1`, the IR-transcript's `transcript_biotype` must not be an `retained_intron` or `sense_intronic`.

NxtIRF considers "included" counts as those that represent abundance of the "included" isoform, whereas "excluded" counts represent the abundance of the "excluded" isoform. For consistency, it applies a convention whereby the "included" transcript is one where its splice junctions are by definition shorter than those of "excluded" transcripts. Specifically, this means the included / excluded isoforms are as follows:

| EventType | Included | Excluded |
|---|---|---|
| IR or RI | Intron Retention | Spliced Intron |
| MXE | Upstream exon inclusion | Downstream exon inclusion |
| SE | Exon inclusion | Exon skipping |
| AFE | Downstream exon usage | Upstream exon usage |
| ALE | Upstream exon usage | Downstream exon usage |
| A5SS | Downstream 5'-SS | Upstream 5'-SS |
| A3SS | Upstream 3'-SS | Downstream 3'-SS |

## Value

A data table containing the following:

- EventName: The name of the ASE event. This identifies each ASE in downstream functions including make_diagonal, make_matrix, and Plot_Coverage
- EventType: The type of event. See details section above.
- EventRegion: The genomic coordinates the event occupies. This spans the most upstream and most downstream splice junction involved in the ASE, and is use to guide the Plot_Coverage function.
- NMD_direction: Indicates whether one isoform is a NMD substrate. +1 means included isoform is NMD, -1 means the excluded isoform is NMD, and 0 means there is no change in NMD status (i.e. both / neither are NMD)
- AvgPSI_nom, Avg_PSI_denom: the average percent spliced in / percent IR levels for the two conditions being contrasted. `nom` and `denom` in column names are replaced with the condition names

### limma specific output

- logFC, AveExpr, t, P.Value, adj.P.Val, B: limma topTable columns of differential ASE. See limma::topTable for details.
- inc/exc_(logFC, AveExpr, t, P.Value, adj.P.Val, B): limma results for differential testing for raw included / excluded counts only

### DESeq2 specific output

- baseMean, log2FoldChange, lfcSE, stat, pvalue, padj: DESeq2 results columns for differential ASE; see DESeq2::results for details.
- inc/exc_(baseMean, log2FoldChange, lfcSE, stat, pvalue, padj): DESeq2 results for differential testing for raw included / excluded counts only

### DoubleExp specific output

- MLE_nom, MLE_denom: Expectation values for the two groups. `nom` and `denom` in column names are replaced with the condition names
- MLE_LFC: Log2-fold change of the MLE
- P.Value, adj.P.Val: Nominal and BH-adjusted P values
- n_eff: Number of effective samples (i.e. non-zero or non-unity PSI)
- mDepth: Mean Depth of splice coverage in each of the two groups.
- Dispersion_Reduced, Dispersion_Full: Dispersion values for reduced and full models. See DoubleExpSeq::DBGLM1 for details.

## Functions

- `limma_ASE`: Use limma to perform differential ASE analysis of a filtered NxtSE object
- `DESeq_ASE`: Use DESeq2 to perform differential ASE analysis of a filtered NxtSE object
- `DoubleExpSeq_ASE`: Use DoubleExpSeq to perform differential ASE analysis of a filtered NxtSE object (uses double exponential beta-binomial model) to estimate group dispersions, followed by LRT

## References

Ritchie ME, Phipson B, Wu D, Hu Y, Law CW, Shi W, Smyth GK (2015). 'limma powers differential expression analyses for RNA-sequencing and microarray studies.' Nucleic Acids Research, 43(7), e47. https://doi.org/10.1093/nar/gkv007

Love MI, Huber W, Anders S (2014). 'Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2.' Genome Biology, 15, 550. https://doi.org/10.1186/s13059-014-0550-8

Ruddy S, Johnson M, Purdom E (2016). 'Shrinkage of dispersion parameters in the binomial family, with application to differential exon skipping.' Ann. Appl. Stat. 10(2): 690-725. https://doi.org/10.1214/15-AOAS871

## Examples

```
# see ?MakeSE on example code of generating this NxtSE object
se <- NxtIRF_example_NxtSE()

colData(se)$treatment <- rep(c("A", "B"), each = 3)

require("limma")
res_limma <- limma_ASE(se, "treatment", "A", "B")

require("DoubleExpSeq")
```

```
res_DES <- DoubleExpSeq_ASE(se, "treatment", "A", "B")
## Not run:

require("DESeq2")
res_DESeq <- DESeq_ASE(se, "treatment", "A", "B")

## End(Not run)
```

---

BuildReference *Builds reference files used by IRFinder / NxtIRF.*

---

## Description

These function builds the reference required by the IRFinder engine, as well as alternative splicing annotation data for NxtIRF. See examples below for guides to making the NxtIRF reference.

## Usage

```
GetReferenceResource(
  reference_path = "./Reference",
  fasta = "",
  gtf = "",
  overwrite = FALSE,
  force_download = FALSE
)

BuildReference(
  reference_path = "./Reference",
  fasta = "",
  gtf = "",
  overwrite = FALSE,
  force_download = FALSE,
  chromosome_aliases = NULL,
  genome_type = "",
  nonPolyARef = "",
  MappabilityRef = "",
  BlacklistRef = "",
  UseExtendedTranscripts = TRUE
)

GetNonPolyARef(genome_type)

BuildReference_Full(
  reference_path,
  fasta,
  gtf,
  chromosome_aliases = NULL,
```

```
  overwrite = FALSE,
  force_download = FALSE,
  genome_type = genome_type,
  use_STAR_mappability = FALSE,
  nonPolyARef = GetNonPolyARef(genome_type),
  BlacklistRef = "",
  UseExtendedTranscripts = TRUE,
  n_threads = 4
)
```

## Arguments

| | |
|---|---|
| reference_path | (REQUIRED) The directory path to store the generated reference files |
| fasta | The file path or web link to the user-supplied genome FASTA file. Alternatively, the name of the AnnotationHub record containing the genome resource. May be omitted if GetReferenceResource() has already been run using the same reference_path. |
| gtf | The file path or web link to the user-supplied transcript GTF file (or gzipped GTF file). Alternatively, the name of the AnnotationHub record containing the transcript GTF file. May be omitted if GetReferenceResource() has already been run using the same reference_path. |
| overwrite | (default FALSE) For GetReferenceResource(): if the genome FASTA and gene annotation GTF files already exist in the resource subdirectory, it will not be overwritten. For BuildReference() and BuildReference_Full(): the NxtIRF reference will not be overwritten if one already exist. A reference is considered to exist if the file IRFinder.ref.gz is present inside reference_path. |
| force_download | (default FALSE) When online resources are retrieved, a local copy is stored in the NxtIRFcore BiocFileCache. Subsequent calls to the web resource will fetch the local copy. Set force_download to TRUE will force the resource to be downloaded from the web. Set this to TRUE only if the web resource has been updated since the last retrieval. |
| chromosome_aliases | (Highly optional) A 2-column data frame containing chromosome name conversions. If this is set, allows IRFinder to parse BAM files where alignments are made to a genome whose chromosomes are named differently to the reference genome. The most common scenario is where Ensembl genome typically use chromosomes "1", "2", ..., "X", "Y", whereas UCSC/Gencode genome use "chr1", "chr2", ..., "chrX", "chrY". See example below. Refer to https://github.com/dpryan79/ChromosomeMappings for a list of chromosome alias resources. |
| genome_type | Allows BuildReference() to select default nonPolyARef and MappabilityRef for selected genomes. Allowed options are: hg38, hg19, mm10, and mm9. |
| nonPolyARef | (Optional) A BED file of regions defining known non-polyadenylated transcripts. This file is used for QC analysis of IRFinder-processed files to measure Poly-A enrichment quality of samples. If omitted, and genome_type is defined, the default for the specified genome will be used. |

`MappabilityRef` (Optional) A BED file of low mappability regions due to repeat elements in the genome. If omitted, the file generated by `Mappability_CalculateExclusions()` will be used where available, and if this is not, the default file for the specified `genome_type` will be used. If `genome_type` is not specified, `MappabilityRef` is not used. See details.

`BlacklistRef` A BED file of regions to be otherwise excluded from IR analysis. If omitted, a blacklist is not used (this is the default).

`UseExtendedTranscripts`
(default `TRUE`) Should non-protein-coding transcripts such as anti-sense and lincRNA transcripts be included in searching for IR / AS events? Setting `FALSE` (vanilla IRFinder) will exclude transcripts other than `protein_coding` and `processed_transcript` transcripts from IR analysis.

`use_STAR_mappability`
(default FALSE) In `BuildReference_Full()`, whether to run STAR_Mappability to calculate low-mappability regions. We recommend setting this to `FALSE` for the common genomes (human and mouse), and to `TRUE` for genomes not supported by `genome_type`. When set to false, the MappabilityExclusion default file corresponding to `genome_type` will automatically be used.

`n_threads` The number of threads used to generate the STAR reference and mappability calculations. Multi-threading is not used for NxtIRF reference generation (but multiple cores are utilised in data-table and fst file processing automatically, where available). See STAR-methods

## Details

`GetReferenceResource()` processes the files, downloads resources from web links or from `AnnotationHub()`, and saves a local copy in the "resource" subdirectory within the given `reference_path`. Resources are retrieved via either:

1. User-supplied FASTA and GTF file. This can be a file path, or a web link (e.g. 'http://', 'https://' or 'ftp://'). Use `fasta` and `gtf` to specify the files or web paths to use.
2. AnnotationHub genome and gene annotation (Ensembl): supply the names of the genome sequence and gene annotations to `fasta` and `gtf`.

`BuildReference()` will first run `GetReferenceResource()` if resources are not yet saved locally (i.e. `GetReferenceResource()` is not already run). Then, it creates the NxtIRF / IRFinder references. Typical run-times are 5 to 10 minutes for human and mouse genomes (after resources are downloaded).

NB: the parameters `fasta` and `gtf` can be omitted in `BuildReference()` if `GetReferenceResource()` is already run.

Typical usage involves running `BuildReference()` for human and mouse genomes and specifying the `genome_type` to use the default `MappabilityRef` and `nonPolyARef` files for the specified genome. For non-human non-mouse genomes, use one of the following alternatives:

- Create the NxtIRF reference without using Mappability Exclusion regions. To do this, simply run `BuildReference()` and omit `MappabilityRef`. This is acceptable assuming the introns assessed are short and do not contain intronic repeats

- Calculating Mappability Exclusion regions using the STAR aligner, and building the NxtIRF reference. This can be done using the `BuildReference_Full()` function, on systems where `STAR` is installed
- Instead of using the STAR aligner, any genome splice-aware aligner could be used. See Mappability-methods for details. After producing the `MappabilityExclusion.bed.gz` file (in the `Mappability` subfolder), run `BuildReference()` using this file (or simply leave it blank).

BED files are tab-separated text files containing 3 unnamed columns specifying chromosome, start and end coordinates. To view an example BED file, open the file specified in the path returned by `GetNonPolyARef("hg38")`

See examples below for common use cases.

## Value

For `GetReferenceResource`: creates the following local resources:

- `reference_path/resource/genome.2bit`: Local copy of the genome sequences as a TwoBitFile.
- `reference_path/resource/transcripts.gtf.gz`: Local copy of the gene annotation as a gzip-compressed file. For `BuildReference` and `BuildReference_Full`: creates a NxtIRF reference which is written to the given directory specified by `reference_path`. Files created includes:
- `reference_path/settings.Rds`: An RDS file containing parameters used to generate the NxtIRF reference
- `reference_path/IRFinder.ref.gz`: A gzipped text file containing collated IRFinder reference files. This file is used by IRFinder
- `reference_path/fst/`: Contains fst files for subsequent easy access to NxtIRF generated references
- `reference_path/cov_data.Rds`: An RDS file containing data required to visualise genome / transcript tracks.

`BuildReference_Full` also creates a `STAR` reference located in the `STAR` subdirectory inside the designated `reference_path`

For `GetNonPolyARef`: Returns the file path to the BED file for the nonPolyA loci for the specified genome.

## Functions

- `GetReferenceResource`: Processes / downloads a copy of the genome and gene annotations and stores this in the "resource" subdirectory of the given reference path
- `BuildReference`: First calls `GetReferenceResource()` (if required). Afterwards creates the NxtIRF reference in the given reference path
- `GetNonPolyARef`: Returns the path to the BED file containing coordinates of known non-polyadenylated transcripts for genomes `hg38`, `hg19`, `mm10` and `mm9`,
- `BuildReference_Full`: One-step function that fetches resources, creates a STAR reference (including mappability calculations), then creates the NxtIRF reference

## See Also

Mappability-methods for methods to calculate low mappability regions

STAR-methods for a list of STAR wrapper functions

AnnotationHub

## Examples

```
# Quick runnable example: generate a reference using NxtIRF's example genome

example_ref <- file.path(tempdir(), "Reference")
GetReferenceResource(
    reference_path = example_ref,
    fasta = chrZ_genome(),
    gtf = chrZ_gtf()
)
BuildReference(
    reference_path = example_ref
)

# NB: the above is equivalent to:

example_ref <- file.path(tempdir(), "Reference")
BuildReference(
    reference_path = example_ref,
    fasta = chrZ_genome(),
    gtf = chrZ_gtf()
)

# Get the path to the Non-PolyA BED file for hg19

GetNonPolyARef("hg19")
## Not run:

### Long examples ###

# Generate a NxtIRF reference from user supplied FASTA and GTF files for a
# hg38-based genome:

BuildReference(
    reference_path = "./Reference_user",
    fasta = "genome.fa", gtf = "transcripts.gtf",
    genome_type = "hg38"
)

# NB: Setting `genome_type = hg38`, will automatically use default
# nonPolyARef and MappabilityRef for `hg38`
```

```
# Reference generation from Ensembl's FTP links:

FTP <- "ftp://ftp.ensembl.org/pub/release-94/"
BuildReference(
    reference_path = "./Reference_FTP",
    fasta = paste0(FTP, "fasta/homo_sapiens/dna/",
        "Homo_sapiens.GRCh38.dna.primary_assembly.fa.gz"),
    gtf = paste0(FTP, "gtf/homo_sapiens/",
        "Homo_sapiens.GRCh38.94.chr.gtf.gz"),
    genome_type = "hg38"
)

# Get AnnotationHub record names for Ensembl release-94:

# First, search for the relevant AnnotationHub record names:

ah <- AnnotationHub::AnnotationHub()
AnnotationHub::query(ah, c("Homo Sapiens", "release-94"))
# snapshotDate(): 2021-09-23
# $dataprovider: Ensembl
# $species: Homo sapiens
# $rdataclass: TwoBitFile, GRanges
# additional mcols(): taxonomyid, genome, description, coordinate_1_based,
#   maintainer, rdatadateadded, preparerclass, tags,
#   rdatapath, sourceurl, sourcetype
# retrieve records with, e.g., 'object[["AH64628"]]'
#
#            title
#   AH64628 | Homo_sapiens.GRCh38.94.abinitio.gtf
#   AH64629 | Homo_sapiens.GRCh38.94.chr.gtf
#   AH64630 | Homo_sapiens.GRCh38.94.chr_patch_hapl_scaff.gtf
#   AH64631 | Homo_sapiens.GRCh38.94.gtf
#   AH65744 | Homo_sapiens.GRCh38.cdna.all.2bit
#   AH65745 | Homo_sapiens.GRCh38.dna.primary_assembly.2bit
#   AH65746 | Homo_sapiens.GRCh38.dna_rm.primary_assembly.2bit
#   AH65747 | Homo_sapiens.GRCh38.dna_sm.primary_assembly.2bit
#   AH65748 | Homo_sapiens.GRCh38.ncrna.2bit

BuildReference(
    reference_path = "./Reference_AH",
    fasta = "AH65745",
    gtf = "AH64631",
    genome_type = "hg38"
)

# Build a NxtIRF reference, setting chromosome aliases to allow
# this reference to process BAM files aligned to UCSC-style genomes:

chrom.df <- GenomeInfoDb::genomeStyles()$Homo_sapiens

BuildReference(
    reference_path = "./Reference_UCSC",
    fasta = "AH65745",
```

```
    gtf = "AH64631",
    genome_type = "hg38",
    chromosome_aliases = chrom.df[, c("Ensembl", "UCSC")]
)

# One-step generation of NxtIRF and STAR references, using 4 threads.
# NB1: requires a linux-based system with STAR installed.
# NB2: A STAR reference genome will be generated in the `STAR` subfolder
#      inside the given `reference_path`.
# NB3: A custom Mappability Exclusion file will be calculated using STAR
#      and will be used to generate the NxtIRF reference.

BuildReference_Full(
    reference_path = "./Reference_with_STAR",
    fasta = "genome.fa", gtf = "transcripts.gtf",
    genome_type = "",
    use_STAR_mappability = TRUE,
    n_threads = 4
)

# NB: the above is equivalent to running the following in sequence:

GetReferenceResource(
    reference_path = "./Reference_with_STAR",
    fasta = "genome.fa", gtf = "transcripts.gtf"
)
STAR_buildRef(
    reference_path = reference_path,
    also_generate_mappability = TRUE,
    n_threads = 4
)
BuildReference(
    reference_path = "./Reference_with_STAR",
    genome_type = ""
)

## End(Not run)
```

---

`CollateData` *Processes data from IRFinder output*

---

## Description

CollateData unifies a list of IRFinder output files belonging to an experiment.

## Usage

```
CollateData(
  Experiment,
  reference_path,
```

```
  output_path,
  IRMode = c("SpliceOverMax", "SpliceMax"),
  overwrite = FALSE,
  n_threads = 1,
  samples_per_block = 16
)
```

## Arguments

| | |
|---|---|
| `Experiment` | (Required) A 2 or 3 column data frame, ideally generated by Find_IRFinder_Output or Find_Samples. The first column designate the sample names, and the 2nd column contains the path to the IRFinder output file (of type `sample.txt.gz`). (Optionally) a 3rd column contains the coverage files (of type `sample.cov`) of the corresponding samples. NB: all other columns are ignored. |
| `reference_path` | (Required) The path to the reference generated by BuildReference |
| `output_path` | (Required) The path to contain the output files for this function |
| `IRMode` | (default `SpliceOverMax`) The algorithm to calculate 'splice abundance' in IR quantification. Valid options are `SpliceOverMax` and `SpliceMax`. See details |
| `overwrite` | (default `FALSE`) If `CollateData()` has previously been run using the same set of samples, it will not be overwritten unless this is set to `TRUE`. |
| `n_threads` | (default 1) The number of threads to use. On low memory systems, reduce the number of `n_threads` and `samples_per_block` |
| `samples_per_block` | (default 16) How many samples to process per thread, maximum. Setting this to a lower value may help in memory-constrained systems. |

## Details

All sample IRFinder outputs must be generated using the same reference.

The combination of junction counts and IR quantification from IRFinder is used to calculate percentage spliced in (PSI) of alternative splice events, and percent intron retention (PIR) of retained introns. Also, QC information is extracted. Data is organised in a H5file and FST files for memory and processor efficient downstream access using MakeSE.

The original IRFinder algorithm, see the following wiki, uses `SpliceMax` to estimate abundance of spliced transcripts. This calculates the number of mapped splice events that share the boundary coordinate of either the left or right flanking exon `SpliceLeft`,`SpliceRight`, estimating splice abundance as the larger of the two values.

NxtIRF proposes a new algorithm,`SpliceOverMax`, to account for the possibility that the major isoform shares neither boundary, but arises from either of the flanking "exon islands". Exon islands are contiguous regions covered by exons from any transcript (except those designated as `retained_intron` or `sense_intronic`), and are separated by obligate intronic regions (genomic regions that are introns for all transcripts). For introns that are internal to a single exon island (i.e. akin to "known-exon" introns from IRFinder), `SpliceOverMax` uses GenomicRanges::findOverlaps to sum all splice reads that overlap the same genomic region as the intron of interest.

## Value

`CollateData()` writes to the directory given by `output_path`. This output directory is portable (i.e. it can be moved to a different location after running `CollateData()` before running MakeSE), but individual files within the output folder should not be moved.

Also, the IRFinder and CollateData output folders should be copied to the same destination and their relative paths preserved. Otherwise, the locations of the "COV" files will not be recorded in the collated data and will have to be re-assigned using `covfile(se)<-`. See MakeSE

## See Also

IRFinder, MakeSE

## Examples

```
BuildReference(
    reference_path = file.path(tempdir(), "Reference"),
    fasta = chrZ_genome(),
    gtf = chrZ_gtf()
)

bams <- NxtIRF_example_bams()
IRFinder(bams$path, bams$sample,
  reference_path = file.path(tempdir(), "Reference"),
  output_path = file.path(tempdir(), "IRFinder_output")
)

expr <- Find_IRFinder_Output(file.path(tempdir(), "IRFinder_output"))
CollateData(expr,
  reference_path = file.path(tempdir(), "Reference"),
  output_path = file.path(tempdir(), "NxtIRF_output")
)
```

---

# CoordToGR *Converts genomic coordinates into a GRanges object*

---

## Description

This function takes a string vector of genomic coordinates and converts it into a GRanges object.

## Usage

```
CoordToGR(coordinates)
```

## Arguments

| | |
|---|---|
| `coordinates` | A string vector of one or more genomic coordinates to be converted |

## Details

Genomic coordinates can take one of the following syntax:

- `seqnames:start`
- `seqnames:start-end`
- `seqnames:start-end/strand`

The following examples are considered valid genomic coordinates:

- "chr1:21535"
- "chr3:10550-10730"
- "X:51231-51330/-"
- "chrM:2134-5232/+"

## Value

A GRanges object that corresponds to the given coordinates

## Examples

```
se <- NxtIRF_example_NxtSE()

coordinates <- rowData(se)$EventRegion

gr <- CoordToGR(coordinates)
```

---

# Coverage *Calls NxtIRF's C++ function to retrieve coverage from a COV file*

---

## Description

This function returns an RLE / RLEList or data.frame containing coverage data from the given COV file

COV files are generated by NxtIRF's IRFinder and BAM2COV functions. It records alignment coverage for each nucleotide in the given BAM file. It stores this data in "COV" format, which is an indexed BGZF-compressed format specialised for the storage of unstranded and stranded alignment coverage in RNA sequencing.

Unlike BigWig files, COV files store coverage for both positive and negative strands.

These functions retrieves coverage data from the specified COV file. They are computationally efficient as they utilise random-access to rapidly search for the requested data from the COV file.

## Usage

```
GetCoverage(file, seqname = "", start = 0, end = 0, strand = c("*", "+", "-"))

GetCoverage_DF(
  file,
  seqname = "",
  start = 0,
  end = 0,
  strand = c("*", "+", "-")
)

GetCoverageRegions(
  file,
  regions,
  strandMode = c("unstranded", "forward", "reverse")
)

GetCoverageBins(
  file,
  region,
  bins = 2000,
  strandMode = c("unstranded", "forward", "reverse"),
  bin_size
)
```

## Arguments

| | |
|---|---|
| `file` | (Required) The file name of the COV file |
| `seqname` | (Required for `GetCoverage_DF`) A string denoting the chromosome name. If left blank in `GetCoverage`, retrieves RLEList containing coverage of the entire file. |
| `start, end` | 1-based genomic coordinates. If `start = 0` and `end = 0`, will retrieve RLE of specified chromosome. |
| `strand` | Either "*", "+", or "-" |
| `regions` | A GRanges object for a set of regions to obtain mean / total coverage from the given COV file. |
| `strandMode` | The stranded-ness of the RNA-seq experiment. "unstranded" means that an unstranded protocol was used. Stranded protocols can be either "forward", where the first read is the same strand as the expressed transcript, or "reverse" where the second strand is the same strand as the expressed transcript. |
| `region` | In `GetCoverageBins`, a single query region as a GRanges object |
| `bins` | In `GetCoverageBins`, the number of bins to divide the given `region`. If `bin_size` is given, overrides this parameter |
| `bin_size` | In `GetCoverageBins`, the number of nucleotides per bin |

## Value

For GetCoverage: If seqname is left as "", returns an RLEList of the whole BAM file, with each RLE in the list containing coverage data for one chromosome. Otherwise, returns an RLE containing coverage data for the requested genomic region

For GetCoverage_DF: Returns a two-column data frame, with the first column coordinate denoting genomic coordinate, and the second column value containing the coverage depth for each coordinate nucleotide.

For GetCoverageRegions: Returns a GRanges object with an extra metacolumn: cov_mean, which gives the mean coverage of each of the given ranges.

For GetCoverageBins: Returns a GRanges object which spans the given region, divided by the number of bins or by width as given by bin_size. Mean coverage in each bin is calculated (returned by the cov_mean metadata column). This function is useful for retrieving coverage of a large region for visualisation, especially when the size of the region vastly exceeds the width of the figure.

## Functions

- GetCoverage: Retrieves alignment coverage as an RLE or RLElist
- GetCoverage_DF: Retrieves alignment coverage as a data.frame
- GetCoverageRegions: Retrieves total and mean coverage of a GRanges object from a COV file
- GetCoverageBins: Retrieves coverage of a single region from a COV file, binned by the given number of bins or bin_size

## Examples

```
se <- NxtIRF_example_NxtSE()

cov_file <- covfile(se)[1]

# Retrieve Coverage as RLE

cov <- GetCoverage(cov_file, seqname = "chrZ",
  start = 10000, end = 20000,
  strand = "*"
)

# Retrieve Coverage as data.frame

cov.df <- GetCoverage_DF(cov_file, seqname = "chrZ",
  start = 10000, end = 20000,
  strand = "*"
)

# Retrieve mean coverage of 100-nt window regions as defined
# in a GRanges object:

gr <- GenomicRanges::GRanges(
    seqnames = "chrZ",
```

```
        ranges = IRanges::IRanges(
            start = seq(1, 99901, by = 100),
            end = seq(100, 100000, by = 100)
        ), strand = "-"
    )

    gr.unstranded <- GetCoverageRegions(cov_file,
        regions = gr,
        strandMode = "unstranded"
    )

    gr.stranded <- GetCoverageRegions(cov_file,
        regions = gr,
        strandMode = "reverse"
    )

    # Retrieve binned coverage of a large region

    gr.fetch = GetCoverageBins(
        cov_file,
        region = GenomicRanges::GRanges(seqnames = "chrZ",
            ranges = IRanges::IRanges(start = 100, end = 100000),
            strand = "*"
        ),
        bins = 2000
    )

    # Plot coverage using ggplot:

    require(ggplot2)

    ggplot(cov.df, aes(x = coordinate, y = value)) +
        geom_line() + theme_white

    ggplot(as.data.frame(gr.unstranded),
        aes(x = (start + end) / 2, y = cov_mean)) +
        geom_line() + theme_white

    ggplot(as.data.frame(gr.fetch),
        aes(x = (start + end)/2, y = cov_mean)) +
        geom_line() + theme_white

    # Export COV data as BigWig

    cov_whole <- GetCoverage(cov_file)
    bw_file <- file.path(tempdir(), "sample.bw")
    rtracklayer::export(cov_whole, bw_file, "bw")
```

---

example-NxtIRF-data *NxtIRF Example BAMs and NxtSE Experiment Object*

---

## Description

`NxtIRF_example_bams()` is a wrapper function to obtain and make a local copy of 6 example files provided by the NxtIRFdata companion package to demonstrate the use of NxtIRFcore. See NxtIRFdata::example_bams for a description of the provided BAM files.

`NxtIRF_example_NxtSE()` retrieves a ready-made functioning NxtSE object. The steps to reproduce this object is shown in the example code in MakeSE

## Usage

```
NxtIRF_example_bams()

NxtIRF_example_NxtSE()
```

## Value

In `NxtIRF_example_bams()`: returns a 2-column data frame containing sample names and BAM paths of the example dataset.

In `NxtIRF_example_NxtSE()`: returns a NxtSE object.

## Functions

- `NxtIRF_example_bams`: Returns a 2-column data frame, containing sample names and sample paths (in tempdir()) of example BAM files
- `NxtIRF_example_NxtSE`: Returns a (in-memory / realized) NxtSE object that was pre-generated using the NxtIRF example reference and example BAM files

## References

Generation of the mappability files was performed using NxtIRF using a method analogous to that described in:

Middleton R, Gao D, Thomas A, Singh B, Au A, Wong JJ, Bomane A, Cosson B, Eyras E, Rasko JE, Ritchie W. IRFinder: assessing the impact of intron retention on mammalian gene expression. Genome Biol. 2017 Mar 15;18(1):51. doi:10.1186/s1305901711844

## See Also

MakeSE

## Examples

```
# returns a data frame with the first column as sample names, and the
# second column as BAM paths

NxtIRF_example_bams()

# Returns a NxtSE object created by the example bams aligned to the
# mock NxtSE reference
```

```
se <- NxtIRF_example_NxtSE()
```

---

`Find_Samples` *Convenience Function to (recursively) find all files in a folder.*

---

## Description

Often, files e.g. raw sequencing FASTQ files, alignment BAM files, or IRFinder output files, are stored in a single folder under some directory structure. They can be grouped by being in common directory or having common names. Often, their sample names can be gleaned by these common names or the names of the folders in which they are contained. This function (recursively) finds all files and extracts sample names assuming either the files are named by sample names (`level = 0`), or that their names can be derived from the parent folder (`level = 1`). Higher `level` also work (e.g. `level = 2`) mean the parent folder of the parent folder of the file is named by sample names. See details section below.

## Usage

```
Find_Samples(sample_path, suffix = ".txt.gz", level = 0)

Find_FASTQ(
  sample_path,
  paired = TRUE,
  fastq_suffix = c(".fastq", ".fq", ".fastq.gz", ".fq.gz"),
  level = 0
)

Find_Bams(sample_path, level = 0)

Find_IRFinder_Output(sample_path, level = 0)
```

## Arguments

| | |
|---|---|
| `sample_path` | The path in which to recursively search for files that match the given `suffix` |
| `suffix` | A vector of or or more strings that specifies the file suffix (e.g. '.bam' denotes BAM files, whereas ".txt.gz" denotes gzipped txt files). |
| `level` | Whether sample names can be found in the file names themselves (level = 0), or their parent directory (level = 1). Potentially parent of parent directory (level = 2). Support max level <= 3 (for sanity). |
| `paired` | Whether to expect single FASTQ files (of the format "sample.fastq"), or paired files (of the format "sample_1.fastq", "sample_2.fastq") |
| `fastq_suffix` | The name of the FASTQ suffix. Options are: ".fastq", ".fastq.gz", ".fq", or ".fq.gz" |

## Details

Paired FASTQ files are assumed to be named using the suffix `_1` and `_2` after their common names; e.g. `sample_1.fastq`, `sample_2.fastq`. Alternate FASTQ suffixes for `Find_FASTQ()` include ".fq", ".fastq.gz", and ".fq.gz".

In BAM files, often the parent directory denotes their sample names. In this case, use `level = 1` to automatically annotate the sample names using `Find_Bams()`.

IRFinder outputs two files per BAM processed. These are named by the given sample names. The text output is named "sample1.txt.gz", and the COV file is named "sample1.cov", where `sample1` is the name of the sample. These files can be organised / tabulated using the function `Find_IRFinder_Output`. The generic function `Find_Samples` will organise the IRFinder text output files but exclude the COV files. Use the latter as the `Experiment` in CollateData if one decides to collate an experiment without linked COV files, for portability reasons.

## Value

A multi-column data frame with the first column containing the sample name, and subsequent columns being the file paths with suffix as determined by `suffix`.

## Functions

- `Find_Samples`: Finds all files with the given suffix pattern. Annotates sample names based on file or parent folder names.
- `Find_FASTQ`: Use Find_Samples() to return all FASTQ files in a given folder
- `Find_Bams`: Use Find_Samples() to return all BAM files in a given folder
- `Find_IRFinder_Output`: Use Find_Samples() to return all IRFinder output files in a given folder, including COV files

## Examples

```
# Retrieve all BAM files in a given folder, named by sample names
bam_path <- tempdir()
example_bams(path = bam_path)
df.bams <- Find_Samples(sample_path = bam_path,
  suffix = ".bam", level = 0)
# equivalent to:
df.bams <- Find_Bams(bam_path, level = 0)

# Retrieve all IRFinder output files in a given folder,
# named by sample names

expr <- Find_IRFinder_Output(file.path(tempdir(), "IRFinder_output"))
## Not run:

# Find FASTQ files in a directory, named by sample names
# where files are in the form:
# - "./sample_folder/sample1.fastq"
# - "./sample_folder/sample2.fastq"

Find_FASTQ("./sample_folder", paired = FALSE, fastq_suffix = ".fastq")
```

```
# Find paired gzipped FASTQ files in a directory, named by parent directory
# where files are in the form:
# - "./sample_folder/sample1/raw_1.fq.gz"
# - "./sample_folder/sample1/raw_2.fq.gz"
# - "./sample_folder/sample2/raw_1.fq.gz"
# - "./sample_folder/sample2/raw_2.fq.gz"

Find_FASTQ("./sample_folder", paired = TRUE, fastq_suffix = ".fq.gz")

## End(Not run)
```

---

IRFinder *Runs the OpenMP/C++-based NxtIRF/IRFinder algorithm*

---

## Description

These function calls the IRFinder C++ routine on one or more BAM files.

The routine is an improved version over the original IRFinder, with OpenMP-based multi-threading and the production of compact "COV" files to record alignment coverage. A NxtIRF reference built using BuildReference is required.

After IRFinder is run, users should call CollateData to collate individual outputs into an experiment / dataset.

BAM2COV creates COV files from BAM files without running the full IRFinder algorithm.

See details for performance info.

## Usage

```
BAM2COV(
  bamfiles = "./Unsorted.bam",
  sample_names = "sample1",
  output_path = "./cov_folder",
  n_threads = 1,
  Use_OpenMP = TRUE,
  overwrite = FALSE,
  verbose = FALSE
)

IRFinder(
  bamfiles = "./Unsorted.bam",
  sample_names = "sample1",
  reference_path = "./Reference",
  output_path = "./IRFinder_Output",
```

```
  n_threads = 1,
  Use_OpenMP = TRUE,
  overwrite = FALSE,
  run_featureCounts = FALSE,
  verbose = FALSE
)
```

## Arguments

| | |
|---|---|
| bamfiles | A vector containing file paths of 1 or more BAM files |
| sample_names | The sample names of the given BAM files. Must be a vector of the same length as bamfiles |
| output_path | The output directory of this function |
| n_threads | (default 1) The number of threads to use. See details. |
| Use_OpenMP | (default TRUE) Whether to use OpenMP to run IRFinder. If set to FALSE, BiocParallel will be used if n_threads is set |
| overwrite | (default FALSE) If IRFinder output files already exist, will not attempt to re-run. If run_featureCounts is TRUE, will not overwrite gene counts of previous run unless overwrite is TRUE. |
| verbose | (default FALSE) Set to TRUE to allow IRFinder to output progress bars and messages |
| reference_path | The directory containing the NxtIRF reference |
| run_featureCounts | (default FALSE) Whether this function will run Rsubread::featureCounts on the BAM files after running IRFinder. If so, the output will be saved to "main.FC.Rds in the output_path directory as a list object. |

## Details

Typical run-times for a 100-million paired-end alignment BAM file takes 10 minutes using a single core. Using 8 threads, the runtime is approximately 2 minutes. Approximately 10 Gb of RAM is used when OpenMP is used. If OpenMP is not used (see below), this memory usage is multiplied across the number of processor threads (i.e. 40 Gb if n_threads = 4).

OpenMP is natively available to Linux / Windows compilers, and OpenMP will be used if Use_OpenMP is set to TRUE, using multiple threads to process each BAM file. On Macs, if OpenMP is not available at compilation, BiocParallel will be used, processing BAM files simultaneously, with one BAM file per thread.

## Value

IRFinder output will be saved to output_path. Output files will be named using the given sample names.

- sample.txt.gz: The main IRFinder output file containing the quantitation of IR and splice junctions, as well as QC information

- sample.cov: Contains coverage information in compressed binary. See GetCoverage
- main.FC.Rds: A single file containing gene counts for the whole dataset (only if `run_featureCounts == TRUE`)

## Functions

- `BAM2COV`: Converts BAM files to COV files without running IRFinder algorithm
- `IRFinder`: Runs IRFinder algorithm on BAM files. Requires a NxtIRF/IRFinder reference generated by BuildReference()

## See Also

BuildReference CollateData IsCOV

## Examples

```
# Run BAM2COV, which only produces COV files but does not run IRFinder:

bams <- NxtIRF_example_bams()

BAM2COV(bams$path, bams$sample,
  output_path = file.path(tempdir(), "IRFinder_output"),
  n_threads = 2, overwrite = TRUE
)

# Run IRFinder algorithm, which produces:
# - text output of intron coverage and spliced read counts
# - COV files which record read coverages

example_ref <- file.path(tempdir(), "Reference")

BuildReference(
    reference_path = example_ref,
    fasta = chrZ_genome(),
    gtf = chrZ_gtf()
)

bams <- NxtIRF_example_bams()

IRFinder(bams$path, bams$sample,
  reference_path = file.path(tempdir(), "Reference"),
  output_path = file.path(tempdir(), "IRFinder_output"),
  n_threads = 2
)
```

IsCOV *Validates the given file as a valid COV file*

## Description

This function takes the path of a possible COV file and checks whether its format complies with that of the COV format defined by this package.

## Usage

```
IsCOV(coverage_files)
```

## Arguments

| | |
|---|---|
| coverage_files | A vector containing the file names of files to be checked |

## Details

COV files are BGZF-compressed files. The first 4 bytes of the file must always be 'COV\1', distinguishing it from BAM or other files in BGZF format. This function checks whether the given file complies with this.

## Value

TRUE if all files are valid COV files. FALSE otherwise

## See Also

IRFinder CollateData

## Examples

```
se <- NxtIRF_example_NxtSE()

cov_files <- covfile(se)

IsCOV(cov_files) # returns true if these are true COV files
```

MakeSE *Constructs a NxtSE object from the collated data*

## Description

Creates a NxtSE object from the data from IRFinder output collated using CollateData. This object is used for downstream differential analysis of IR and alternative splicing events using ASE-methods as well as visualisation using Plot_Coverage

## Usage

```
MakeSE(collate_path, colData, RemoveOverlapping = TRUE, realize = FALSE)
```

## Arguments

| | |
|---|---|
| collate_path | (Required) The output path of CollateData pointing to the collated data |
| colData | (Optional) A data frame containing the sample annotation information. The first column must contain the sample names. Omit colData to generate a NxtSE object of the whole dataset without any assigned annotations. Alternatively, if the names of only a subset of samples are given, then MakeSE() will construct the NxtSE object based only on the samples given. The colData can be set later using colData() |
| RemoveOverlapping | (default = TRUE) Whether to filter out overlapping novel IR events belonging to minor isoforms. See details. |
| realize | (default = FALSE) Whether to load all assay data into memory. See details |

## Details

MakeSE retrieves the generic SummarizedExperiment structure saved by CollateData, and initialises a NxtSE object. It references the required on-disk assay data using DelayedArrays, thereby utilising 'on-disk' memory to conserve memory usage.

For extremely large datasets, loading the entire data into memory may consume too much memory. In such cases, make a subset of the NxtSE object (e.g. subset by samples) before loading the data into memory (RAM) using realize_NxtSE

It should be noted that downstream applications of NxtIRF, including ASE-methods, Plot_Coverage, are much faster if the NxtSE is realized. It is recommended to realize the NxtSE object before extensive usage.

If COV files assigned via CollateData have been moved relative to the collate_path, the created NxtSE object will not have any linked COV files and Plot_Coverage cannot be used. To reassign these files, a vector of file paths corresponding to all the COV files of the data set can be assigned using covfile(se) <- vector_of_cov_files. See example below for details.

If RemoveOverlapping = TRUE, MakeSE will try to identify which introns belong to major isoforms, then remove introns of minor introns that overlaps those of major isoforms. Non-overlapping introns are then reassessed iteratively, until all introns are included or excluded in this way. This is important to ensure that overlapping novel IR events are not 'double-counted'.

## Value

A NxtSE object containing the compiled data in DelayedArrays pointing to the assay data contained in the given collate_path

## Examples

```
# The following code can be used to reproduce the NxtSE object
# that can be fetched with NxtIRF_example_NxtSE()

BuildReference(
    reference_path = file.path(tempdir(), "Reference"),
    fasta = chrZ_genome(),
    gtf = chrZ_gtf()
)

bams <- NxtIRF_example_bams()
IRFinder(bams$path, bams$sample,
  reference_path = file.path(tempdir(), "Reference"),
  output_path = file.path(tempdir(), "IRFinder_output")
)

expr <- Find_IRFinder_Output(file.path(tempdir(), "IRFinder_output"))
CollateData(expr,
  reference_path = file.path(tempdir(), "Reference"),
  output_path = file.path(tempdir(), "NxtIRF_output")
)

se <- MakeSE(collate_path = file.path(tempdir(), "NxtIRF_output"))

# "Realize" NxtSE object to load all H5 assays into memory:

se <- realize_NxtSE(se)

# If COV files have been removed since the last call to CollateData()
# reassign them to the NxtSE object, for example:

covfile_path <- system.file("extdata", package = "NxtIRFcore")
covfile_df <- Find_Samples(covfile_path, ".cov")

covfile(se) <- covfile_df$path

# Check that the produced object is identical to the example NxtSE

example_se <- NxtIRF_example_NxtSE()
identical(se, example_se) # should return TRUE
```

---

make_plot_data *Construct data of percent-spliced-in (PSI) matrices and "diagonal" for heatmaps and scatter plots*

---

## Description

`make_matrix()` constructs a matrix of PSI values of the given alternative splicing events (ASEs).

`make_diagonal()` constructs a table of "average" PSI values, with samples grouped by two given conditions (e.g. "group A" and "group B") of a given condition category (e.g. condition "treatment"). See details below.

## Usage

```
make_matrix(
  se,
  event_list,
  sample_list = colnames(se),
  method = c("PSI", "logit", "Z-score"),
  depth_threshold = 10,
  logit_max = 5,
  na.percent.max = 0.1
)

make_diagonal(
  se,
  event_list = rownames(se),
  condition,
  nom_DE,
  denom_DE,
  depth_threshold = 10,
  logit_max = 5
)
```

## Arguments

| | |
|---|---|
| `se` | (Required) A NxtSE object generated by MakeSE |
| `event_list` | A character vector containing the row names of ASE events (as given by the `EventName` column of differential ASE results table using `limma_ASE()` or `DESeq_ASE()`) |
| `sample_list` | (default = `colnames(se)`) In `make_matrix()`, a list of sample names in the given experiment to be included in the returned matrix |
| `method` | In `make_matrix()`, rhe values to be returned (default = "PSI"). It can alternately be "logit" which returns logit-transformed PSI values, or "Z-score" which returns Z-score-transformed PSI values |
| `depth_threshold` | (default = 10) Samples with the number of reads supporting either included or excluded isoforms below this values are excluded |
| `logit_max` | (default = 5) PSI values close to 0 or 1 are rounded up/down to `plogis(-logit_max)` and `plogis(logit_max)`, respectively. See details. |

| | |
|---|---|
| na.percent.max | (default = 0.1) The maximum proportion of values in the given dataset that were transformed to NA because of low splicing depth. ASE events where there are a higher proportion (default 10%) NA values will be excluded from the final matrix. Most heatmap functions will spring an error if there are too many NA values in any given row. This option caps the number of NA values to avoid returning this error. |
| condition | The name of the column containing the condition values in colData(se) |
| nom_DE | The condition to be contrasted, e.g. nom_DE = "treatment" |
| denom_DE | The condition to be contrasted against, e.g. denom_DE = "control" |

## Details

Note that this function takes the geometric mean of PSI, by first converting all values to logit(PSI), taking the average logit(PSI) values of each condition, and then converting back to PSI using inverse logit.

Samples with low splicing coverage (either due to insufficient sequencing depth or low gene expression) are excluded from calculation of mean PSIs. The threshold can be set using depth_threshold. Excluding these samples is appropriate because the uncertainty of PSI is high when the total included / excluded count is low. Note that events where all samples in a condition is excluded will return a value of NaN.

Using logit-transformed PSI values is appropriate because PSI values are bound to the (0,1) interval, and are often thought to be beta-distributed. The link function often used with beta-distributed models is the logit function, which is defined as logit(x) = function(x) log(x / (1 - x)), and is equivalent to stats::qlogis. Its inverse is equivalent to stats::plogis.

Users wishing to calculate arithmetic means of PSI are advised to use make_matrix, followed by rowMeans on subsetted sample columns.

## Value

For make_matrix: A matrix of PSI (or alternate) values, with columns as samples and rows as ASE events.

For make_diagonal: A 3 column data frame, with the first column containing event_list list of ASE events, and the last 2 columns containing the average PSI values of the nominator and denominator conditions.

## Functions

- make_matrix: constructs a matrix of PSI values of the given alternative splicing events (ASEs)
- make_diagonal: constructs a table of "average" PSI values

## Examples

```
se <- NxtIRF_example_NxtSE()

colData(se)$treatment <- rep(c("A", "B"), each = 3)

event_list <- rowData(se)$EventName
```

```
mat <- make_matrix(se, event_list[1:10])

diag_values <- make_diagonal(se, event_list,
  condition = "treatment", nom_DE = "A", denom_DE = "B"
)
```

---

Mappability-methods *Calculate low mappability genomic regions*

---

## Description

These functions empirically calculate low-mappability (Mappability Exclusion) regions using the given genome FASTA file. A splice-aware alignment software capable of aligning reads to the genome is required. See details and examples below.

## Usage

```
Mappability_GenReads(
  reference_path,
  read_len = 70,
  read_stride = 10,
  error_pos = 35,
  verbose = TRUE,
  alt_fasta_file
)

Mappability_CalculateExclusions(
  reference_path,
  aligned_bam = file.path(reference_path, "Mappability", "Aligned.out.bam"),
  threshold = 4,
  n_threads = 1
)
```

## Arguments

| | |
|---|---|
| reference_path | The directory of the reference prepared by GetReferenceResource() |
| read_len | The nucleotide length of the synthetic reads |
| read_stride | The nucleotide distance between adjacent synthetic reads |
| error_pos | The position of the procedurally-generated nucleotide error from the start of each synthetic reads |
| verbose | Whether additional status messages are shown |
| alt_fasta_file | (Optional) The path to the user-supplied genome fasta file, if different to that found inside the resource subdirectory of the reference_path. If GetReferenceResource() has already been run, this parameter should be omitted. |

| | |
|---|---|
| aligned_bam | The BAM file of alignment of the synthetic reads generated by Mappability_GenReads(). Users should use a genome splice-aware aligner, preferably the same aligner used to align the samples in their experiment. |
| threshold | Genomic regions with this alignment read depth (or below) in the aligned synthetic read BAM are defined as low mappability regions. |
| n_threads | The number of threads used to calculate mappability exclusion regions from aligned bam file of synthetic reads. |

## Details

Creating a Mappability Exclusion BED file is a three-step process.

- First, using Mappability_GenReads(), synthetic reads are systematically generated using the given genome contained within reference_path.
- Second, an aligner such as STAR (preferably the same aligner used for the subsequent RNA-seq experiment) is required to align these reads to the source genome. Poorly mapped regions of the genome will be reflected by regions of low coverage depth.
- Finally, the BAM file containing the aligned reads is analysed using Mappability_CalculateExclusions(), to identify low-mappability regions to compile the Mappability Exclusion BED file.

It is recommended to leave all parameters to their default settings. Regular users should only specify reference_path, aligned_bam and n_threads, as required.

NB: STAR_Mappability runs all 3 steps required, using the STAR aligner. This only works in systems where STAR is installed.

NB2: In systems where STAR is not available, consider using HISAT2 or Rsubread. A working example using Rsubread is shown below.

## Value

- For Mappability_GenReads: writes Reads.fa to the Mappability subdirectory inside the given reference_path.
- For Mappability_CalculateExclusions: writes a gzipped BED file named MappabilityExclusion.bed.gz to the Mappability subdirectory inside reference_path. This BED file is automatically used by BuildReference() if MappabilityRef is not specified.

## Functions

- Mappability_GenReads: Generates synthetic reads from a genome FASTA file, for mappability calculations.
- Mappability_CalculateExclusions: Generate a BED file defining low mappability regions, using reads generated by Mappability_GenReads(), aligned to the genome.

## See Also

BuildReference

## Examples

```
# (1a) Creates genome resource files

ref_path <- file.path(tempdir(), "Reference")

GetReferenceResource(
    reference_path = ref_path,
    fasta = chrZ_genome(),
    gtf = chrZ_gtf()
)

# (1b) Systematically generate reads based on the NxtIRF example genome:

Mappability_GenReads(
    reference_path = ref_path
)
## Not run:

# (2) Align the generated reads using Rsubread:

# (2a) Build the Rsubread genome index:

setwd(ref_path)
Rsubread::buildindex(basename = "./reference_index",
    reference = chrZ_genome())

# (2b) Align the synthetic reads using Rsubread::subjunc()

Rsubread::subjunc(
    index = "./reference_index",
    readfile1 = file.path(ref_path, "Mappability", "Reads.fa"),
    output_file = file.path(ref_path, "Mappability", "AlignedReads.bam"),
    useAnnotation = TRUE,
    annot.ext = chrZ_gtf(),
    isGTF = TRUE
)

# (3) Analyse the aligned reads in the BAM file for low-mappability regions:

Mappability_CalculateExclusions(
    reference_path = ref_path,
    aligned_bam = file.path(ref_path, "Mappability", "AlignedReads.bam")
)

# (4) Build the NxtIRF reference using the calculated Mappability Exclusions

BuildReference(ref_path)

# NB the default is to search for the BED file generated by
# `Mappability_CalculateExclusions()` in the given reference_path

## End(Not run)
```

NxtFilter-class *NxtIRF filters to remove low-abundance alternative splicing and intron retention events*

## Description

NxtIRF filters to remove low-abundance alternative splicing and intron retention events

## Usage

```
NxtFilter(
  filterClass = c("Data", "Annotation"),
 filterType = c("Depth", "Coverage", "Consistency", "Protein_Coding", "NMD", "TSL",
    "Terminus", "ExclusiveMXE"),
  pcTRUE = 100,
  minimum = 20,
  maximum = 1,
  minDepth = 5,
  condition = "",
  minCond = -1,
  EventTypes = c("IR", "MXE", "SE", "A3SS", "A5SS", "AFE", "ALE", "RI")
)
```

## Arguments

| | |
|---|---|
| filterClass | Must be either "Data" or "Annotation". See details |
| filterType | If filterClass = "Data", then must be one of c("Depth", "Coverage", "Consistency"). If filterClass = "Annotation", must be one of c("Protein_Coding", "NMD", "TSL"). See details |
| pcTRUE | If conditions are set, what percentage of all samples in each of the condition must the filter be satisfied for the event to pass the filter check. Must be between 0 and 100 (default 100) |
| minimum | Filter-dependent argument. See details |
| maximum | Filter-dependent argument. See details |
| minDepth | Filter-dependent argument. See details |
| condition | (default "") If set, must match the name of an experimental condition in the NxtSE object to be filtered, i.e. a column name in colData(se). Leave blank to disable filtering by condition |
| minCond | (default -1) If condition is set, how many minimum number of conditions must pass the filter criteria. For example, if condition = "Batch", and batches are "A", "B", or "C", setting minCond = 2 with pcTRUE = 100 means that all samples belonging to two of the three types of Batch must pass the filter criteria. Setting -1 means all elements of condition must pass criteria. Set to -1 when the number of elements in the experimental condition is unknown. Ignored if condition is left blank. |

`EventTypes` What types of events are considered for filtering. Must be one of `c("IR", "MXE", "SE", "A3SS", "A5SS", "AFE", "ALE", "RI")`. Events not specified in `EventTypes` are not filtered (i.e. they will pass the filter without checks)

## Details

### Annotation Filters

- **Protein_Coding**: Filters for alternative splicing or IR events involving protein-coding transcripts. No additional parameters required.
- **NMD**: Filters for events in which one isoform is a predicted NMD substrate.
- **TSL**: filters for events in which both isoforms have a TSL level below or equal to `minimum`
- **Terminus** (New as of version 1.1.1): In alternate first exons, the splice junction must not be shared with another transcript for which it is not its first intron. For alternative last exons, the splice junction must not be shared with another transcript for which it is not its last intron
- **ExclusiveMXE** (New as of version 1.1.1): For MXE events, the two alternate casette exons must not overlap in their genomic regions

### Data Filters

- **Depth**: Filters IR or alternative splicing events of transcripts that are "expressed" with adequate `Depth` as calculated by the sum of all splicing and IR reads spanning the event. Events with `Depth` below `minimum` are filtered out
- **Coverage**: Coverage means different things to IR and alternative splicing.

  For **IR**, Coverage refers to the percentage of the measured intron covered with reads. Introns of samples with an IntronDepth above `minDepth` are assessed, with introns with coverage below `minimum` are filtered out.

  For **Alternative Splicing**, Coverage refers to the percentage of all splicing events observed across the genomic region that is compatible with either the included or excluded event. This prevents NxtIRF from doing differential analysis between two minor isoforms. Instead of IntronDepth, in AS events NxtIRF considers events where the spliced reads from both exonic regions exceed `minDepth`. Then, events with a splicing coverage below `minimum` are excluded.

  We recommend testing IR events for > 90% coverage and AS events for > 60% coverage as given in the default filters which can be accessed using get_default_filters

- **Consistency**: Skipped exons (SE) and mutually exclusive exons (MXE) comprise reads aligned to two contiguous splice junctions. Most algorithms take the average counts from both junctions. This will inadvertently include transcripts that share one but not both splice events. To check that this is not happening, we require both splice junctions to have comparable counts. This filter checks whether reads from each splice junction comprises a reasonable proportion of the sum of these reads.

  Events are excluded if either of the upstream or downstream event is lower than total splicing events by a log-2 magnitude above `maximum`. For example, if `maximum = 2`, we require both

upstream and downstream events to represent at least 1/(2^2) = 1/4 of the sum of upstream and downstream event. If `maximum = 3`, then each junction must be at least 1/8 of total, etc. This is considered for each isoform of each event, as long as the total counts belonging to the considered isoform is above `minDepth`.

IR-events are also checked. For IR events, the upstream and downstream exon-intron spanning reads must comprise a reasonable proportion of total exon-intron spanning reads.

We highly recommend using the default filters, which can be acquired using get_default_filters

## Value

A NxtFilter object with the specified parameters

## Functions

- `NxtFilter`: Constructs a NxtFilter object

## See Also

Run_NxtIRF_Filters

## Examples

```
# Create a NxtFilter that filters for protein-coding ASE
f1 <- NxtFilter(filterClass = "Annotation", filterType = "Protein_Coding")

# Create a NxtFilter that filters for Depth >= 20 in IR events
f2 <- NxtFilter(
    filterClass = "Data", filterType = "Depth",
    minimum = 20, EventTypes = c("IR", "RI")
)

# Create a NxtFilter that filters for Coverage > 60% in splice events
# that must be satisfied in at least 2 categories of condition "Genotype"
f3 <- NxtFilter(
    filterClass = "Data", filterType = "Coverage",
    minimum = 60, EventTypes = c("MXE", "SE", "AFE", "ALE", "A3SS", "A5SS"),
    condition = "Genotype", minCond = 2
)

# Create a NxtFilter that filters for Depth > 10 in all events
# that must be satisfied in at least 50% of each gender
f4 <- NxtFilter(
    filterClass = "Data", filterType = "Depth",
    minimum = 10, condition = "gender", pcTRUE = 50
)

# Get a description of what these filters do:
f1
f2
f3
```

```
f4
```

---

`NxtSE-class` *The NxtSE class*

---

## Description

The NxtSE class inherits from the SummarizedExperiment class and is constructed from MakeSE. NxtSE extends SummarizedExperiment by housing additional assays pertaining to IR and splice junction counts.

## Usage

```
NxtSE(...)

## S4 method for signature 'NxtSE'
up_inc(x, withDimnames = TRUE, ...)

## S4 method for signature 'NxtSE'
down_inc(x, withDimnames = TRUE, ...)

## S4 method for signature 'NxtSE'
up_exc(x, withDimnames = TRUE, ...)

## S4 method for signature 'NxtSE'
down_exc(x, withDimnames = TRUE, ...)

## S4 method for signature 'NxtSE'
covfile(x, withDimnames = TRUE, ...)

## S4 method for signature 'NxtSE'
sampleQC(x, withDimnames = TRUE, ...)

## S4 method for signature 'NxtSE'
ref(x, withDimnames = TRUE, ...)

## S4 method for signature 'NxtSE'
realize_NxtSE(x, withDimnames = TRUE, ...)

## S4 replacement method for signature 'NxtSE'
up_inc(x, withDimnames = TRUE) <- value

## S4 replacement method for signature 'NxtSE'
down_inc(x, withDimnames = TRUE) <- value

## S4 replacement method for signature 'NxtSE'
```

```
up_exc(x, withDimnames = TRUE) <- value

## S4 replacement method for signature 'NxtSE'
down_exc(x, withDimnames = TRUE) <- value

## S4 replacement method for signature 'NxtSE'
covfile(x, withDimnames = TRUE) <- value

## S4 replacement method for signature 'NxtSE'
sampleQC(x, withDimnames = TRUE) <- value

## S4 method for signature 'NxtSE,ANY,ANY,ANY'
x[i, j, ..., drop = TRUE]

## S4 replacement method for signature 'NxtSE,ANY,ANY,NxtSE'
x[i, j, ...] <- value

## S4 method for signature 'NxtSE'
cbind(..., deparse.level = 1)

## S4 method for signature 'NxtSE'
rbind(..., deparse.level = 1)
```

## Arguments

| | |
|---|---|
| `...` | In NxtSE(), additional arguments to be passed onto SummarizedExperiment() |
| `x` | A NxtSE object |
| `withDimnames` | (default TRUE) Whether exported assays should be supplied with row and column names of the NxtSE object. See SummarizedExperiment |
| `value` | The value to replace. Must be a matrix for the up_inc<-, down_inc<-, up_exc<- and down_exc<- replacers, and a character vector for covfile<- |
| `i, j` | Row and column subscripts to subset a NxtSE object. |
| `drop` | A logical(1), ignored by these methods. |
| `deparse.level` | See base::cbind for a description of this argument. |

## Value

See Functions section (below) for details

## Functions

- `NxtSE`: Constructor function for NxtSE; akin to SummarizedExperiment(...)
- `up_inc,NxtSE-method`: Gets upstream included events (SE/MXE), or upstream exon-intron spanning reads (IR)
- `down_inc,NxtSE-method`: Gets downstream included events (SE/MXE), or downstream exon-intron spanning reads (IR)
- `up_exc,NxtSE-method`: Gets upstream excluded events (MXE only)

- down_exc,NxtSE-method: Gets downstream excluded events (MXE only)
- covfile,NxtSE-method: Gets a named vector with the paths to the corresponding COV files
- sampleQC,NxtSE-method: Gets a data frame with the QC parameters of the samples
- ref,NxtSE-method: Retrieves a list of annotation data associated with this NxtSE object; primarily used in Plot_Coverage()
- realize_NxtSE,NxtSE-method: Converts all DelayedMatrix assays as matrices (i.e. performs all delayed calculation and loads resulting object to RAM)
- up_inc<-,NxtSE-method: Sets upstream included events (SE/MXE), or upstream exon-intron spanning reads (IR)
- down_inc<-,NxtSE-method: Sets downstream included events (SE/MXE), or downstream exon-intron spanning reads (IR)
- up_exc<-,NxtSE-method: Sets upstream excluded events (MXE only)
- down_exc<-,NxtSE-method: Sets downstream excluded events (MXE only)
- covfile<-,NxtSE-method: Sets the paths to the corresponding COV files
- sampleQC<-,NxtSE-method: Sets the values in the data frame containing sample QC
- [,NxtSE,ANY,ANY,ANY-method: Subsets a NxtSE object
- [<-,NxtSE,ANY,ANY,NxtSE-method: Sets a subsetted NxtSE object
- cbind,NxtSE-method: Combines two NxtSE objects (by samples - columns)
- rbind,NxtSE-method: Combines two NxtSE objects (by AS/IR events - rows)

## Examples

```
# Run the full pipeline to generate a NxtSE object:

BuildReference(
    reference_path = file.path(tempdir(), "Reference"),
    fasta = chrZ_genome(),
    gtf = chrZ_gtf()
)

bams <- NxtIRF_example_bams()
IRFinder(bams$path, bams$sample,
  reference_path = file.path(tempdir(), "Reference"),
  output_path = file.path(tempdir(), "IRFinder_output")
)

expr <- Find_IRFinder_Output(file.path(tempdir(), "IRFinder_output"))
CollateData(expr,
  reference_path = file.path(tempdir(), "Reference"),
  output_path = file.path(tempdir(), "NxtIRF_output")
)

se <- MakeSE(collate_path = file.path(tempdir(), "NxtIRF_output"))

# Coerce NxtSE -> SummarizedExperiment
se_raw <- as(se, "SummarizedExperiment")
```

```
# Coerce SummarizedExperiment -> NxtSE
se_NxtSE <- as(se_raw, "NxtSE")
identical(se, se_NxtSE) # Returns TRUE

# Get Junction reads of SE / MXE and spans-reads of IR events
up_inc(se)
down_inc(se)
up_exc(se)
down_exc(se)

# Get list of available coverage files
covfile(se)

# Get sample QC information
sampleQC(se)

# Get resource NxtIRF data (used internally for Plot_Coverage())
cov_data <- ref(se)
names(cov_data)

# Subset functions
se_by_samples <- se[,1:3]
se_by_events <- se[1:10,]
se_by_rowData <- subset(se, EventType == "IR")

# Cbind (bind event_identical NxtSE by samples)
se_by_samples_1 <- se[,1:3]
se_by_samples_2 <- se[,4:6]
se_cbind <- cbind(se_by_samples_1, se_by_samples_2)
identical(se, se_cbind) # should return TRUE

# Rbind (bind sample_identical NxtSE by events)
se_IR <- subset(se, EventType == "IR")
se_SE <- subset(se, EventType == "SE")
se_IRSE <- rbind(se_IR, se_SE)
identical(se_IRSE, subset(se, EventType %in% c("IR", "SE"))) # TRUE

# Convert HDF5-based NxtSE to in-memory se
# MakeSE() creates a HDF5-based NxtSE object where all assay data is stored
# as an h5 file instead of in-memory. All operations are performed as
# delayed operations as per DelayedArray package.
# To realize the NxtSE object as an in-memory object, use:

se_real <- realize_NxtSE(se)
identical(se, se_real) # should return FALSE

# To check the difference, run:
class(up_inc(se))
class(up_inc(se_real))
```

Plot_Coverage *RNA-seq Coverage Plots and Genome Tracks*

## Description

Generate plotly / ggplot RNA-seq genome and coverage plots from command line. For some quick working examples, see the Examples section below.

## Usage

```
Plot_Coverage(
  se,
  Event,
  Gene,
  seqname,
  start,
  end,
  coordinates,
  strand = c("*", "+", "-"),
  zoom_factor,
  bases_flanking = 100,
  tracks,
  track_names = tracks,
  condition,
  selected_transcripts,
  condense_tracks = FALSE,
  stack_tracks = FALSE,
  t_test = FALSE,
  norm_event
)

Plot_Genome(
  se,
  reference_path,
  Gene,
  seqname,
  start,
  end,
  coordinates,
  zoom_factor,
  bases_flanking = 100,
  selected_transcripts,
  condense_tracks = FALSE
)

as_egg_ggplot(p_obj)
```

## Arguments

| | |
|---|---|
| se | A NxtSE object, created by MakeSE. COV files must be linked to the NxtSE object. To do this, see the example in MakeSE. Required by Plot_Coverage. |
| Event | The EventName of the IR / alternative splicing event to be displayed. Use rownames(se) to display a list of valid events. |
| Gene | Whether to use the range for the given Gene. If given, overrides Event (but Event or norm_event will be used to normalise by condition). Valid Gene entries include gene_id (Ensembl ID) or gene_name (Gene Symbol). |
| seqname, start, end | The chromosome (string) and genomic start/end coordinates (numeric) of the region to display. If present, overrides both Event and Gene. E.g. for a given region of chr1:10000-11000, use the parameters: seqname = "chr1", start = 10000, end = 11000 |
| coordinates | A string specifying genomic coordinates can be given instead of seqname,start,end. Must be of the format "chr:start-end", e.g. "chr1:10000-11000" |
| strand | Whether to show coverage of both strands "*" (default), or from the "+" or "-" strand only. |
| zoom_factor | Zoom out from event. Each level of zoom zooms out by a factor of 3. E.g. for a query region of chr1:10000-11000, if a zoom_factor of 1.0 is given, chr1:99000-12000 will be displayed. |
| bases_flanking | (Default = 100) How many bases flanking the zoomed window. Useful when used in conjunction with zoom_factor == 0. E.g. for a given region of chr1:10000-11000, if zoom_factor = 0 and bases_flanking = 100, the region chr1:9900-11100 will be displayed. |
| tracks | The names of individual samples, or the names of the different conditions to be plotted. For the latter, set condition to the specified condition category. |
| track_names | The names of the tracks to be displayed. If omitted, the track_names will default to the input in tracks |
| condition | To display normalised coverage per condition, set this to the condition category. If omitted, tracks are assumed to refer to the names of individual samples. |
| selected_transcripts | (Optional) A vector containing transcript ID or transcript names of transcripts to be displayed on the gene annotation track. Useful to remove minor isoforms that are not relevant to the samples being displayed. |
| condense_tracks | (default FALSE) Whether to collapse the transcript track annotations by gene. |
| stack_tracks | (default FALSE) Whether to graph all the conditions on a single coverage track. If set to TRUE, each condition will be displayed in a different colour on the same track. Ignored if condition is not set. |
| t_test | (default FALSE) Whether to perform a pair-wise T-test. Only used if there are TWO condition tracks. |
| norm_event | Whether to normalise by an event different to that given in "Event". The difference between this and Event is that the genomic coordinates can be centered around a different Event, Gene or region as given in seqname/start/end. If norm_event is different to Event, norm_event will be used for normalisation |

and Event will be used to define the genomic coordinates of the viewing window. norm_event is required if Event is not set and condition is set.

reference_path The path of the reference generated by BuildReference. Required by Plot_Genome if a NxtSE object is not specified.

p_obj In as_egg_ggplot, takes the output of Plot_Coverage and plots all tracks in a static plot using ggarrange function of the egg package. Requires egg to be installed.

## Details

In RNA sequencing, alignments to spliced transcripts will "skip" over genomic regions of introns. This can be illustrated in a plot using a horizontal genomic axis, with the vertical axis representing the number of alignments covering each nucleotide. As a result, the coverage "hills" represent the expression of exons, and "valleys" to introns.

Different alternatively-spliced isoforms thus produce different coverage patterns. The change in the coverage across an alternate exon relative to its constitutively-included flanking exons, for example, represents its alternative inclusion or skipping. Similarly, elevation of intron valleys represent increased intron retention.

With multiple replicates per sample, coverage is dependent on library size and gene expression. To compare alternative splicing ratios, normalisation of the coverage of the alternate exon (or alternatively retained intron) relative to their constitutive flanking exons, is required. There is no established method for this normalisation, and can be confounded in situations where flanking elements are themselves alternatively spliced.

NxtIRF performs this coverage normalisation using the same method as its estimate of spliced / intronic transcript abundance using the SpliceOverMax method (see details section in CollateData). This normalisation can be applied to correct for library size and gene expression differences between samples of the same experimental condition. After normalisation, mean and variance of coverage can be computed as ratios relative to total transcript abundance. This method can visualise alternatively included genomic regions including casette exons, alternate splice site usage, and intron retention.

Plot_Coverage generates plots showing depth of alignments to the genomic axis. Plots can be generated for individual samples or samples grouped by experimental conditions. In the latter, mean and 95% confidence intervals are shown.

Plot_Genome generates genome transcript tracks only. Protein-coding regions are denoted by thick rectangles, whereas non-protein coding transcripts or untranslated regions are denoted with thin rectangles. Introns are denoted as lines.

## Value

A list containing two objects. final_plot is the plotly object. ggplot is a list of ggplot tracks, with:

- ggplot[[n]] is the nth track (where n = 1, 2, 3 or 4).
- ggplot[[5]] contains the T-test track if one is generated.
- ggplot[[6]] always contains the genome track.

## Functions

- Plot_Coverage: generates plots showing depth of alignments to the genomic axis. Plots can be generated for individual samples or samples grouped by experimental conditions. In the latter, mean and 95 intervals are shown.
- Plot_Genome: Generates a plot of transcripts within a given genomic region, or belonging to a specified gene
- as_egg_ggplot: Coerce the 'Plot_Coverage()' output as a vertically stacked ggplot, using egg::ggarrange

## Examples

```
se <- NxtIRF_example_NxtSE()

# Plot the genome track only, with specified gene:
p <- Plot_Genome(se, Gene = "SRSF3")
p$ggplot

# View the genome track, specifying a genomic region via coordinates:
p <- Plot_Genome(se, coordinates = "chrZ:10000-20000")
p$ggplot

# Assign annotation re experimental conditions

colData(se)$treatment <- rep(c("A", "B"), each = 3)

# Verify that the COV files are linked to the NxtSE object:
covfile(se)

# Return a list of ggplot and plotly objects
p <- Plot_Coverage(
    se = se,
    Event = rowData(se)$EventName[1],
    tracks = colnames(se)[1:4]
)

# Display a static ggplot / egg::ggarrange stacked plot:

as_egg_ggplot(p)

# Display the plotly-based interactive Coverage plot:
p$final_plot

# Plot the same event but by condition "treatment"
p <- Plot_Coverage(
    se, rowData(se)$EventName[1],
    tracks = c("A", "B"), condition = "treatment"
)
as_egg_ggplot(p)
```

`Run_NxtIRF_Filters` *Filtering for IR and Alternative Splicing Events*

## Description

This function implements filtering of IR or AS events based on customisable criteria. See NxtFilter for details.

## Usage

```
get_default_filters(legacy = FALSE)

apply_filters(se, filters = get_default_filters())

runFilter(se, filterObj)
```

## Arguments

| | |
|---|---|
| `legacy` | (default `FALSE`) Set to `TRUE` to get the first four default filters introduced in the initial NxtIRFcore release. |
| `se` | the NxtSE object to filter |
| `filters` | A vector or list of one or more NxtFilter objects. If left blank, the NxtIRF default filters will be used. |
| `filterObj` | A single NxtFilter object. |

## Details

We highly recommend using the default filters, which are as follows:

- (1) Depth filter of 20,
- (2) Coverage filter requiring 90% coverage in IR events.
- (3) Coverage filter requiring 60% coverage in AS events (i.e. Included + Excluded isoforms must cover at least 60% of all junction events across the given region)
- (4) Consistency filter requring log difference of 2 (for skipped exon and mutually exclusive exon events, each junction must comprise at least 1/(2^2) = 1/4 of all reads associated with each isoform). For retained introns, the exon-intron overhangs must not differ by 1/4

Also, in NxtIRFcore version 1.1.1 and above, we introduced two annotation-based filters:

- (5) Terminus filter: In alternate first exons, the splice junction must not be shared with another transcript for which it is not its first intron. For alternative last exons, the splice junction must not be shared with another transcript for which it is not its last intron
- (6) ExclusiveMXE filter: For MXE events, the two alternate casette exons must not overlap in their genomic regions

In all data-based filters, we require at least 80% samples (`pcTRUE = 80`) to pass this filters from the entire dataset (`minCond = -1`).

Events with event read depth (reads supporting either included or excluded isoforms) lower than 5 (`minDepth = 5`) are not assessed in filter #2, and in #3 and #4 this threshold is (`minDepth = 20`).

For an explanation of the various parameters mentioned here, see NxtFilter

## Value

For `runFilter` and `apply_filters`: a vector of type `logical`, representing the rows of NxtSE that should be kept.

For `get_default_filters`: returns a list of default recommended filters that should be parsed into `apply_filters`.

## Functions

- `get_default_filters`: Returns a vector of recommended default NxtIRF filters
- `apply_filters`: Run a vector or list of NxtFilter objects on a NxtSE object
- `runFilter`: Run a single filter on a NxtSE object

## See Also

NxtFilter for details describing how to create and assign settings to NxtFilter objects.

## Examples

```
# see ?MakeSE on example code of how this object was generated

se <- NxtIRF_example_NxtSE()

# Get the list of NxtIRF recommended filters

filters <- get_default_filters()

# View a description of what these filters do:

filters

# Filter the NxtSE using the first default filter ("Depth")

se.depthfilter <- se[runFilter(se, filters[[1]]), ]

# Filter the NxtSE using all four default filters

se.defaultFiltered <- se[apply_filters(se, get_default_filters()), ]
```

STAR-methods *STAR wrapper for building reference for STAR, and aligning RNA-sequencing*

## Description

These functions run the STAR aligner to build a STAR genome reference, calculate mappability exclusion regions using STAR, and align one or more FASTQ files (single or paired) to the generated genome. These functions only work on Linux-based systems with STAR installed. STAR must be accessible via $PATH. See details and examples

## Usage

```
STAR_version()

STAR_buildRef(
  reference_path,
  STAR_ref_path = file.path(reference_path, "STAR"),
  also_generate_mappability = TRUE,
  map_depth_threshold = 4,
  sjdbOverhang = 149,
  n_threads = 4,
  additional_args = NULL,
  ...
)

STAR_Mappability(
  reference_path,
  STAR_ref_path = file.path(reference_path, "STAR"),
  map_depth_threshold = 4,
  n_threads = 4,
  ...
)

STAR_align_experiment(
  Experiment,
  STAR_ref_path,
  BAM_output_path,
  trim_adaptor = "AGATCGGAAG",
  two_pass = FALSE,
  n_threads = 4
)

STAR_align_fastq(
  fastq_1 = c("./sample_1.fastq"),
  fastq_2 = NULL,
  STAR_ref_path,
```

```
  BAM_output_path,
  two_pass = FALSE,
  trim_adaptor = "AGATCGGAAG",
  memory_mode = "NoSharedMemory",
  additional_args = NULL,
  n_threads = 4
)
```

## Arguments

| | |
|---|---|
| reference_path | The path to the reference. GetReferenceResource must first be run using this path as its reference_path |
| STAR_ref_path | (Default - the "STAR" subdirectory under reference_path) The directory containing the STAR reference to be used or to contain the newly-generated STAR reference |
| also_generate_mappability | Whether STAR_buildRef() also calculates Mappability Exclusion regions. |
| map_depth_threshold | (Default 4) The depth of mapped reads threshold at or below which Mappability exclusion regions are defined. See Mappability-methods. Ignored if also_generate_mappability = FALSE |
| sjdbOverhang | (Default = 149) A STAR setting indicating the length of the donor / acceptor sequence on each side of the junctions. Ideally equal to (mate_length - 1). As the most common read length is 150, the default of this function is 149. See the STAR aligner manual for details. |
| n_threads | The number of threads to run the STAR aligner. |
| additional_args | A character vector of additional arguments to be parsed into STAR. See examples below. |
| ... | Additional arguments to be parsed into Mappability_GenReads(). See Mappability-methods. |
| Experiment | A two or three-column data frame with the columns denoting sample names, forward-FASTQ and reverse-FASTQ files. This can be conveniently generated using Find_FASTQ |
| BAM_output_path | The path under which STAR outputs the aligned BAM files. In STAR_align_experiment(), STAR will output aligned BAMS inside subdirectories of this folder, named by sample names. In STAR_align_fastq(), STAR will output directly into this path. |
| trim_adaptor | The sequence of the Illumina adaptor to trim via STAR's --clip3pAdapterSeq option |
| two_pass | Whether to use two-pass mapping. In STAR_align_experiment(), STAR will first align every sample and generate a list of splice junctions but not BAM files. The junctions are then given to STAR to generate a temporary genome (contained within _STARgenome) subdirectory within that of the first sample), using these junctions to improve novel junction detection. In STAR_align_fastq(), STAR will run --twopassMode Basic |

`fastq_1, fastq_2`
: In STAR_align_fastq: character vectors giving the path(s) of one or more FASTQ (or FASTA) files to be aligned. If single reads are to be aligned, omit `fastq_2`

`memory_mode`
: The parameter to be parsed to `--genomeLoad`; either `NoSharedMemory` or `LoadAndKeep` are used.

## Details

### Pre-requisites

`STAR_buildRef` requires GetReferenceResource to be run to fetch the required genome and gene annotation files.

`STAR_Mappability`, `STAR_align_experiment` and `STAR_align_fastq` requires a `STAR` genome, which can be built using `STAR_buildRef`

### Function Description

For `STAR_buildRef`: this function will create a `STAR` genome reference in the `STAR` subdirectory in the path given by `reference_path`. Optionally, it will run STAR_Mappability if `also_generate_mappability` is set to `TRUE`

For `STAR_Mappability`: this function will first will run Mappability_GenReads, then use the given `STAR` genome to align the synthetic reads using `STAR`. The aligned BAM file will then be processed using Mappability_CalculateExclusions to calculate the lowly-mappable genomic regions, producing the `MappabilityExclusion.bed.gz` output file.

For `STAR_align_fastq`: aligns a single or pair of FASTQ files to the given `STAR` genome using the `STAR` aligner.

For `STAR_align_experiment`: aligns a set of FASTQ or paired FASTQ files using the given `STAR` genome using the `STAR` aligner. A data.frame specifying sample names and corresponding FASTQ files are required

## Value

None. STAR will output files into the given output directories.

## Functions

- `STAR_version`: Checks whether STAR is installed, and its version
- `STAR_buildRef`: Creates a STAR genome reference.
- `STAR_Mappability`: Calculates lowly-mappable genomic regions using STAR
- `STAR_align_experiment`: Aligns multiple sets of FASTQ files, belonging to multiple samples
- `STAR_align_fastq`: Aligns a single sample (with single or paired FASTQ or FASTA files)

## See Also

BuildReference Find_Samples Mappability-methods

The latest STAR documentation

## Examples

```
# 0) Check that STAR is installed and compatible with NxtIRF

STAR_version()
## Not run:

# The below workflow illustrates
# 1) Getting the reference resource
# 2) Building the STAR Reference, including Mappability Exclusion calculation
# 3) Building the NxtIRF Reference, using the Mappability Exclusion file
# 4) Aligning (a) one or (b) multiple raw sequencing samples.


# 1) Reference generation from Ensembl's FTP links

FTP <- "ftp://ftp.ensembl.org/pub/release-94/"

GetReferenceResource(
    reference_path = "Reference_FTP",
    fasta = paste0(FTP, "fasta/homo_sapiens/dna/",
        "Homo_sapiens.GRCh38.dna.primary_assembly.fa.gz"),
    gtf = paste0(FTP, "gtf/homo_sapiens/",
        "Homo_sapiens.GRCh38.94.chr.gtf.gz")
)

# 2) Generates STAR genome within the NxtIRF reference. Also generates
# mappability exclusion gzipped BED file inside the "Mappability/" sub-folder

STAR_buildRef(
    reference_path = "Reference_FTP",
    n_threads = 8,
    also_generate_mappability = TRUE
)

# 2 alt) Generates STAR genome of the example NxtIRF genome.
#     This demonstrates using custom STAR parameters, as the example NxtIRF
#     genome is ~100k in length, so --genomeSAindexNbases needs to be
#     adjusted to be min(14, log2(GenomeLength)/2 - 1)

GetReferenceResource(
    reference_path = "Reference_chrZ",
    fasta = chrZ_genome(),
    gtf = chrZ_gtf()
)

STAR_buildRef(
    reference_path = "Reference_chrZ",
    n_threads = 8,
    additional_args = c("--genomeSAindexNbases", "7"),
    also_generate_mappability = TRUE
)
```

```
# 3) Build NxtIRF reference using the newly-generated Mappability exclusions

#' NB: also specifies to use the hg38 nonPolyA resource

BuildReference(reference_path = "Reference_FTP", genome_type = "hg38")

# 4a) Align a single sample using the STAR reference

STAR_align_fastq(
    STAR_ref_path = file.path("Reference_FTP", "STAR"),
    BAM_output_path = "./bams/sample1",
    fastq_1 = "sample1_1.fastq", fastq_2 = "sample1_2.fastq",
    n_threads = 8
)

# 4b) Align multiple samples, using two-pass alignment

Experiment <- data.frame(
    sample = c("sample_A", "sample_B"),
    forward = file.path("raw_data", c("sample_A", "sample_B"),
        c("sample_A_1.fastq", "sample_B_1.fastq")),
    reverse = file.path("raw_data", c("sample_A", "sample_B"),
        c("sample_A_2.fastq", "sample_B_2.fastq"))
)

STAR_align_experiment(
    Experiment = Experiment,
    STAR_ref_path = file.path("Reference_FTP", "STAR"),
    BAM_output_path = "./bams",
    two_pass = TRUE,
    n_threads = 8
)

## End(Not run)
```

---

theme_white *ggplot2 themes*

---

## Description

A ggplot theme object for white background figures +/- a legend

## Usage

```
theme_white

theme_white_legend

theme_white_legend_plot_track
```

## Format

An object of class `theme` (inherits from gg) of length 10.

An object of class `theme` (inherits from gg) of length 9.

An object of class `theme` (inherits from gg) of length 10.

## Functions

- `theme_white`: White theme without figure legend
- `theme_white_legend`: White theme but with a figure legend (if applicable)
- `theme_white_legend_plot_track`: White theme with figure legend but without horizontal grid lines. Used internally in PlotGenome

## See Also

[Plot_Coverage]

## Examples

```
library(ggplot2)
df <- data.frame(
  gp = factor(rep(letters[1:3], each = 10)),
  y = rnorm(30))
ggplot(df, aes(gp, y)) +
  geom_point() +
  theme_white
```

# Index