Package 'iscream'

November 7, 2025

Title Make fast and memory efficient BED file queries, summaries and matrices

Version 1.1.1

Description BED files store ranged genomic data that can be queried even when the files are compressed. iscream can query data from BED files and return them in muliple formats: parsed records or their summary statistics as data frames or GenomicRanges objects, and matrices as matrix, GenomicRanges, or SummarizedExperiment objects. iscream also provides specialized support for importing methylation data.

URL https://huishenlab.github.io/iscream/, https://github.com/huishenlab/iscream/

BugReports https://github.com/huishenlab/iscream/issues/

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Depends R (>= 4.5)

LinkingTo Rcpp, RcppArmadillo, RcppProgress, RcppSpdlog, Rhtslib, stringfish

Imports Rcpp, Matrix, data.table, methods, pbapply, parallelly, stringfish,

Suggests BiocFileCache, BiocStyle, bsseq, ggplot2, ggridges, knitr, microbenchmark, rmarkdown, GenomicRanges, IRanges, Rsamtools, SummarizedExperiment, S4Vectors, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation yes

SystemRequirements htslib: htslib-devel (rpm) or libhts-dev (deb) & tabix: htslib-tools (rpm) or tabix (deb) & GNU make

biocViews DataImport, Software, Sequencing, SingleCell, DNAMethylation

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

git_branch devel

git_last_commit b1bda1c

git_last_commit_date 2025-10-29

Repository Bioconductor 3.23

Date/Publication 2025-11-06

Author James Eapen [aut, cre] (ORCID: <https://orcid.org/0000-0001-6016-3598>),
Jacob Morrison [aut] (ORCID: <https://orcid.org/0000-0001-8592-4744>),
Nathan Spix [ctb],
Hui Shen [aut, ths, fnd] (ORCID:
<https://orcid.org/0000-0001-9767-4084>)

Maintainer James Eapen <james.eapen@vai.org>

Contents

check_files_exist *Check that files exist*

Description

Check that files exist

Usage

```
check_files_exist(files_vec, error_file_prefix = "Bedfile")
```

Arguments

files_vec	A vector of file paths
error_file_prefix	Error message prefix for 'Bedfile' vs 'Tabix file'

Value

TRUE if all input BED files have an associated tabix index file. FALSE if not

check_thread_count *Check that the required threads are available*

Description

Check that the required threads are available

Usage

```
check_thread_count(
  n_threads,
  avail_threads = availableCores(),
  opt_set = FALSE
)
```

Arguments

n_threads	The number of threads to check availability for
avail_threads	The number of threads that are available on the system. Defaults to `parallelly::availableCores()`
opt_set	Whether the `iscream.threads` options is set

Value

`n_threads` if the requested number of threads are available and stops if not

Cpp_query_all *Query all methylation info into M and coverage matrices*

Description

Query all methylation info into M and coverage matrices

Usage

```
Cpp_query_all(
  bedfiles,
  regions,
  aligner,
  valInd,
  merged,
  sparse,
  prealloc,
  nthreads
)
```

Arguments

bedfiles	A vector of BED files
regions	A vector of regions
aligner	The aligner used to make the WGBS BED files, only for make_mat_bsseq
valInd	The index of the data column needed for the matrix, for make_mat
merged	Whether the input strands have been merged/collapsed
prealloc	The number of rows to initialize the matrices with
nthreads	Set number of threads to use overriding the "iscream.threads" option. See ?set_threads for more information.

Value

A list of one or two matrices, chromosome, position, and filename vectors

Cpp_set_log_level *spdlog Logging Lever Setter*

Description

A helper function to turn a logging level given as string into the current logging level

Usage

```
Cpp_set_log_level(name)
```

Arguments

name	A string with the logging level. Value understood are, in decreasing verbosity 'trace', 'debug', 'info', 'warning', 'error', 'critical', and 'off'. Unrecognised names are equivalent to 'off'.

Value

Nothing is returned.

Cpp_summarize_regions *Apply a function over BED file records within genomic features*

Description

This function should be called from `summarize_regions()` since there are few sanity checks on the C++ side.

Usage

```
Cpp_summarize_regions(
  bedfiles,
  regions,
  fun_vec,
  col_indices,
  col_names,
  aligner,
  mval = FALSE,
  region_rownames = FALSE,
  nthreads = 1L
)
```

Arguments

bedfiles	A vector of BED file paths
regions	A vector of genomic regions
fun_vec	Vector of the armadillo-supported stats functions to apply over the CpGs in the ' regions: `"sum"`, `"mean"`, `"median"`, `"stddev"`, `"variance"` `"count"`, `"min"`,`"max"`, and `"range"`.
col_indices	A vector of genomic regions
col_names	A vector of genomic regions
mval	Calculates M values when TRUE, use beta values when FALSE

region_rownames
: Whether to set rownames to the regions strings. Not necessary if your regions vector is unnamed. If its names, then the "feature" column is set to the names and the rownames are set to the regions string

nthreads
: Number of cores to use. See details.

Details

The optimal number of threads depends on the number of bedfiles, but is set to half the available OpenMP cores. See `?get_threads` for more details. It can be manaully set with `set_threads()`.

Value

A summary data.frame

---

get_df_string *DataFrame to region strings*

Description

Convert DataFrame to a vector of strings. Set feature names in a "name" column

Usage

```
get_df_string(regions_df, feature_col = NULL)
```

Arguments

regions_df
: A data frame with "chr", "start" and "end" columns

feature_col
: The data frame column to use as the names of the output string vector

Value

A character vector

Examples

```
(df <- data.frame(chr = c("chr1", "chr2"), start = c(1, 5), end = c(4, 10)))
get_df_string(df)
```

get_granges_string *GRanges to region strings*

Description

Coerces GenomicRanges to `chr:start-end` strings with `as.character`. If any regions have the same start and end, `as.character` returns `chr:start` strings which are invalid for the htslib API. These are corrected to `chr:start-start`.

Usage

```
get_granges_string(gr)
```

Arguments

`gr`	A GRanges object

Value

A character vector

Examples

```
if (requireNamespace("GenomicRanges", quietly = TRUE)) {
  get_granges_string(GenomicRanges::GRanges(c("chr1:1-10", "chr2:15-20")))
}
```

get_threads *Get the number of available threads*

Description

Gets the number of threads iscream is currently set to use, whether the `"iscream.threads"` option is set and how many threads are available for use. To set the number of threads use `set_threads()` or set the `iscream.threads` option in your `~/.Rprofile`. See `?set_threads` for more information.

Usage

```
get_threads()
```

Value

A named vector:

- `use_threads` = the number of threads iscream will use
- `opt_set` = whether the option was set by the user
- `avail_threads` = The number of available threads as reported by `parallelly::availableCores`

Examples

```
get_threads()
```

`htslib_version` *Get htslib version and available features*

Description

Returns the version of htslib being used by iscream and whether features such as libdeflate support are available. This information may not always correspond to the htslib version used during iscream's installation if a different htslib version is available for linking at runtime.

Usage

```
htslib_version()
```

Value

None

Examples

```
htslib_version()
```

`make_mat` *Make a matrix from a numeric column of BED files*

Description

Queries the provided regions and produces a matrix along with genomic positions as a named list (`make_mat()`), a `RangedSummarizedExperiment` (`make_mat_se()`), `GRanges` (`make_mat_gr()`). Parallelized across files using threads from the `"iscream.threads"` option.

Usage

```
make_mat(
  bedfiles,
  regions,
  column,
  mat_name = "value",
  sparse = FALSE,
  prealloc = 10000,
  nthreads = NULL
)
```

```
make_mat_se(
  bedfiles,
  regions,
  column,
  mat_name = "value",
  sparse = FALSE,
  prealloc = 10000,
  nthreads = NULL
)

make_mat_gr(
  bedfiles,
  regions,
  column,
  mat_name = "value",
  prealloc = 10000,
  nthreads = NULL
)
```

Arguments

bedfiles	A vector of BED file paths
regions	A vector, data frame or GenomicRanges of genomic regions. See details.
column	The index of the data column needed for the matrix
mat_name	What to name the matrix in the returned object
sparse	Whether to return a sparse matrix
prealloc	The number of rows to initialize the matrices with. If the number of loci are approximately known, this can reduce runtime as fewer resizes need to be made.
nthreads	Set the number of threads to use. Overrides the "iscream.threads" option. See ?set_threads for more information.

Details

The input regions may be string vector in the form "chr:start-end" or a GRanges object. If a data frame is provided, they must have "chr", "start", and "end" columns.

Value

- make_mat(): A named list of
  - the matrix with the value of interest
  - a character vector of chromosomes and numeric vector of base positions
  - a character vector of the input sample BED file names
- make_mat_gr(): if GenomicRanges is available, a GRanges
- make_mat_se(): if SummarizedExperiment is available, a RangedSummarizedExperiment

Examples

```
bedfiles <- system.file("extdata", package = "iscream") |>
  list.files(pattern = "[a|b|c|d].bed.gz$", full.names = TRUE)
# examine the bedfiles
colnames <- c("chr", "start", "end", "beta", "coverage")
lapply(bedfiles, function(i) knitr::kable(read.table(i, col.names = colnames)))

# make a vector of regions
regions <- c("chr1:1-6", "chr1:7-10", "chr1:11-14")
# make matrix of beta values
make_mat(bedfiles, regions, column = 4)
```

`make_mat_bsseq` *Make M/beta and coverage matrices from WGBS BED files*

Description

Queries the CpG/CpH loci from provided regions and produces M/beta and coverage matrices with their genomic positions. Parallelized across files using threads from the `"iscream.threads"` option. The output of `make_mat_bsseq` may be used to create a BSseq object: `do.call(BSseq, make_mat_bsseq(...))`.

Usage

```
make_mat_bsseq(
  bedfiles,
  regions,
  aligner = "biscuit",
  mval = TRUE,
  merged = TRUE,
  sparse = FALSE,
  prealloc = 10000,
  nthreads = NULL
)
```

Arguments

`bedfiles`	A vector of BED file paths
`regions`	A vector, data frame or GenomicRanges of genomic regions. See details.
`aligner`	The aligner used to produce the BED files - one of "biscuit", "bismark", "bsbolt".
`mval`	Whether to return M-values or beta-values with the coverage matrix. Defaults to M-value. Set `mval=FALSE` to get beta value matrix.
`merged`	Whether the input strands have been merged/collapsed
`sparse`	Whether to return a sparse matrix
`prealloc`	The number of rows to initialize the matrices with. If the number of loci are approximately known, this can reduce runtime as fewer resizes need to be made.
`nthreads`	Set the number of threads to use. Overrides the `"iscream.threads"` option. See `?set_threads` for more information.

Details

The input regions may be string vector in the form "chr:start-end" or a GRanges object. If a data frame is provided, they must have "chr", "start", and "end" columns.

Value

A named list of

- coverage and either a beta- or M-value matrix
- a character vector of chromosomes and numeric vector of corresponding CpG base positions
- a character vector of the input sample names

Bitpacking limits

`make_mat_bsseq()` makes two matrices: M-value (or beta-value) and coverage. For speed and memory efficiency these two values are bitpacked during matrix creation so that only one matrix needs to be populated and resized. This matrix is unpacked into the two required matrices only after the matrix dimensions are known after querying all input files. The two values are packed using the INT16 type, which has an upper limit of 32,767, into one INT32. If the coverage values exceed 32,767, the upper limit of a 16-bit signed integer, it will be capped at the limit. Beta values will also be capped similarly, but any such beta values would indicate a bug in the aligner that produced the data.

Examples

```
bedfiles <- system.file("extdata", package = "iscream") |>
  list.files(pattern = "[a|b|c|d].bed.gz$", full.names = TRUE)
# examine the BED files
colnames <- c("chr", "start", "end", "beta", "coverage")
lapply(bedfiles, function(i) knitr::kable(read.table(i, col.names = colnames)))

# make a vector of regions
regions <- c("chr1:1-6", "chr1:7-10", "chr1:11-14")
mat <- make_mat_bsseq(bedfiles, regions)
# for BSseq object run
if (requireNamespace("bsseq", quietly = TRUE)) {
  do.call(bsseq::BSseq, mat)
}
```

`query_chroms` *Query the chromosomes or seqnames from a vector of BED files*

Description

Query the chromosomes or seqnames from a vector of BED files

Usage

```
query_chroms(bedfiles, nthreads = NULL)
```

Arguments

bedfiles	The vector of BED file paths
nthreads	Set number of threads to use overriding the "iscream.threads" option. See ?set_threads for more information.

Value

A vector of seqnames

Examples

```
bedfiles <- system.file("extdata", package = "iscream") |>
  list.files(pattern = "[a|b|c|d].bed.gz$", full.names = TRUE)
query_chroms(bedfiles)
```

---

set_log_level *Set and get logging level*

Description

Set and get logging level

Usage

```
set_log_level(level = "info")

get_log_level()
```

Arguments

level	The logging verbosity level to use • "info": the default that gives provides basic information about the number of files and regions used in a function • "debug": more verbose about row allocations, how many CpGs were found in a region, filename parsing etc. This mode cannot be used on more than one thread as R cannot output messages from multiple threads without crashing. • "off": no logging

Value

- set_log_level(): None; sets the log level to the provided level
- get_log_level(): The current logging level as a string

Examples

```
set_log_level("info")
get_log_level()
```

set_threads *Set the number of available threads*

Description

Sets the "iscream.threads" option to n_threads. To see how many threads you have available see ?get_threads().

Usage

```
set_threads(n_threads)
```

Arguments

n_threads	The number of threads to use

Details

iscream uses OpenMP to parallelize certain functions. You can use as many threads as are available to you on your system to varying degrees of performance improvements. The get_threads() function uses parallelly::availableCores() to report the number of available threads. Although OpenMP can detect the number of available cores, on high performance computers (HPCs) with resource allocating job schedulers like SLURM, OpenMP may detect all available threads across the HPC and not limit itself to the cores that were allocated to you by the scheduler. If your system administrator has not set up any limits, this may result in your job taking resources from other jobs. If there are limits, trying to use more threads that those available will reduce iscream's performance. Job schedulers will typically have an environment variable (e.g. SLURM_CPUS_ON_NODE with SLURM) that gives you the actual number of available cores. Further, on hyperthreaded systems, this count may be double that of the available processors. Using hyperthreading does not guarantee any performance improvement - it may be better to set the number of threads to half the reported number. parallelly::availableCores() takes HPC scheduler/CRAN/Bioconductor limits into account when reporting the number of available threads but it may not reliably report hyperthreading ('system' or 'nproc'). To set the number of threads without having to call set_threads() in every session, put

```
options(iscream.threads = [n_threads])
```

in your .Rprofile See help('Rprofile') for information on startup options.

Functions currently using multithreading:

- tabix(), tabix_gr(), tabix_raw()
- query_chroms()
- make_mat(), make_mat_se(), make_mat_gr(), make_mat_bsseq()
- summarize_regions(), summarize_meth_regions()

Value

None. Sets the `iscream.threads` option to the requested number of threads if available

Examples

```
(ncores <- parallelly::availableCores())

set_threads(ncores)
```

`summarize_meth_regions` *Summarize methylation information over genomic regions*

Description

Run summarizing functions on the CpG/CpH loci in BED files across genomic regions. Parallelized across files using threads from the `"iscream.threads"` option.

Usage

```
summarize_meth_regions(
  bedfiles,
  regions,
  fun = "all",
  aligner = "biscuit",
  feature_col = NULL,
  mval = TRUE,
  set_region_rownames = FALSE,
  nthreads = NULL
)
```

Arguments

`bedfiles`	A vector of BED file paths
`regions`	A vector, data frame or GenomicRanges of genomic regions. See details.
`fun`	Function(s) to apply over the region. See details.
`aligner`	The aligner used to produce the BED files - one of "biscuit", "bismark", "bsbolt".
`feature_col`	Column name of the input `regions` data frame containing a name for each genomic region. Set only if the using a data frame as the input regions format. See details.
`mval`	Whether to calculate the M value (coverage $\times\beta$) or use the beta value when applying the function.

`set_region_rownames`
: Use the region strings as the returned data frame's rownames. Can be useful if you have a named regions and want both the regions strings rownames and the feature names. See details.

`nthreads`
: Set number of threads to use overriding the `"iscream.threads"` option. See `?set_threads` for more information.

Value

A data.frame

Supported functions

- Sum: `"sum"`
- Mean: `"mean"`
- Median: `"median"`
- Standard deviation: `"stddev"`
- Variance: `"variance"`
- Minimum: `"min"`
- Maximum: `"max"`
- Range: `"range"`
- No. of records in the region: `"count"`

The summarizing computations are backed by the Armadillo library. See https://arma.sourceforge.net/docs.html#stats_fns for futher details on the supported functions

Using feature identifiers

`regions` may be string vector in the form "chr:start-end", a GRanges object or a data frame with "chr", "start", and "end" columns. The `feature` column of the output will contain a "chr:start-end" identifier for each summarized region. To use other identifiers, like a gene name for a region instead of the coordinates, set the names of the vector or GRanges to those identifiers. These names will be used instead of the genomic region string to describe each feature in the output dataframe. If `regions` is a data frame make an additional column with the identifiers and pass that column name to `feature_col`. See examples.

Examples

```
# also see examples from ?summarize_regions

bedfiles <- system.file("extdata", package = "iscream") |>
  list.files(pattern = "[a|b|c|d].bed.gz$", full.names = TRUE)

# make a vector of regions
regions <- c("chr1:1-6", "chr1:7-10", "chr1:11-14")
summarize_meth_regions(bedfiles, regions)
names(regions) <- c("A", "B", "C")
summarize_meth_regions(bedfiles, regions, fun = c("mean", "stddev"), mval = FALSE)
summarize_meth_regions(bedfiles, regions, fun = "sum")
```

summarize_regions *Summarize information over genomic regions from any BED file*

Description

Run summarizing functions on BED file records across genomic regions. Parallelized across files using threads from the "iscream.threads" option.

Usage

```
summarize_regions(
  bedfiles,
  regions,
  columns,
  col_names = NULL,
  fun = "all",
  feature_col = NULL,
  set_region_rownames = FALSE,
  nthreads = NULL
)
```

Arguments

bedfiles	A vector of BED file paths
regions	A vector, data frame or GenomicRanges of genomic regions. See details.
columns	A vector of indices of the numeric columns to be summarized
col_names	A vector of names to use for columns in the output
fun	Function(s) to apply over the region. See details.
feature_col	Column name of the input regions data frame containing a name for each genomic region. Set only if the using a data frame as the input regions format. See details.
set_region_rownames	Use the region strings as the returned data frame's rownames. Can be useful if you have a named regions and want both the regions strings rownames and the feature names. See details.
nthreads	Set number of threads to use overriding the "iscream.threads" option. See ?set_threads for more information.

Value

A data.frame

Supported functions

- Sum: "sum"
- Mean: "mean"
- Median: "median"
- Standard deviation: "stddev"
- Variance: "variance"
- Minimum: "min"
- Maximum: "max"
- Range: "range"
- No. of records in the region: "count"

The summarizing computations are backed by the Armadillo library. See https://arma.sourceforge.net/docs.html#stats_fns for futher details on the supported functions

Using feature identifiers

regions may be string vector in the form "chr:start-end", a GRanges object or a data frame with "chr", "start", and "end" columns. The feature column of the output will contain a "chr:start-end" identifier for each summarized region. To use other identifiers, like a gene name for a region instead of the coordinates, set the names of the vector or GRanges to those identifiers. These names will be used instead of the genomic region string to describe each feature in the output dataframe. If regions is a data frame make an additional column with the identifiers and pass that column name to feature_col. See examples.

Examples

```
bedfiles <- system.file("extdata", package = "iscream") |>
  list.files(pattern = "[a|b|c|d].bed.gz$", full.names = TRUE)
# examine the bedfiles
colnames <- c("chr", "start", "end", "beta", "coverage")
lapply(bedfiles, function(i) knitr::kable(read.table(i, col.names = colnames)))

# make a vector of regions
regions <- c("chr1:1-6", "chr1:7-10", "chr1:11-14")
summarize_regions(bedfiles, regions, columns = c(4, 5), col_names = c("beta", "cov"))

# select functions
summarize_regions(
  bedfiles,
  regions,
  fun = c("mean", "stddev"),
  columns = c(4, 5),
  col_names = c("beta", "cov")
)

# add names to the regions
names(regions) <- c("A", "B", "C")
summarize_regions(
  bedfiles,
```

```
  regions,
  fun = "sum",
  columns = 5,
  col_names = "coverage"
)

# using `feature_col`
library(data.table)

# convert string vector to a data.table
regions_df <- data.table::as.data.table(regions) |>
_[, tstrsplit(regions, ":|-", fixed = FALSE, names = c("chr", "start", "end"))] |>
_[, start := as.integer(start)] |>
_[, feature := LETTERS[.I]][]
regions_df

summarize_regions(
  bedfiles,
  regions_df,
  fun = "sum",
  columns = 5,
  col_names = "coverage",
  feature_col = "feature"
)
```

---

tabix *Query records from tabixed BED files*

Description

Query records from tabixed BED files

Usage

```
tabix(bedfiles, regions, aligner = NULL, col.names = NULL, nthreads = NULL)

tabix_gr(
  bedfiles,
  regions,
  aligner = NULL,
  col.names = NULL,
  zero_based = TRUE,
  nthreads = NULL
)

tabix_raw(bedfiles, regions, nthreads = NULL)
```

Arguments

bedfiles	The BED files to be queried
regions	A vector, data frame or GenomicRanges of genomic regions. See details.
aligner	The aligner used to produce the BED files - one of "biscuit", "bismark", "bsbolt". Will set the result data.table's column names based on this argument.
col.names	A vector of column names for the data columns of the result.table, not including "chr", "start", and "end". Set if your BED file is not from the supported aligners or is a general BED file.
nthreads	Set number of threads to use overriding the "iscream.threads" option. See ?set_threads for more information.
zero_based	Whether the input BED file has a zero-based start column - used when coverting the result data frame to GenomicRanges.

Details

Query method:

'*iscream* has two methods to query records from BED files:

- the *tabix* shell executable: fast since its output can be redirected to a file (which data.table::fread() can then read) instead of having to allocate memory and store it during the query
- *iscream's* tabix implementation, based on the *tabix* executable using *htslib*, but slower on large queries since it stores the records as they are found instead of writing to a file. However it's able to store each region's records independently instead of in a single file and is used in make_mat(), make_mat_bsseq(), and summarize_regions().

When *iscream* is attached, it checks that the *tabix* executable is available with Sys.which() and, if available, sets options("tabix.method" = "shell"). tabix() then uses the *tabix* executable to make queries, except for tabix_raw(). If *tabix* is not found, *iscream* uses its tabix implementation. To use only *iscream's* tabix implementation, set options("tabix.method" = "htslib").

Input region formats:

The input regions format may be string vector in the form "chr:start-end", a dataframe with "chr", "start" and "end" columns or a GRanges object. Input regions must be 1-based. When using "htslib" as the query method, if the input GRanges object of regions contains any single locus regions where the start and end positions are the same, iscream will notify that such regions were found and fixed as chr:start format strings are invalid for the htslib API (see ?get_granges_string).

Value

- tabix(): A data frame
- tabix_gr(): A GRanges object for single files and GRangesList for multiple files. When making GRanges, the 0-based records from BED-files will be converted to 1-based with GenomicRanges::makeGRangesFromDataFrame(). Bismark's coverage files will not be converted as they are already 1-based and the ranges slot will be only one position.
- tabix_raw(): A named list of raw strings from the regions in the style of Rsamtools::scanTabix

Examples

```
bedfiles <- system.file("extdata", package = "iscream") |>
  list.files(pattern = "[a|b|c|d].bed.gz$", full.names = TRUE)
regions <- c("chr1:1-6", "chr1:7-10", "chr1:11-14")
tabix(bedfiles, regions, col.names = c("beta", "coverage"))
if (require("GenomicRanges", quietly = TRUE)) {
  tabix_gr(bedfiles, regions, col.names = c("beta", "coverage"))
}
tabix_raw(bedfiles, regions)
```

---

validate_log_level *Validate provided logging level*

Description

Only "info" and "debug" are currently supported, with "debug" only supported when using 1 thread

Usage

```
validate_log_level(level = get_log_level(), n_threads)
```

Arguments

level	The logging level to validate
n_threads	The number of threads that the next iscream function call will use

Value

None; sets the log level to the provide level

Examples

```
set_log_level("info")
```

---

verify_aligner_or_stop *Validate provided aligner*

Description

Only "biscuit", "bismark", and "bsbolt" are currently supported

Usage

```
verify_aligner_or_stop(aligner)
```

Arguments

aligner	The input alinger

Value

true; quits if the input is not among supported_aligners

---

verify_files_or_stop *Verify that BED files are tabixed*

Description

Verify that BED files are tabixed

Usage

```
verify_files_or_stop(bedfiles, verify_tabix = TRUE)
```

Arguments

bedfiles	A vector of BED file paths
verify_tabix	Whether to verify the presence of tabix files

Value

TRUE if all input BED files have an associated tabix index file. FALSE if not

---

verify_filetype *Verify that the input BED files are of the type specified by the input aligner*

Description

Verify that the input BED files are of the type specified by the input aligner

Usage

```
verify_filetype(bedfiles, aligner, stop_on_error = FALSE)
```

Arguments

bedfiles	A vector of BED file paths
aligner	The aligner chosen
stop_on_error	Whether to warn or stop on aligner-filename mismatch

Value

TRUE if all input BED files have an associated tabix index file. FALSE if not

Index