Package: disprose (via r-universe)

September 14, 2024

Type Package

Title Discriminating Probes Selection

Version 0.1.6

Maintainer Elena Filatova <filatova@nniiem.ru>

Description Set of tools for molecular probes selection and design of a microarray, e.g. the assessment of physical and chemical properties, blast performance, selection according to sensitivity and selectivity. Methods used in package are described in: Lorenz R., Stephan H.B., Höner zu Siederdissen C. et al. (2011) <doi:10.1186/1748-7188-6-26>; Camacho C., Coulouris G., Avagyan V. et al. (2009) <doi:10.1186/1471-2105-10-421>.

License GPL-3

Encoding UTF-8

Language en-us

LazyData true

Suggests BBmisc, biomartr, curl, DBI, dplyr, rentrez, reutils, RSQLite, stats, stringr, seqinr, TmCalculator, utils, XML

Depends R (>= 3.5.0)

RoxygenNote 7.1.2

NeedsCompilation no

Author Elena Filatova [aut, cre] (<https://orcid.org/0000-0002-6683-7191>), Oleg Utkin [ctb] (<https://orcid.org/0000-0002-7571-525X>), Blokhina Scientific Research Institute of Epidemiology and Microbiology of Nizhny Novgorod, Russia [fnd]

Repository CRAN

Date/Publication 2022-03-18 23:30:02 UTC

Contents

`add_adapters` *Add adapters to probes*

Description

Add set of adapters to oligonucleotide probes

Usage

```
add_adapters(
  probe.id.var,
  probe.var,
  ad.len,
  ad.nucl = "t",
  end = c(3, 5),
```

```
  mc.cores = 1,
  digits = 4,
  return = "dataframe",
  data,
  data.probe.id.var,
  count.mfe = FALSE,
  RNAfold.path,
  temperature = 40,
  trim.mfe = FALSE,
  MFEmin = 0,
  MFE.files.dir = NULL,
  delete.MFE.files = FALSE,
  verbose = TRUE
)
```

Arguments

`probe.id.var`	vector of probes' identification numbers
`probe.var`	character; character; vector of nucleotide probes
`ad.len`	integer; vector of adapter length
`ad.nucl`	character; vector of adapter nucleotides
`end`	integer; probe's end for adapter attachment. Possible values are 3 and 5.
`mc.cores`	integer; number of processors for parallel computation (not supported on Windows)
`digits`	integer; number of decimal places to round the result (MFE)
`return`	character; returned object; possible values are: `"vector"` (vector of nucleotide probes with added adapters), `"dataframe"` (data frame with probes, adapters and their characteristics), `"add"` (user's data frame with added data of probes, adapters and their characteristics)
`data`, `data.probe.id.var`	user's data frame and it's variable with probes identification numbers (used if `return = "add"`)
`count.mfe`	logical; count minimum folding energy for probes with adapters
`RNAfold.path`, `temperature`, `trim.mfe`, `MFEmin`, `MFE.files.dir`, `delete.MFE.files`	used if `count.mfe = TRUE`; see count_MFE[disprose] for details
`verbose`	logical; show messages

Details

`ad.len` parameter indicates number of `ad.nucl` repeats. For example, with `ad.len =5` for `ad.nucl = "t"` adapter will be `"ttttt"` and for `ad.nucl = "ac"` adapter will be `"acacacacac"`.

`ad.len`, `ad.nucl` and `end` might be vectors of any length. All possible variants of adapters will be added to probes and tested.

For MFE counting ViennaRNA Package (UNIX or Windows) must be installed. see count_MFE[disprose] for details

Value

Vector of nucleotide probes with added adapters, or data frame with probes, adapters and their characteristics, or user's data frame with added data of probes, adapters and their characteristics.

Author(s)

Elena N. Filatova

Examples

```
probes <- data.frame (ids = 1:3,  probes = c ("acacacacacaca", "aaaaaggggtttttccccc",
                                              "atgcgctagctcagc"))
ad.data <- add_adapters(probe.var = probes$probes, probe.id.var = probes$ids,
                        ad.len = c(5, 8), ad.nucl = c("t", "dt"), end = c(3, 5),
                        count.mfe = FALSE, mc.cores = 1, digits = 4,
                      return = "dataframe", data = probes, data.probe.id.var = probes$ids)
```

`ann.data` *Chlamydia pneumoniae genome annotation.*

Description

A dataset containing Chlamydia pneumoniae TW-183 (complete sequence, NC_005043.1.) genome annotation

Usage

```
ann.data
```

Format

A data frame with 2218 rows and 9 variables:

seqid sequence identification number

source source database name

type type of annotated region

start region's start position

end region's end position

score score

strand strand

phase phase

attribute region description

Source

https://www.ncbi.nlm.nih.gov/

annotate_probes *Annotate probes*

Description

Get genome annotation for oligonucleotide sequence

Usage

```
annotate_probes(
  source = "data.frame",
  ann.data = NULL,
  gff.path = NULL,
  org.name,
  db = "refseq",
  refs = TRUE,
  probe.id.var,
  probe.start.var,
  probe.stop.var,
  file.annot = NULL,
  save.format = "txt",
  sep = ";",
  return = "add.resume",
  priority = c("CDS", "gene", "region"),
  data,
  data.probe.id.var,
  delete.downloads = FALSE,
  verbose = TRUE
)
```

Arguments

source	character; genome annotation source. Possible values are: "data.frame" (from data frame), "giff" (from GIFF file), "load" (download from NCBI with getGFF function)
ann.data	genome annotation data frame
gff.path	character; .gff file name and path
org.name	character; the scientific name of the organism of interest
db	character; database from which the genome shall be retrieved; possible values are "refseq", "genbank", "ensembl"
refs	logical; download genome if it isn't marked in the database as either a reference or a representative genome
probe.id.var	vector of probes' identification numbers
probe.start.var, probe.stop.var	integer; vector of probes' start and end coordinates

file.annot	character; resulting annotation file name and path
save.format	character; format of resulting annotation file; possible values are "txt", "csv"
sep	character; field separator string
return	character; returned object; possible values are: "annotation" (annotation data frame), "resume" (annotation attributes only), "add.resume" (user's data frame with added annotation attributes)
priority	character; vector of sequence ontology types that should be returned in resume in the first place
data, data.probe.id.var	users data frame and probes' identification variable in it (used if return = "add.resume")
delete.downloads	logical; delete files that were downloaded from NCBI
verbose	logical; show messages

Details

This function uses boimartr genome annotation retrieval instruments. See getGFF for details. If retrieval is not available, GFF file may be used.

This function creates annotation ".txt" or ".csv" file. By default file is created in working directory. Optionally function returns annotation resume, i.e. annotation attribute for specified sequence ontology (SO). Priorities of SOs are set by user in priopity parameter. For example, if priopity = c("CDS", "gene", "region"), the function returns resume for "CDS" SO, if there are none - for "gene" CO etc. If there are several attributes meet priority, the first annotation attribute is returned. If none of priority COs found, the first annotation attribute is returned.

Number of found annotations are indicated in returned data ("ann.n" column).

Value

Annotation data frame, or annotation attributes, or user's data frame with added annotation attributes. Also annotation file is created.

Author(s)

Elena N. Filatova

Examples

```
path<-tempdir()
dir.create(path) # create temporal directory
data(ann.data) # load genome annotation data frame
annotation<-annotate_probes(source = "data.frame", ann.data = ann.data,
                probe.id.var = 1:5,
                probe.start.var = c (1, 100, 200, 300, 400),
                probe.stop.var = c (99, 199, 299, 399, 499),
                file.annot = paste0(path, "/annotation.txt"), save.format = "txt",
                return = "resume")
file.remove(paste0(path, "/annotation.txt")) # delete files
unlink(path, recursive = TRUE)
```

`blast.fill` *Local BLAST results with added content.*

Description

Result of BLAST of 5 probes against local database of target nucleotide sequences of Chlamydia pneumoniae. Local BLAST was performed with blast_local () function. Subjects' Genbank Identifiers are added with fill_blast_result () function.

Usage

```
blast.fill
```

Format

A data frame with 72 rows and 19 variables:

probe probe sequence

probe.length probe sequence's length

Qid query identification number

Qstart query start position

Qend query end position

Rgi subject GenInfo Identifier number

Racc subject NCBI accession number

Rtitle subject title

Rtaxid subject taxon identificator

Rstart subject start position

Rend subject end position

alig.length length of alignment

mismatch amount of mismatches

gaps amount of gaps

ident.number amount of identical positions

score alignment score

bitscore alignment bitscore

Evalue alignment e-value

Qcover query coverage, %

`blast.raw` *Local BLAST results.*

Description

Result of BLAST of 5 probes against local database of target nucleotide sequences of Chlamydia pneumoniae. Local BLAST was performed with blast_local () function.

Usage

```
blast.raw
```

Format

A data frame with 72 rows and 19 variables:

probe probe sequence

probe.length probe sequence's length

Qid query identification number

Qstart query start position

Qend query end position

Rgi subject GenInfo Identifier number

Racc subject NCBI accession number

Rtitle subject title

Rtaxid subject taxon identificator

Rstart subject start position

Rend subject end position

alig.length length of alignment

mismatch amount of mismatches

gaps amount of gaps

ident.number amount of identical positions

score alignment score

bitscore alignment bitscore

Evalue alignment e-value

Qcover query coverage, %

`blast_local` *Local BLAST*

Description

Perform nucleotide BLAST with local database

Usage

```
blast_local(
  probe.var,
  probe.id.var = NULL,
  fasta.way = NULL,
  blastn.way = NULL,
  db.way = NULL,
  out.way = NULL,
  mc.cores = 1,
  add.query.info = FALSE,
  temp.db = NULL,
  delete.files = FALSE,
  eval = 1000,
  ws = 7,
  reward = 1,
  penalty = -3,
  gapopen = 5,
  gapextend = 2,
  maxtargetseqs = 500,
  verbose = TRUE
)
```

Arguments

`probe.var`	character; query - vector of nucleotide sequences
`probe.id.var`	vector of identification numbers for query sequences
`fasta.way`	character; name and path to FASTA file
`blastn.way`	character; name and path to blastn executable file
`db.way`	character; name and path to local BLAST database
`out.way`	character; name and path to blastn output file
`mc.cores`	integer; number of processors for parallel computation (not supported on Windows)
`add.query.info`	logical; add query nucleotide sequence and its length to result
`temp.db`	character; temporal SQLite database name and path
`delete.files`	logical; delete created FASTA and output files
`eval`	integer; expect value for saving hits

ws	integer; length of initial exact match
reward	integer; reward for a nucleotide match
penalty	integer; penalty for a nucleotide mismatch
gapopen	integer; cost to open a gap
gapextend	integer; cost to extend a gap
maxtargetseqs	integer; number of aligned sequences to keep
verbose	logical; show messages

Details

For this function BLAST+ executables (blastn) must be installed and local nucleotide database must be created.

While working, the function creates blastn input FASTA file and output file. If files exist already, they will be overwritten. Those files could be deleted by `delete.files = TRUE` parameter.

If no `probe.id.var` is provided, query sequences are numbered in order, starting with 1.

Query cover is query coverage per HSP (as a percentage)

If `add.query.info = TRUE` function saves data in temporal SQLite database. Function will stop if same database already exists, so deleting temporal database (by setting `delete.files = TRUE`) is highly recommended.

"no lines available in input" error is returned when there are no BLAST results matching the specified parameters. Adjust BLAST parameters.

Value

Data frame with BLAST alignments: query sequence id, start and end of alignment in query, subject GI, accession, title and taxon id, start and end of alignment in subject, length of alignment, number of mismatches and gaps, number of identical matches, raw score, bit score, expect value and query cover. If `add.result.info = TRUE`, query sequence and its length are also added to data frame.

Author(s)

Elena N. Filatova

References

Camacho C., Coulouris G., Avagyan V. et al. (2009). BLAST+: architecture and applications. BMC Bioinformatics 10, 421. https://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-10-421.

Examples

```
## Not run:
# This function is using BLAST applications. BLAST+ should be installed.
# Local nucleotide database should be created
# Local database of target sequences of Chlamydia pneumoniae was created
# in temporal directory previously (see make_blast_DB () function)
path <- tempdir()
```

```
dir.create (path)
#set probes for local BLAST
probes <- c ("catctctatttcggtagcagctcc", "aaagtcatagaaaagcctgtagtcgc",
             "ccttcttctcgaactctgaagtacact", "aaaaaaaaaaaaaaaaa", "acacacacacacaac")
blast.raw <- blast_local(probe.var = probes, probe.id.var = NULL,
                         fasta.way = paste0 (path, "/blast.fasta"),
                         blastn.way = "D:/Blast/blast-2.11.0+/bin/blastn.exe",
                         db.way = paste0 (path, "/DB"),
                         out.way = paste0 (path, "/blast.out"),
                    mc.cores=1, add.query.info = TRUE, temp.db = paste0 (path, "/temp.db"),
                         delete.files = TRUE, eval = 1, maxtargetseqs = 200)

## End(Not run)
```

---

count_PhCh *Calculate physical and chemical properties*

Description

Calculates GC-content, detects several nucleotides in a row, calculates minimum folding energy and melting temperature for oligonucleotide probes.

Usage

```
count_PhCh(
  probe.var,
  trim = FALSE,
  data,
  digits = 4,
  mc.cores = 1,
  MFE.files.dir = NULL,
  delete.MFE.files = FALSE,
  GCmin = 40,
  GCmax = 60,
  nucl.pattern = c("a", "t", "g", "c"),
  n.crit = 5,
  RNAfold.path,
  temperature = 40,
  MFEmin = -3,
  TD.params = NULL,
  TMmin = 55,
  TMmax = 60,
  verbose = TRUE,
  Na = 50,
  K = 0,
  Tris = 0,
  Mg = 0,
```

```
  dNTPs = 0
)

count_GC(
  probe.var,
  trim.gc = FALSE,
  GCmin = 40,
  GCmax = 60,
  mc.cores = 1,
  add.to.data = FALSE,
  data,
  digits = 4
)

count_REP(
  probe.var,
  trim.rep = FALSE,
  nucl.pattern = c("a", "t", "g", "c"),
  n.crit = 5,
  mc.cores = 1,
  add.to.data = FALSE,
  data
)

count_MFE(
  probe.var,
  RNAfold.path,
  temperature = 40,
  trim.mfe = FALSE,
  MFEmin = -3,
  add.to.data = FALSE,
  data,
  MFE.files.dir = NULL,
  delete.MFE.files = FALSE,
  mc.cores = 1,
  digits = 4,
  verbose = TRUE
)

count_TM(
  probe.var,
  TD.params = NULL,
  trim.tm = FALSE,
  TMmin = 55,
  TMmax = 60,
  add.to.data = FALSE,
  data,
  digits = 4,
```

```
  mc.cores = 1,
  verbose = TRUE,
  Na = 50,
  K = 0,
  Tris = 0,
  Mg = 0,
  dNTPs = 0
)
```

Arguments

probe.var	character; vector of nucleotide probes
trim, trim.gc, trim.rep, trim.mfe, trim.tm	logical; whether to select results that meet the criterion
digits	integer; number of decimal places to round the result
mc.cores	integer; number of processors for parallel computation (not supported on Windows)
MFE.files.dir	character; directory for RNAfold input and output files
delete.MFE.files	logical; delete RNAfold input and output files
GCmin, GCmax	numeric; minimum and maximum value of GC-content (percent, used if trim = TRUE)
nucl.pattern	character; vector of nucleotide pattern
n.crit	integer; minimal amount of nucleotide pattern's repeats in a row to detect
RNAfold.path	character; name and path to RNAfold executable file
temperature	numeric; folding design temperature
MFEmin	numeric; maximum value of folding energy (used if trim = TRUE)
TD.params	character; vector of length 4, contains designation for four tables with thermodynamic values (nn_table - thermodynamic NN values, tmm_table - thermodynamic values for terminal mismatches, imm_table - thermodynamic values for internal mismatches, de_table - thermodynamic values for dangling ends). See Tm_NN for details.
TMmin, TMmax	numeric; minimum and maximum value of melting temperature (used if trim = TRUE)
verbose	logical; show messages
Na	numeric; millimolar concentration of Na, default is 50 (used for count_TM function)
K	numeric; millimolar concentration of K, default is 0 (used for count_TM function)
Tris	numeric; millimolar concentration of Tris, default is 0 (used for count_TM function)
Mg	numeric; millimolar concentration of Mg, default is 0 (used for count_TM function)

dNTPs numeric; millimolar concentration of dNTPs, default is 0 (used for count_TM function)

add.to.data, data
logical; add result vector to specified data frame (used unconditionally if trim = TRUE)

Details

GC-content trimming selects results that are between GCmin and GCmax (inclusive). Nucleotides' amount trimming deletes probes that contain n.crit or more of same nucleotides (pattern) in a row. Minimum folding energy trimming selects results that are equal or more than MFEmin. Melting temperature trimming selects results that are between TMmin and TMmax (inclusive).

This function is using ViennaRNA service to count minimum folding energy. ViennaRNA Package (UNIX or Windows) must be installed. While counting MFE, working directory is set to MFE.files.dir and input and output files for ViennaRNA ("seq_in" and "seq_out") are created in the working directory.Afterwards the working directory is changed back to user's setting. If no MFE.files.dir exists it is created and is not deleted even if delete.MFE.files = TRUE.

Melting temperature is counted with Tm_NN function. Indication of thermodynamic values must be provided. By default they are: nn_table = "DNA_NN4", tmm_table = "DNA_TMM1", imm_table = "DNA_IMM1", de_table = "DNA_DE1".

Value

If trim = FALSE, count_PhCh function returns data frame with GC-count (GC.percent), nucleotide repeats (repeats, TRUE/FALSE), minimum folding energy (MFE) and melting temperature (TM) columns. If trim = TRUE, count_PhCh function returns provided data frame with attached four columns and rows selected according to values GCmin, GCmax, n.crit, MFEmin, TMmin, TMmax.

If trim.gc= FALSE, count_GC function returns GC.percent vector or data with attached GC.percent column (when add.to.data = TRUE). If trim.gc = TRUE, count_GC function returns provided data frame with attached GC.percent column and rows selected according to GCmin, GCmax values.

If trim.rep = FALSE, count_REP function returns repeats vector (logical; TRUE/FALSE - there are/there are no nucleotide repeats) or data with attached repeats column (when add.to.data = TRUE). If trim.rep = TRUE, count_REP function returns provided data frame with attached repeats column and rows selected according to n.crit value.

If trim.mfe = FALSE, count_MFE function returns MFE vector or data with attached MFE column (when add.to.data = TRUE). If trim.mfe = TRUE, count_MFE function returns provided data frame with attached MFE column and rows selected according to MFEmin value.

If trim.tm = FALSE, count_TM function returns TM vector or data with attached TM column (when add.to.data = TRUE). If trim.tm = TRUE, count_TM function returns provided data frame with attached TM column and rows selected according to TMmin, TMmax values.

Functions

- count_PhCh: Calculates GC.percent, detects several nucleotides in a row, calculates minimum folding energy and melting temperature
- count_GC: Calculates GC-content (percent)
- count_REP: Detects several nucleotides in a row

- count_MFE: Calculates minimum folding energy
- count_TM: Calculates melting temperature

Author(s)

Elena N. Filatova

References

Lorenz R., Stephan H.B., Höner zu Siederdissen C. et al. (2011). ViennaRNA Package 2.0. Algorithms for Molecular Biology, 6, 1. https://almob.biomedcentral.com/articles/10.1186/1748-7188-6-26.

Examples

```
probes <- data.frame (ids = 1:3,  probes = c ("acacacacacaca", "aaaaaggggtttttccccc",
                                              "atgcgctagctcagc"))
probes <- count_GC (probe.var = probes$probes, trim.gc = FALSE, GCmin = 40, GCmax = 60,
                    add.to.data = TRUE, data = probes)

probes <- count_REP (probe.var = probes$probes, trim.rep = FALSE, n.crit = 5,
                     add.to.data = TRUE, data = probes)
## Not run:
# This function is using ViennaRNA service. ViennaRNA Package must be installed.
MFE.files.dir <- tempdir()
probes <- count_MFE (probe.var = probes$probes, RNAfold.path = "D:/Vienna/RNAfold.exe",
                     temperature = 40, trim.mfe = FALSE, MFEmin = 0,
                     MFE.files.dir = MFE.files.dir, delete.MFE.files = TRUE,
                     add.to.data = TRUE, data = probes, mc.cores = 1)
unlink (MFE.files.dir, recursive = TRUE)

## End(Not run)
probes <- count_TM (probe.var = probes$probes, TD.params = NULL, trim.tm = FALSE,
                    TMmin = 55, TMmax = 60, add.to.data = TRUE, data = probes,
                    digits = 4, mc.cores = 1)
# All in one command
## Not run:
# This function is using ViennaRNA service. ViennaRNA Package must be installed.
MFE.files.dir <- tempdir()
probes2 <- count_PhCh (probe.var = probes$probes, trim = FALSE,
                       nucl.pattern = c ("a", "t", "g", "c"), n.crit = 5,
                       MFE.files.dir = MFE.files.dir, delete.MFE.files = TRUE,
                       RNAfold.path = "D:/Vienna/RNAfold.exe", temperature = 40,
                       TD.params = NULL, digits = 3, mc.cores = 1,
                       data = probes)
unlink (MFE.files.dir, recursive = TRUE)

## End(Not run)
```

cut_probes *Cut probes*

Description

Generate probes from nucleotide reference sequences

Usage

```
cut_probes(
  ref.seq.from.file = FALSE,
  ref.seq.id,
  ref.seq.db,
  fasta.file = NULL,
  delete.fasta = FALSE,
  start = 1,
  stop = NULL,
  start.correction = FALSE,
  size = 24:32,
  delete.incomplete = FALSE,
  delete.identical = FALSE,
  give.probes.id = FALSE,
  mc.cores = 1,
  verbose = TRUE
)
```

Arguments

ref.seq.from.file	logical; read reference sequences from file (TRUE) or download them from NCBI data base (FALSE).
ref.seq.id	identification number of reference nucleotide sequences. Only used when ref.seq.from.file = FALSE. GenBank accession numbers, GenInfo identifiers (GI) or Entrez unique identifiers (UID) may be used.
ref.seq.db	character; NCBI database for search. See entrez_dbs for possible values. Only used when ref.seq.from.file = FALSE.
fasta.file	character; FASTA file name and path, only used when ref.seq.from.file = TRUE.
delete.fasta	logical; delete FASTA file.
start, stop	integer; number of first and last nucleotide of the reference sequence's segment that should be cut into probes. All sequence is used by default.
start.correction	logical; count probes' start and stop nucleotides relatively to the specified segment (FALSE) or to the whole sequence (TRUE). Only used if start>1.
size	integer; vector of probe size

`delete.incomplete`
: logical; remove probes that contain undeciphered nucleotides

`delete.identical`
: logical; remove identical (duplicated) probes

`give.probes.id`
: logical; add probes' identification numbers

`mc.cores`
: integer; number of processors for parallel computation (not supported on Windows)

`verbose`
: logical; show messages

Details

This function takes nucleotide sequences and cut them on segments (probes) of given size. Sequences might be downloaded from given FASTA file or from NCBI data bases. In the latter case, FASTA file is created. If desired, FASTA file can be deleted after.

Not all sequence must be cut on probes, you may define needed segment by `start` and `stop` parameters. Note that in this case probes' start and stop nucleotides would be counted relatively to the specified segment (`start.correction = FALSE`) or to the whole sequence (`start.correction = TRUE`).

Undeciphered nucleotides are the one that are indicated by "rywsmkhbvdn" symbols.

Probes' identification numbers are created by adding numeric indexes to reference sequence's identification number.

See cut_string, delete_duplicates_DF and make_ids for details.

Value

Data frame with probe id (optionally), sequence id, probe size, start and stop nucleotide, sequence.

Author(s)

Elena N. Filatova

Examples

```
path <- tempdir()
dir.create (path)
# download and save as FASTA "Chlamydia pneumoniae B21 contig00001,
# whole genome shotgun sequence" (GI = 737435910)
if (!requireNamespace("rentrez", quietly = TRUE)) {
stop("Package \"rentrez\" needed for this function to work. Please install it.", call. = FALSE)}
reference.string <- rentrez::entrez_fetch(db = "nucleotide", id = 737435910,
                                          rettype="fasta")
write( x= reference.string, file = paste0 (path, "/fasta"))
probes <- cut_probes (ref.seq.from.file = TRUE, fasta.file = paste0(path, "/fasta"),
                      delete.fasta = TRUE, start = 1000, stop = 1500,
                      start.correction = FALSE, size = c(400, 500),
                      delete.incomplete = FALSE,
                      delete.identical = FALSE, give.probes.id = TRUE, mc.cores = 1)
unlink (path, recursive = TRUE)
```

cut_string *Cut string into segments*

Description

Cuts character string into segments of given size.

Usage

```
cut_string(string, size)
```

Arguments

string	character string; vector of length 1
size	integral; vector of length of segments

Details

This function works with one string only. Segments are cut from start to end of a string. size might be a vector of any length, all possible variants will be cut.

Value

Data frame with segment size, start and end point, segment string.

Author(s)

Elena N. Filatova

Examples

```
cut_string (string = "aaatttttttccgc", size = 12:14)
```

delete_duplicates_DF *Delete rows with duplicated values*

Description

Delete data frame rows if they contain duplicated values.

Usage

```
delete_duplicates_DF(
  data,
  duplicated.var,
  exact = FALSE,
  stay = "first",
  choose.var,
  choose.stay.val,
  pattern,
  mc.cores = 1,
  verbose = TRUE
)
```

Arguments

`data`	data frame;
`duplicated.var`	variable that contains duplicated values
`exact`	logical; values are to be matched as is
`stay`	character; which row with duplicated values will stay; possible values are `"first"` (first of rows), `"choose"` (depending of the value of other variable) and `"none"` (rows with values that contain pattern will be removed)
`choose.var`, `choose.stay.val`	vector of additional variable to choose the preferred row and it's preferred value (used if `stay = "choose"`)
`pattern`	deleted pattern (used if `stay = "none"`)
`mc.cores`	integer; number of processors for parallel computation (not supported on Windows)
`verbose`	logical; show messages

Details

This function checks if there are repeated values in the data frame (in the `duplicated.var`). If repeated values are found, the first row with duplicated value stays, others are deleted (if `stay = "first"`). If `stay = "choose"` the first row with duplicated values and `choose.var = choose.stay.val` will stay. If there are no rows with `choose.var = choose.stay.val`, the first row will stay.

If `stay = "none"` all rows with values that contain pattern will be removed.

Value

Data frame without rows that contain duplicates in `duplicated.var`

Author(s)

Elena N. Filatova

Examples

```
data <- data.frame (N = c(1:5, 11:15), name = c(rep( "A",4), "AA", rep( "B",3), "BB", "C"),
                choose = c(rep(c("yes", "no"), 3), "yes", "yes", "no", "no"))
delete_duplicates_DF (data = data, duplicated.var = data$N, exact = TRUE, stay = "first")
delete_duplicates_DF (data = data, duplicated.var = data$N, exact = FALSE, stay = "first")
delete_duplicates_DF (data = data, duplicated.var = data$name, exact = TRUE, stay = "first")
delete_duplicates_DF (data = data, duplicated.var = data$name, exact = TRUE,
                    stay = "choose", choose.var = data$choose, choose.stay.val = "yes")
delete_duplicates_DF (data = data, duplicated.var = data$name, exact = FALSE, stay = "first")
delete_duplicates_DF (data = data, duplicated.var = data$name, exact = FALSE,
                    stay = "choose", choose.var = data$choose, choose.stay.val = "yes")
delete_duplicates_DF (data =data, duplicated.var = data$name, stay = "none",
                    pattern = c("A", "B"), exact = TRUE)
delete_duplicates_DF (data =data, duplicated.var = data$name, stay = "none",
                    pattern = c("A", "B"), exact = FALSE)
```

---

fill_blast_result *Complement BLAST result*

Description

Provides subjects' GenInfo Identifiers if BLAST alignment result does not contain one.

Usage

```
fill_blast_results(
  blast.result,
  AcNum.column.name = "Racc",
  GI.column.name = "Rgi",
  delete.version = FALSE,
  version.sep = ".",
  add.gi = "DB",
  add.gi.df,
  temp.db = NULL,
  delete.temp = FALSE,
  add.gi.db = NULL,
  add.gi.table = NULL,
  add.gi.ac.column.name = "AC",
  add.gi.gi.column.name = "GI",
  mc.cores = 1,
  verbose = TRUE
)

delete_AcNum_version(ac.num.var, version.sep = ".", mc.cores = 1)
```

Arguments

blast.result	data frame; BLAST alignment result
AcNum.column.name, GI.column.name	character; name of column with subject accession numbers and GenInfo Identifier numbers from BLAST result data frame
delete.version	logical; remove version suffix from subject accession number
version.sep	character; accession number and version suffix separator (a dot for NCBI accession numbers)
add.gi	character; table with linked accession and GI numbers is taken from SQLite database ("DB") or data frame ("DF")
add.gi.df	data frame with table (used if add.gi = "DF")
temp.db	character; temporal SQLite database name and path
delete.temp	logical; delete created temporal SQLite database
add.gi.db, add.gi.table, add.gi.ac.column.name, add.gi.gi.column.name	SQLite database name and path, table name and name of columns with accession and GI numbers (used if add.gi = "DB")
mc.cores	integer; number of processors for parallel computation (not supported on Windows)
verbose	logical; show messages
ac.num.var	vector of accession numbers

Details

BLAST alignment, performed with local database, may not contain subject GI information. Also subject accession may contain version suffix. This can make it difficult to analyze the results further. This function adds subject GI and removes subject accession version suffix.

To add GI GenInfo Identifiers table with them linked to accession numbers must be provided as data frame or SQLite database table. add.gi.df must be a data frame with column one - accession numbers, column two - GenInfo Identifier numbers. If add.gi = "DF" temporal SQLite database is created.

SQLite database table with accession and GI numbers should not contain duplicated rows. It is also highly recommended to index accession numbers' variable in database.

delete.version executes in the first step, so if you use this option accession numbers in add.gi table must not contain version suffix.

AcNum.column.name, GI.column.name, add.gi.ac.column.name and dd.gi.gi.column.name must be column names exactly as in data frame.

Value

blast.result data frame with added GI and deleted accession version suffix.

Functions

- fill_blast_results: Provides subjects' Genbank Identifiers if BALST alignment result does not contain one
- delete_AcNum_version: Remove accession version suffix

Author(s)

Elena N. Filatova

Examples

```
path <- tempdir()
dir.create (path)
# load raw blast results
data (blast.raw)
#load meta.target with result (targets' sequences) GI and Acc.nums
data (meta.target)
blast.fill <- fill_blast_results(blast.result = blast.raw, delete.version = TRUE,
                            add.gi = "DF", add.gi.df = meta.target[, c("GB_AcNum", "gi")],
                                 temp.db = paste0 (path, "/temp.db"), delete.temp = TRUE)
```

`get_GA_files` *Read GISAID sequence file*

Description

Get metadata and nucleotide sequence from GISAID files

Usage

```
get_GA_files(
  dir.path,
  return = "both",
  seq.return = "data.frame",
  fasta.file = NULL,
  verbose = TRUE
)
```

Arguments

`dir.path`	character; directory name and path
`return`	character; type of returned object; possible values are: `"info"` (sequence metadata), `"seq"` (nucleotide sequences), `"both"` (both of them).
`seq.return`	character; sequence returned object; possible values are "vector", "data.frame" and "fasta"
`fasta.file`	character; FASTA file name and path, only used if `return = "fasta"`
`verbose`	logical; show messages

Details

This function works with downloaded from GISAID "Input for the Augur pipeline" archives (with "metadata.tsv" and "sequences.fasta" files). Archives must be unzipped before usage. All extracted from GISAID archive files must be in one directory.

If `return = "seq"`, serial numbers are used as sequence identification numbers.

Metadata is transformed into data frame of the same format as get_seq_info function does. Sequences are transformed into data type of the same format as get_seq_for_DB function does.

Value

List of length two, where first is metadata and second is nucleotide sequence. If `return = "info"` or `return = "seq"` only first or second element is returned.

Author(s)

Elena N. Filatova

Examples

```
## Not run:
# First download some sequences' archives from GISAID (https://www.gisaid.org/)
# unzip them and put into "gisaidfiles" directory

res <- get_GA_files (dir.path = "gisaidfiles", return = "info")
res <- get_GA_files (dir.path = "gisaidfiles", return = "seq", seq.return = "data.frame")
res <- get_GA_files (dir.path = "gisaidfiles", return ="both", seq.return = "fasta")

## End(Not run)
```

---

get_GIs *Get GenInfo Identifier numbers*

Description

Retrieves NCBI sequence identifiers (GIs) for given organism name or taxon identifier.

Usage

```
get_GIs(
  org.name,
  db,
  n.start = 1,
  n.stop = NULL,
  step = 99999,
  return.vector = TRUE,
  check.result = FALSE,
```

```
  term = NULL,
  temp.dir = NULL,
  delete.temp = FALSE,
  verbose = TRUE
)

get_GIs_fix(
  gis.list,
  org.name,
  db,
  n.start = 1,
  n.stop = NULL,
  step = 99999,
  term = NULL,
  temp.dir = NULL,
  delete.temp = FALSE,
  verbose = TRUE
)
```

Arguments

org.name	character; scientific name or taxon identifier (written as "txid0000") of the organism/taxon.
db	character; NCBI database for search. See entrez_dbs() for possible values.
n.start	integer; download starting value. Default is 1.
n.stop	integer; download finishing value. Default is NULL, which provides retrieval of all available GIs.
step	integer; download increment value.
return.vector	logical; whether to return GI numbers as character vector (another variant is list of vectors).
check.result	logical; check if download was done correctly.
term	character; search query.
temp.dir	character; name and path of directory for downloaded temporary files (only for "Windows" OS)
delete.temp	logical; delete downloaded files (only for "Windows" OS, does not delete directory).
verbose	logical; show messages
gis.list	list of previously downloaded GIs vectors.

Details

This function sends the query to NCBI database and returns sequence identifiers according to the query. By default the query is organism, so the function returns GI numbers for all sequences that are associated with the requested organism. For example, if `org.name = "Homo sapiens"` the function will download GI numbers for all sequences that answer the query "Homo sapiens[Organism]". For any other query use parameter `term`.

The function downloads GI numbers by piecemeal, by several pieces in one block. The size of the block is defined by parameter `step`. It is useful if by any reason the download was interrupted, so later it is possible to reload only the missing blocks without the need to reload the entire amount of data. By default, all available GI numbers are downloaded, but you may also choose start and finish notes by specifying the parameters `n.start` and `n.stop`. The numeration starts with 1, not 0. At the end the resulting list of blocks (list of character vectors) is unlisted into one character vector. You may prevent this by setting `return.vector = FALSE`. Also, regardless of `return.vector` settings, the list of blocks is returned if the download was somehow compromised.

If download was corrupted you may use `get_GIs_fix()` function to reload the missing block. The corrupted list of blocks should be set in `gis.list` parameter. You may also check and reload data when `get_GIs()` function is running by specifying `check.result = TRUE`.

The function checks for user's OS type. For Windows temporal files are created while downloading, so `temp.dir` and `delete.temp` parameters should be set. This helps to solve the "routines:SSL23_GET_SERVER_HELLO:tlsv1 alert protocol version" problem by using `curl` instead of `RCurl`. However it slows down the function.If there is no `temp.dir` directory, it will be created and will not be removed (only temporal files will be deleted if `delete.temp = TRUE`).

In progress the functions turn off and on scientific notation.

Value

`get_GIs()` returns character vector of GI numbers. If `return.vector = FALSE` or there are missing data, list of character vectors is returned.

`get_GIs_fix()` returns list of character vectors.

Functions

- `get_GIs`: Retrieves NCBI sequence identifiers (GIs) for given organism name or taxon identifier.
- `get_GIs_fix`: Checks the downloads and tries to retrieve the compromised data.

Author(s)

Elena N. Filatova

Examples

```
gi.list<-get_GIs(org.name="txid9606", db="nucleotide",
                n.start=1, n.stop=3, step=1,
                return.vector = FALSE, check.result=TRUE,
                temp.dir = tempdir(),  delete.temp=TRUE)
```

`get_seq_for_DB` *Get nucleotide sequences from NCBI*

Description

Retrieves nucleotide sequences from NCBI for given identification numbers.

Usage

```
get_seq_for_DB(
  ids,
  db,
  check.result = FALSE,
  return = "data.frame",
  fasta.file = NULL,
  exclude.from.download = FALSE,
  exclude.var,
  exclude.pattern,
  exclude.fixed = TRUE,
  verbose = TRUE
)

get_seq_for_DB_fix(res.data, db, verbose = TRUE)
```

Arguments

`ids`	vector of NCBI sequences' identification numbers: GenBank accession numbers, GenInfo identifiers (GI) or Entrez unique identifiers (UID)
`db`	character; NCBI database for search. See entrez_dbs() for possible values
`check.result`	logical; check if download was done correctly
`return`	character; sequence returned object; possible values are "vector", "data.frame" and "fasta"
`fasta.file`	character; FASTA file name and path, only used if `return = "fasta"`
`exclude.from.download`	logical; ignore some sequences while downloading
`exclude.var`	vector that is used to define which sequences should be ignored, only used if `exclude.from.download = TRUE`.
`exclude.pattern`	value that matches to `exclude.var` and marks unwanted sequences, only used if `exclude.from.download = TRUE`
`exclude.fixed`	logical; match `exclude.pattern` as is, only used if `exclude.from.download = TRUE`.
`verbose`	logical; show messages
`res.data`	data.frame; data frame of nucleotide ids and previously downloaded sequences

Details

Master records (for example, in WGS-project) do not contain any nucleotide. They might be excluded from download with `exclude.from.download` parameters. However this has no affect and such ids do not have to be excluded when loading.

If writing FASTA to existing FASTA file, sequences are appended.

Value

If `return = "vector"` function returns vector of nucleotide sequences, `return = "data.frame"` - data frame with nucleotide ids and nucleotide sequences, `return = "fasta"` - writes FASTA file, no data returned.

Functions

- `get_seq_for_DB`: Retrieves NCBI nucleotide sequences for given identification numbers.
- `get_seq_for_DB_fix`: Checks the downloads and tries to retrieve the compromised data.

Author(s)

Elena N. Filatova

Examples

```
ids<-c(2134240466, 2134240465, 2134240464)
fasta.file<-tempfile()
get_seq_for_DB (ids = ids, db = "nucleotide", check.result = TRUE,
                return = "fasta", fasta.file = fasta.file, exclude.from.download=FALSE)
file.remove(fasta.file)
```

---

get_seq_info *Get NCBI sequence record*

Description

Retrieves information about sequences from NCBI records for given organism name or taxon identifier.

Usage

```
get_seq_info(
  org.name,
  db,
  n.start = 1,
  n.stop = NULL,
  step = 500,
  return.dataframe = FALSE,
```

```
  check.result = FALSE,
  term = NULL,
  verbose = TRUE
)

get_seq_info_fix(
  info.list,
  web.history = NULL,
  org.name = NULL,
  db,
  n.start = 1,
  n.stop = NULL,
  step = 500,
  term = NULL,
  verbose = TRUE
)

info_listtodata(info.list, unlist = TRUE, verbose = TRUE)
```

Arguments

org.name	character; scientific name or taxon identifier (written as "txid0000") of the organism/taxon.
db	character; NCBI database for search. See entrez_dbs() for possible values.
n.start	integer; download starting value. Default is 1.
n.stop	integer; download finishing value. Default is NULL, which provides retrieval of all available GIs.
step	integer; download increment value. Maximum is 500.
return.dataframe	integer; whether to return information as structured data frame (another variant is list of lists).
check.result	logical; check if download was done correctly.
term	character; search query.
verbose	logical; show messages
info.list	list of previously downloaded records.
web.history	previously saved web_history object for use in calls to the NCBI. New web.history is created if none is provided.
unlist	logical; unlist result before transforming (only recommended if step > 1).

Details

This function sends the query to NCBI database and returns sequence records according to the query. By default the query is organism, so the function returns data of all sequences that are associated with the requested organism. For example, if org.name = "Homo sapiens" the function will download data for all records that answer the query "Homo sapiens[Organism]". For any other query use parameter term.

The function downloads records by piecemeal, by several pieces in one block. The size of the block is defined by parameter `step`. It is useful if by any reason the download was interrupted, so later it is possible to reload only the missing blocks without the need to reload the entire amount of data. By default, all available records are downloaded, but you may also choose start and finish points by specifying the parameters `n.start` and `n.stop`. The numeration starts with 1, not 0. At the end the resulting list of blocks (list of lists if `step` > 1) is unlisted into one data frame that contains information about record GI, UID, caption, source database, organism, strain etc. You may prevent this by setting `return.dataframe` = `FALSE`. Also, regardless of `return.dataframe` settings, the list of blocks is returned if the download was somehow compromised. Optionally, you can turn the resulting list into data frame later using the function `info_listtodata()`. Note that in this case, if parameter `info.list` was inherited from `get_seq_info()` function, the result must be unlisted first (use `unlist` = `TRUE`).

If download was corrupted you may use `get_seq_info()` function to reload the missing block. The corrupted list of blocks should be set in `info.list` parameter. You may also check and reload data when `get_seq_infos()` function is running by specifying `check.result` = `TRUE`.

In progress the functions turn off and on scientific notation.

Value

`get_seq_info()` returns data frame that contains most of sequence information from NCBI records. If `return.dataframer` = `FALSE` or there are missing data, list of lists is returned. List contains full information from NCBI records.

`get_seq_info_fix()` returns list of lists.

`info_listtodata()` returns data frame.

Functions

- `get_seq_info`: Retrieves NCBI sequence records for given organism name or taxon identifier.
- `get_seq_info_fix`: Checks the downloads and tries to retrieve the compromised data.
- `info_listtodata`: Transforms downloaded list into data frame.

Author(s)

Elena N. Filatova

Examples

```
info.dataframe <- get_seq_info (org.name = "txid9606", db = "nucleotide", n.start = 1,
                                n.stop = 10, step = 5, return.dataframe = TRUE,
                                check.result = TRUE)
```

`make_blast_DB` *Builds local database for BLAST*

Description

Builds a BLAST database with local sequences using FASTA file.

Usage

```
make_blast_DB(
  makeblastdb.way,
  fasta.way,
  db.way,
  db.type = "nucl",
  db.title,
  delete.fasta = FALSE,
  verbose = TRUE
)
```

Arguments

`makeblastdb.way`	character; name and path to makeblastdb executable file
`fasta.way`	character; name and path to FASTA file
`db.way`	character; name and path to local BLAST database
`db.type`	character; type of BLAST database
`db.title`	character; BLAST data base title
`delete.fasta`	logical; delete FASTA file
`verbose`	logical; show messages

Details

This function is using BLAST applications. BLAST+ (UNIX or Windows) should be installed.

Value

The function creates local BLAST data base. No additional data is returned.

Author(s)

Elena N. Filatova

References

Camacho C., Coulouris G., Avagyan V. et al. (2009). BLAST+: architecture and applications. BMC Bioinformatics 10, 421. https://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-10-421.

Examples

```
## Not run:
# This function is using BLAST applications. BLAST+ should be installed.
# FASTA file with sequences for local data base should be downloaded first (see get_seq_for_DB ())
path <- tempdir()
dir.create (path)
# load metadata for target sequences of Chlamydia pneumoniae
(meta.target)
# load sequences, it will take about 3 minutes
get_seq_for_DB (ids = meta.target$gi, db = "nucleotide", check.result = TRUE,
              return="fasta", fasta.file = paste0 (path, "/seq.fasta"), verbose = TRUE)
# create local data base, it will take 0.235217 seconds
make_blast_DB (makeblastdb.way = "D:/Blast/blast-2.11.0+/bin/makeblastdb.exe",
              fasta.way = paste0 (path, "/seq.fasta"), db.title = "Cl_pneumoniae",
              db.way = paste0 (path, "/DB"), db.type = "nucl", delete.fasta = FALSE)
# delete FASTA file (also can set delete.fasta = TRUE)
file.remove (paste0 (path, "/seq.fasta"))

## End(Not run)
```

`make_ids` *Create unique identification values*

Description

Creates unique identification values by adding numbers to identical values.

Usage

```
make_ids(var, sep = "_")
```

Arguments

`var`	vector of values
`sep`	character; string to separate the terms

Details

This function takes vector with same values and adds numbers to create unique values.

Value

Character vector of `var` with attached numbers.

Author(s)

Elena N. Filatova

Examples

```
var<-c("one", "two", "three", "one", "two", "three", "one")
make_ids(var)
```

`meta.all` *Metadata of all available Chlamydia pneumoniae's sequences.*

Description

A dataset containing metadata of all Chlamydia pneumoniae's nucleotide sequences that were downloaded from NCBI Nucleotide database (November, 2021)

Usage

```
meta.all
```

Format

A data frame with 9062 rows and 21 variables:

uid sequence identification number

gi sequence identification number

GB_AcNum sequence identification number

createdate date of note's creation

updatedate date of last note's update

source_db database

organism organism name

title sequence title

strain strain

taxid taxon identificator

length sequence length

biomol biomolecule

moltype molecular type

genome genome type

complete sequence completness

geneticcode type of genetic and codon codes

strand strand

host host

country country

isolation_source isolation material

collection_date collection date

Source

https://www.ncbi.nlm.nih.gov/

`meta.target` *Metadata of target Chlamydia pneumoniae's sequences.*

Description

A dataset containing target nucleotide sequences of Chlamydia pneumoniae that were downloaded from NCBI Nucleotide database (November, 2021). Target sequences are chosen from all available sequences as targets for discriminating probes.

Usage

```
meta.target
```

Format

A data frame with 183 rows and 21 variables:

uid sequence identification number
gi sequence identification number
GB_AcNum sequence identification number
createdate date of note's creation
updatedate date of last note's update
source_db database
organism organism name
title sequence title
strain strain
taxid taxon identificator
length sequence length
biomol biomolecule
moltype molecular type
genome genome type
complete sequence completness
geneticcode type of genetic and codon codes
strand strand
host host
country country
isolation_source isolation material
collection_date collection date

Source

https://www.ncbi.nlm.nih.gov/

`normalize_DF` *Normalize variable*

Description

Normalize variable in a data frame

Usage

```
normalize_DF(
  data,
  var.name,
  method = "mean",
  norm.number,
  return = "add.end",
  digits = 2
)
```

Arguments

`data`	data frame with numeric variable that should be normalized
`var.name`	character; data frame column name with numeric variable that should be normalized
`method`	character; normalization method; possible values are: `"mean"` (normalize to mean), `"median"` (normalize to median), `"number"` (normalize to given number)
`norm.number`	numeric; a value to normalize to (if `method = "number"`)
`return`	character; return object; possible values are: `"vector"` (return a vector of normalized values), `"replace"` (replace `var.name` values with normalized values in data frame), `"add.near"` (add normalized values next to `var.name` values in data frame), `"add.end"` (add normalized values as the latter column in data frame)
`digits`	integer; number of decimal places to round the normalized value

Details

This function scales variable to a range of (0-1), where 1 get values that are the most close to mean, median or given number. See normalize for details.

`var.name` must be exact column name as in data frame.

Value

Vector or data frame with normalized values.

Author(s)

Elena N. Filatova

Examples

```
data <- data.frame (N = 1:5, temperature = c(37.5, 36.6, 41.2, 38.8, 36.7),
                name = c("Bob", "Kate", "Steve", "Sonya", "Mary"))
normalize_DF (data = data, var.name = "temperature", method = "mean", return = "vector")
normalize_DF (data = data, var.name = "temperature", method = "mean", return = "replace")
normalize_DF (data = data, var.name = "temperature", method = "mean", return = "add.near")
normalize_DF (data = data, var.name = "temperature", method = "number",
            norm.number = 36.6, return = "add.end")
```

---

rate_DF *Rate variables*

Description

Count data frame's row rate according to several variables

Usage

```
rate_DF(
  data,
  rate.var,
  weights,
  return = "add",
  as.percent = FALSE,
  percent.var,
  digits = 2
)
```

Arguments

data	data frame with rated variables
rate.var	character; vector of data frame column names with numeric variables of range (0-1) that should be used for rating
weights	numeric; vector of variables' weights (their sum must be 1)
return	character; return object; possible values are: "vector" (return a vector of rate values), "add" (add rated values as the latter column in data frame)
as.percent	logical; if some rated variables are percentages
percent.var	character; vector of data frame column names with rated variables that are percentages
digits	integer; number of decimal places to round the rate value

Details

This function counts rate as `rate = var1*weight1 + var2*weight2 + var3*weight3 +...` etc. All variables must be in range (0-1) and sum of weights must be 1. If you use percentages as rating variable, use `as.percent = TRUE`. Those variables would be divided by 100 before rating and then would be multiplicated by 100 after rating.

`rate.var` and `percent.var` must be exact column names as in data frame.

Value

Vector or data frame with rate values.

Author(s)

Elena N. Filatova

Examples

```
data <- data.frame (N = 1:5, percent = c(12, 15, 18, 20, 94), number = c(0.1, 0.5, 0.6, 0.8 ,0.9))
rate_DF (data = data, rate.var = c("percent", "number"), weights = c(0.4, 0.6), return = "add",
                              as.percent = TRUE, percent.var = "percent")
```

---

read_and_unite_files *Read and unite files*

Description

Read a bunch of table files and unite them in one data frame

Usage

```
read_and_unite_files(
  path,
  pattern,
  sep = ";",
  header = TRUE,
  add.file.id = FALSE,
  file.id = NULL,
  unique = FALSE
)
```

Arguments

`path`	character; directory path
`pattern`	character; file names pattern
`sep`	character; the field separator character

header	logical; files contain the names of the variables as its first line
add.file.id	logical; add file identification columns
file.id	data frame with file identification values
unique	logical; delete repeated rows

Details

All files must be tables of same type. All files must be in one directory.

File identification columns might be added. There might be any number of such columns. They are added at the beginning of result data frame. File identification values are set as `file.id` data frame, where each column contains possible identification values and column names are names of identificator. If no `file.id` provided file names are set by default.

Value

data frame with united files' content.

Author(s)

Elena N. Filatova

Examples

```
path <- tempdir()
dir.create(path)
t1<-paste0(path, "/table1")
t2<-paste0(path, "/table2")
table1 <- data.frame (Num = 1:10, Letter = rep("A", 10))
write.table (table1, t1, sep = ";")
table2 <- data.frame (Num = 1:10, Letter = rep("B", 10))
write.table (table2, t2, sep = ";")
read_and_unite_files (path = path, pattern = "table", header = TRUE, sep = ";",
                      add.file.id = TRUE)
read_and_unite_files (path = path, pattern = "table", header = TRUE, sep = ";",
                      add.file.id = TRUE,
                      file.id = data.frame (id1 = c(1,2), id2 = c("one", "two")))
file.remove (t1); file.remove (t2)
```

---

read_from_table_file *Read table file*

Description

Read table file and selects the required rows and columns.

Usage

```
read_from_table_file(
  file,
  choose.columns = FALSE,
  column.names,
  select = FALSE,
  select.column.name,
  select.val,
  unique = FALSE,
  sep = ";",
  header = TRUE
)
```

Arguments

`file`	character; file name and path
`choose.columns`	logical; return chosen columns only
`column.names`	character; vector of name of columns that are chosen to be returned
`select`	logical; return only rows that contain selected values in one column
`select.column.name`	character; name of column that contains selected values
`select.val`	vector of values that define rows that should be returned
`unique`	logical; delete duplicated rows
`sep`	character; the field separator character
`header`	logical; files contain the names of the variables as its first line

Details

This function reads table files and returns data frame with selected rows (only rows with specified values) and columns. Also duplicated rows may be deleted.

`column.names` and `select.column.name` must be exact column names as in data frame.

Value

Data frame with file content, optionally trimmed.

Author(s)

Elena N. Filatova

Examples

```
mydata <- data.frame (N = 1:10, letter = c(rep ("A", 5), rep ("B", 4), "C"),
                  num = c(1, rep(1:4, 2), 5))
t1<-tempfile()
write.table (mydata, t1, sep = ";")
read_from_table_file (file = t1)
```

```
read_from_table_file (file = t1, select = TRUE, select.column.name = "letter",
                      select.val = c("A", "C"))
read_from_table_file (file = t1, select = TRUE, select.column.name = "letter",
                      select.val = c("A", "C"), unique=TRUE, choose.columns = TRUE,
                      column.names = c("letter", "num"))
read_from_table_file (file = t1, select = TRUE, select.column.name = "letter",
                      select.val = c("A", "C"), unique = TRUE, choose.columns = TRUE,
                      column.names = c("N", "num"))
read_from_table_file (file = t1, select = TRUE, select.column.name = "letter",
                      select.val = c("A", "C"), unique = TRUE, choose.columns = TRUE,
                      column.names = c("letter", "N"))
file.remove (t1)
```

---

store_in_DB *Store data in SQLite database*

Description

Write, read and delete tables from SQLite database.

Usage

```
list_DB(database)

write_to_DB(
  database,
  data,
  table,
  overwrite = FALSE,
  append = FALSE,
  verbose = TRUE
)

index_DB(database, table, index.unique, index.column.name, verbose = TRUE)

read_from_DB(
  database,
  table,
  choose.columns = FALSE,
  column.names,
  select = FALSE,
  select.column.name,
  select.val,
  unique = FALSE
)

delete_from_DB(database, table, verbose = TRUE)
```

Arguments

database	character; SQLite database name and path.
data	data frame that should be stored as database table.
table	character; table name.
overwrite	logical; use overwrite = TRUE if you want to overwrite a table that already exists in database
append	logical; append rows to table
verbose	logical; show messages
index.unique	logical; vector of indicators to create unique or not unique indexes
index.column.name	vector of indexed columns' names
choose.columns	logical; return chosen columns only
column.names	character; vector of name of columns that are chosen to be returned
select	logical; return only rows that contain selected values in one column
select.column.name	character; name of column that contains selected values
select.val	vector of values that define rows that should be returned
unique	logical; delete duplicated rows

Details

This functions help to store big data frames in SQLite database which makes it faster to save and read the data.

This function creates SQLlite connection to database, fulfills the task and then disconnects. If no database has been created yet, creates one.

Do not use overwrite = TRUE if table does not exists. Do not use append = TRUE and overwrite = TRUE at the same time, no append is possible while overwriting.

If multiple indexes are created in one table, they are unrelated.

Do not use dots in data frame character variables, use underscore.

Parameters choose.columns=FALSE, column.names, select, select.column.name, select.val, unique are only used with linkread_from_DB function. Those parameters define rows and columns that will be returned.

Value

list_DB returns character vector of names of database tables.

read_from_DB returns a data frame with the content of SQLite table.

Functions

- list_DB: Lists all tables from SQLite database
- write_to_DB: Writes data frame into SQLite database table
- index_DB: Creates SQLite indexes in database table
- read_from_DB: Reads table from SQLite database and writes it into data frame.
- delete_from_DB: Deletes table from SQLite database.

Author(s)

Elena N. Filatova

Examples

```
mydata <- as.data.frame (matrix(1:10, 2, 5))
database <- tempfile()
write_to_DB (database, data = mydata, table = "table1", overwrite = FALSE)
list_DB (database)
mydata2 <- as.data.frame (matrix(11:20, 2, 5))
write_to_DB (database, data = mydata2, table = "table1", overwrite = TRUE)
mydata3 <- read_from_DB (database, table = "table1")
delete_from_DB (database, table = "table1")
file.remove (database)

# example with reading table with restricted columns and rows.
mydata <- data.frame(ids = c(1:6), titles = c("A", "B", "C", "D", "E", "E"),
                     other = rep("other", 6))
database <- tempfile()
write_to_DB (database, data = mydata, table = "table1", overwrite = FALSE)
read_from_DB(database, "table1", choose.columns = TRUE, column.names = c("ids", "titles", "other"),
            select = TRUE, select.column.name = "ids",  select.val = 3:6, unique = TRUE)
read_from_DB(database, "table1", choose.columns = TRUE, column.names = c("titles", "other"),
            select = TRUE, select.column.name = "ids",  select.val = 3:6, unique = TRUE)
file.remove (database)
```

---

summarize_blast_result

Summarize BLAST result

Description

Summarize aligned, not aligned and undesirably aligned sequences

Usage

```
summarize_blast_result(
  sum.aligned = "sp",
  blast.probe.id.var,
  blast.res.id.var,
  blast.res.title.var,
  reference.id.var,
  reference.title.var,
  titles = FALSE,
  add.blast.info = FALSE,
  data.blast.info,
  check.blast.for.source = FALSE,
```

```
  source = NULL,
  switch.ids = FALSE,
  switch.table,
  mc.cores = 1,
  digits = 2,
  sep = ";",
  temp.db = NULL,
  delete.temp.db = TRUE,
  return = "summary",
  write.alignment = "DB",
  alignment.db = NULL,
  alignment.table.sp.aligned = NULL,
  alignment.table.sp.not.aligned = NULL,
  alignment.table.nonsp = NULL,
  change.colnames.dots = TRUE,
  file.sp.aligned = NULL,
  file.sp.not.aligned = NULL,
  file.nonsp = NULL,
  verbose = TRUE
)
```

Arguments

`sum.aligned`	character; summarize specific or not specific alignments; possible values are `"sp"` (aligned and not aligned specific subjects) and `"nonsp"` (aligned non specific subjects)
`blast.probe.id.var`	vector of query identification numbers from BLAST result data
`blast.res.id.var`, `blast.res.title.var`	vector of subject identification numbers and titles from BLAST result data
`reference.id.var`, `reference.title.var`	vector of identification numbers and titles of specific sequences that should be or might be aligned
`titles`	logical; include titles in alignment reports
`add.blast.info`	logical; add other BLAST results
`data.blast.info`	data frame; additional BLAST result from BLAST result data
`check.blast.for.source`	logical; delete queries that are not aligned with one obligatory sequence
`source`	identification number of obligatory sequence for alignment
`switch.ids`	logical; use different identification numbers for BLAST result's subjects
`switch.table`	data frame; table of old and new identification numbers (and new titles) linked by row
`mc.cores`	integer; number of processors for parallel computation (not supported on Windows)
`digits`	integer; number of decimal places to round the result

sep	character; the field separator character
temp.db	character; temporal SQLite database name and path
delete.temp.db	logical; delete temporal SQLite database afterwards
return	character; returned object; possible values are "list" (list of data frames with alignment summary and report for each probe) and "summary" (data frame with summary for all probes is returned and alignment reports are written into files or SQLite database tables)
write.alignment	character; write alignment reports into files ("file") or SQLite database tables ("DB"; used if (return = "summary"))
alignment.db, alignment.table.sp.aligned, alignment.table.sp.not.aligned, alignment.table.nonsp	character; SQLite database name and path, tables names (used if write.alignment = "DB")
change.colnames.dots	logical; change dots to underscore in data frame column names (used if write.alignment = "DB")
file.sp.aligned, file.sp.not.aligned, file.nonsp	character; file names and path (used if write.alignment = "file")
verbose	logical; show messages

Details

This function works with data frame created by blast_local function. It takes BLAST results, divides aligned subjects on specific (that should be aligned) and non specific (that should not be aligned) according to reference) values. Function summarizes amount of aligned and not aligned specific subjects and amount of aligned non specific subjects.

When sum.aligned = "sp" aligned and not aligned specific subjects are summarized and reference.id.var and reference.title.var should contain sequences that it is necessary to align with. When sum.aligned = "nonsp" aligned non specific subjects are summarized and reference.id.var should contain sequences that may be aligned (that are not considered as non specific), no titles needed.

When return = "summary", function returns summary (amount of aligned and not aligned subjects) and writes sorted alignments (alignment report) in file (write.alignment = "file") or SQLite database (write.alignment = "DB"). Usually only subjects' ids and (optionally) titles are returned, but you may add as many BLAST results as you like with add.blast.info and data.blast.info parameters. If you add some BLAST results, all alignments will present in alignment report, if not - duplicated subjects will be deleted.

By default result tables in database (if write.alignment = "DB") are "sp_aligned", "sp_not_aligned" and "nonsp", Results are written by appending, so if files or tables already exist, data will be added into them.

If subjects identification numbers in BLAST result data differ from those in reference.id.var you may use switch.ids = TRUE to change BLAST ids into new according to switch.table. switch.table must be a data frame with column one - old ids, column two - new ids and (optionally) column three - new titles. Do not use dots in column names.

When check.blast.for.source = TRUE probes that are non blasted for one special subject (usually the sequence that was cut for probes) are deleted. No check.blast.for.source is performed if sum.aligned = "nonsp". Check for source is performed after the possible id.switch, so source should be identification number of same type as reference.

Probe identification number must be character variable.

If alignment report is written into database, probe identification variable is indexed in all tables. Also it is highly recommended to set change.colnames.dots = TRUE to change possible dots to underscore within result data frame's column names and avoid further mistakes.

While working function saves data in temporal SQLite database. Function will stop if same database already exists, so deleting temporal database is highly recommended.

Value

List of data frames with alignment summary and report for each probe or data frame with summary for all probes (alignment reports are written into files or SQLite database tables).

Author(s)

Elena N. Filatova

Examples

```
path <- tempdir()
dir.create (path)
# load blast results with subject accession numbers
data(blast.fill)
#load metadata of all Chlamydia pneumoniae sequences - they are subjects that
# do not count as nonspecific and may be aligned
data(meta.all)
# load metadata with target Chlamydia pneumoniae sequences - they are specific subjects
# that must be aligned
# make new accession numbers to count all WGS sequences as one (see unite_NCBI_ac.nums ())
meta.target.new.ids <- unite_NCBI_ac.nums (data = meta.target,
                                           ac.num.var = meta.target$GB_AcNum,
                                           title.var = meta.target$title,
                                           db.var = meta.target$source_db,
                                           type = "shotgun", order = TRUE,
                                           new.titles = TRUE)
# summarize blast results, count aligned specific subjects with "switch ids" option
# (WGS sequences are counted as one). Add query cover information.
blast.sum.sp <- summarize_blast_result (sum.aligned = "sp",
                                        blast.probe.id.var = blast.fill$Qid,
                                        blast.res.id.var = blast.fill$Racc,
                                        blast.res.title.var = blast.fill$Rtitle,
                                        reference.id.var = meta.target.new.ids$new.id,
                                     reference.title.var = meta.target.new.ids$new.title,
                                        titles = TRUE,
                                        add.blast.info = TRUE,
                                  data.blast.info = data.frame(Qcover = blast.fill$Qcover),
                                     switch.ids = TRUE, switch.table = meta.target.new.ids,
                                 temp.db = paste0 (path, "/temp.db"), delete.temp.db = TRUE,
```

```
                                        return = "summary", write.alignment = "DB",
                                        alignment.db = paste0 (path, "/alig.db"))
    # summarize nonspecific alignments (that are not in meta.all dataframe)
    blast.sum.nonsp <- summarize_blast_result (sum.aligned = "nonsp",
                                            blast.probe.id.var = blast.fill$Qid,
                                            blast.res.id.var = blast.fill$Racc,
                                            blast.res.title.var = blast.fill$Rtitle,
                                            reference.id.var = meta.all$GB_AcNum,
                                            reference.title.var = meta.all$title,
                                            titles = TRUE, switch.ids = FALSE,
                                            add.blast.info = TRUE,
                                    data.blast.info = data.frame(Qcover = blast.fill$Qcover),
                                            temp.db = paste0 (path, "/temp.db"),
                                            delete.temp.db = TRUE,
                                            return = "summary", write.alignment = "DB",
                                            alignment.db = paste0 (path, "/alig.db"))
    # all specific targets are aligned
    sp.aligned <- read_from_DB(database = paste0 (path, "/alig.db"), table = "sp_aligned")
    # no targets that are not aligned
    sp.not.aligned <- read_from_DB(database = paste0 (path, "/alig.db"), table = "sp_not_aligned")
    # No nonspecific alignments
    nonsp <- read_from_DB(database = paste0 (path, "/alig.db"), table = "nonsp")
    file.remove (paste0 (path, "/alig.db"))
```

`trim_DF` *Trim data frame*

Description

If the numeric value of the data frame variable does not meet the specified conditions, the function deletes the entire row.

Usage

```
trim_DF(data, trim.var.name, trim.action, trim.thresh)
```

Arguments

`data`	data frame
`trim.var.name`	character; vector of data frame column names with numeric variables that should be tested for conditions
`trim.action`	character; vector of test conditions; possible values are: `"more"`, `"eqmore"` (more or equal), `"less"`, `"eqless"` (less or equal).
`trim.thresh`	numeric; vector of condition threshold values

Details

This function takes the vector of data frame variables and for each of them test if they satisfy the specified conditions. Not satisfying values are deleted with the entire data frame row. You may set as many conditions for as many variables as you like.

`trim.values` must be exact column names as in data frame.

Value

data frame without rows with values that do not satisfy the specified conditions.

Author(s)

Elena N. Filatova

Examples

```
data <- data.frame ("a" = 1:10, "b" = 101:110)
trim_DF (data, trim.var.name = c("a", "b"), trim.action = c("less", "eqmore"),
       trim.thresh = c(6, 104))
```

`unite_NCBI_ac.nums` *Assigns master record's id to all project records*

Description

The function assigns the project master record's NCBI access number to all records that belong to the project.

Usage

```
unite_NCBI_ac.nums(
  data,
  ac.num.var,
  title.var,
  db.var,
  type = "shotgun",
  order = TRUE,
  new.titles = FALSE
)
```

Arguments

`data`	data frame; contains information about sequence records.
`ac.num.var`	character; data frame variable that contains sequence accession numbers.
`title.var`	character; data frame variable that contains sequence titles.

db.var	character; data frame variable that contains source data base names.
type	character; type of the project which records should be united with one accession number. At the moment "shotgun" is the only possible value which corresponds to the whole genome shotgun sequencing project with shotgun technology.
order	logical; rearrange a data frame in alphabetical order of accession numbers (highly recommended).
new.titles	logical; add new titles according to new access numbers.

Details

The function looks through all records in a data frame. If the record belongs to the project (for example, WGS-project), the function assigns the project master record's NCBI access number to this record. If the record is not related to any project, it retains its own accession number.

It is highly recommended to arrange the data in alphabetical order of accession numbers, since the first record among similar ones is determined as master record.

Value

If new.titles = FALSE data frame with old and new access numbers is returned.

If new.titles = TRUE data frame with old and new access numbers and new titles is returned.

Author(s)

Elena N. Filatova

Examples

```
# Example with sequences from WGS-project of Chlamydia pneumoniae genome
data (meta.target) #load metadata of target sequences with GenBank identificators
meta.target.new.ids <- unite_NCBI_ac.nums (data = meta.target,
                                           ac.num.var = meta.target$GB_AcNum,
                                           title.var = meta.target$title,
                                           db.var = meta.target$source_db,
                                           type = "shotgun", order = TRUE,
                                           new.titles = TRUE)
```

---

unite_two_DF *Combine two data frames*

Description

Combine two data frames according to shared variable

Usage

```
unite_two_DF(
  data1,
  data1.shared.var,
  data1.shared.column.num = 1,
  data2,
  data2.shared.var,
  data2.shared.column.num = 1,
  delete.not.shared = FALSE,
  not.shared = "all",
  verbose = TRUE
)
```

Arguments

data1, data2	data frames
data1.shared.var, data2.shared.var	same variables in data frames
data1.shared.column.num, data2.shared.column.num	integer; column numbers of same variables in data frames
delete.not.shared	logical; delete rows that present in one data frame but do not present in other data frame
not.shared	character; which rows to delete; possible values are "data1" (delete rows that present in data1 but do not present in data2), "data2" (delete rows that present in data2 but do not present in data1), "all" (both variants)
verbose	logical; show messages

Details

This function combines columns of two data frames according to shared.var which acts like rows' identification number. If shared.var value from one data frame do not present in other data frame, NAs are produced. Those absent rows are deleted when delete.not.shared = TRUE.

data1.shared.var and data2.shared.var must contain unique values within its own data frame.

Order of rows in resulting data frame is according to data1. data2.shared.var is removed from resulting data frame.

Value

Combined data frame.

Author(s)

Elena N. Filatova

Examples

```
#same values in shared variables
data1 <- data.frame (N = 1:5, letter = rep("A", 5))
data2 <- data.frame (N = 1:5, letter = rep("B", 5), cs = rep("cs",5))
unite_two_DF (data1 = data1, data1.shared.var = data1$N, data2 = data2, data2.shared.var = data2$N,
            delete.not.shared = TRUE, not.shared = "all")
#different values in shared variables
data1 <- data.frame (N = 1:5, letter = rep("A", 5))
data2 <- data.frame (N = 3:8, letter = rep("B", 6), cs = rep("cs",6))
unite_two_DF (data1 = data1, data1.shared.var = data1$N, data2 = data2, data2.shared.var = data2$N)
unite_two_DF (data1 = data1, data1.shared.var = data1$N, data2 = data2, data2.shared.var = data2$N,
            delete.not.shared = TRUE, not.shared = "data1")
unite_two_DF (data1 = data1, data1.shared.var = data1$N, data2 = data2, data2.shared.var = data2$N,
            delete.not.shared = TRUE, not.shared = "data2")
unite_two_DF (data1 = data1, data1.shared.var = data1$N, data2 = data2, data2.shared.var = data2$N,
            delete.not.shared = TRUE, not.shared = "all")
```

Index