Package 'topdownr'

October 17, 2024

Title Investigation of Fragmentation Conditions in Top-Down Proteomics

Version 1.26.0

Description The topdownr package allows automatic and systemic investigation of fragment conditions. It creates Thermo Orbitrap Fusion Lumos method files to test hundreds of fragmentation conditions. Additionally it provides functions to analyse and process the generated MS data and determine the best conditions to maximise overall fragment coverage.

Depends R (>= 3.5), methods, BiocGenerics (>= 0.20.0), ProtGenerics (>= 1.10.0), Biostrings (>= 2.42.1), S4Vectors (>= 0.12.2)

Imports grDevices, stats, tools, utils, Biobase, Matrix (>= 1.4-2), MSnbase (>= 2.3.10), PSMatch (>= 1.6.0), ggplot2 (>= 2.2.1), mzR (>= 2.27.5)

Suggests topdownrdata (>= 0.2), knitr, rmarkdown, ranger, testthat, BiocStyle, xml2

License GPL (>= 3)

URL https://github.com/sgibb/topdownr/

BugReports https://github.com/sgibb/topdownr/issues/

LazyData true

VignetteBuilder knitr

Roxygen list(markdown=TRUE)

RoxygenNote 7.3.0

biocViews ImmunoOncology, Infrastructure, Proteomics, MassSpectrometry, Coverage

Encoding UTF-8

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

git_branch RELEASE_3_19

git_last_commit 8b077cb

git_last_commit_date 2024-04-30

Repository Bioconductor 3.19

Date/Publication 2024-10-16

Author Sebastian Gibb [aut, cre] (<https://orcid.org/0000-0001-7406-4443>),
Pavel Shliaha [aut] (<https://orcid.org/0000-0003-3092-0724>),
Ole Nørregaard Jensen [aut] (<https://orcid.org/0000-0003-1862-8528>)

Maintainer Sebastian Gibb <mail@sebastiangibb.de>

Contents

`topdownr-package` *Investigation of Fragmentation Conditions in Top-Down Proteomics*

Description

The topdownr package allows automatic and systemic investigation of fragment conditions. It creates Thermo Orbitrap Fusion Lumos method files to test hundreds of fragmentation conditions. Additionally it provides functions to analyse and process the generated MS data and determine the best conditions to maximise overall fragment coverage.

Details

The usage of the topdownr package is demonstrated in the following vignettes:

- Generate .meth files prior data acquisition for the Thermo Orbitrap Fusion Lumos MS devise: `vignette("data-generation", package="topdownr")`.
- How to analyse top-down fragmenation data: `vignette("analysis", package="topdownr")`

Author(s)

Sebastian Gibb <mail@sebastiangibb.de>, Pavel Shliaha <pavels@bmb.sdu.dk>, Ole Nørregaard Jensen <jenseno@bmb.sdu.dk>

References

https://github.com/sgibb/topdownr/

See Also

Useful links:

- https://github.com/sgibb/topdownr/
- Report bugs at https://github.com/sgibb/topdownr/issues/

AbstractTopDownSet-class

The AbstractTopDownSet class

Description

Abstract/VIRTUAL parent class for TopDownSet and NCBSet to provide common interface.

Usage

```
## S4 method for signature 'AbstractTopDownSet,ANY,ANY,ANY'
x[i, j, ..., drop = FALSE]

## S4 method for signature 'AbstractTopDownSet,ANY,missing'
x[[i, j, ...]]

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

## S4 method for signature 'AbstractTopDownSet'
x$name

## S4 replacement method for signature 'AbstractTopDownSet'
x$name <- value

## S4 method for signature 'AbstractTopDownSet'
assayData(object)

## S4 method for signature 'AbstractTopDownSet'
colData(object)

## S4 replacement method for signature 'AbstractTopDownSet'
colData(object, ...) <- value

## S4 method for signature 'AbstractTopDownSet,AbstractTopDownSet'
combine(x, y)
```

```
## S4 method for signature 'AbstractTopDownSet'
conditionData(object, ...)

## S4 replacement method for signature 'AbstractTopDownSet'
conditionData(object, ...) <- value

## S4 method for signature 'AbstractTopDownSet'
conditionNames(object)

## S4 method for signature 'AbstractTopDownSet'
dim(x)

## S4 method for signature 'AbstractTopDownSet'
dimnames(x)

## S4 method for signature 'AbstractTopDownSet'
removeEmptyConditions(object)

## S4 method for signature 'AbstractTopDownSet'
rowViews(object, ...)

## S4 method for signature 'AbstractTopDownSet'
show(object)

## S4 method for signature 'AbstractTopDownSet'
summary(object, what = c("rows", "columns"), ...)

## S4 method for signature 'AbstractTopDownSet'
updateConditionNames(
  object,
  sampleColumns = c("Mz", "AgcTarget", "EtdReagentTarget", "EtdActivation",
    "CidActivation", "HcdActivation", "UvpdActivation"),
  verbose = interactive()
)

## S4 method for signature 'AbstractTopDownSet'
updateMedianInjectionTime(
  object,
  by = list(Mz = object$Mz, AgcTarget = object$AgcTarget)
)
```

Arguments

i, j	numeric, logical or character, indices specifying elements to extract or replace.
drop	logical, currently ignored.
value	replacment value.
name	character name of an (non)existing column in colData.

object, x	AbstractTopDownSet
y	AbstractTopDownSet
what	character, specifies whether "rows" or "columns" should be summarized.
sampleColumns	character, column names of the colData() used to define a sample (technical replicate). This is used to add the Sample column (used for easier aggregation, etc.).
verbose	logical, verbose output?
by	list, grouping information.
...	arguments passed to internal/other methods.

Details

This class just provides a common interface. It is not intended for direct use by the user. Please see TopDownSet for an example usage of its child class.

Value

This is an *Abstract/VIRTUAL* class to provide a common interface for TopDownSet and NCBSet. It is not possible to create an AbstractTopDownSet object.

Methods (by generic)

- x[i: Subset operator.

  For i numeric, logical or character vectors or empty (missing) or NULL are supported. Subsetting is done on the fragment/bond (row) level. character indices could be names (e.g. c("a1", "b1", "c1", "c2", "c3")) or types (e.g. c("c", "x")) of the fragments for TopDownSet objects, or names of the bonds (e.g. c("bond001")) for NCBSet objects.
  j could be a numeric or logical vector and subsetting is done on the condition/run (column) level.

- x[[i: Subset operator.

  i could be a numeric or logical vector and subsetting is done on the condition/run (column) level.

- `[[`(x = AbstractTopDownSet, i = ANY, j = missing) <- value: Setter for a column in the colData slot.

  The [[<- operator is used to add/replace a single column of the colData DataFrame.

- $: Accessor for columns in the colData slot.

  The $ simplifies the accession of a single column of the colData. It is identical to the [[operator.

- `$`(AbstractTopDownSet) <- value: Setter for a column in the colData slot.

  The $<- operator is used to add/replace a single column of the colData DataFrame. It is identical to the [[<- operator.

- assayData(AbstractTopDownSet): Accessor for the assay slot.

  Returns a Matrix::dgCMatrix that stores the intensity/coverage information of AbstractTopDownSet object.

- `colData(AbstractTopDownSet)`: Accessor for the `colData` slot.

  Returns a S4Vectors::DataFrame that stores metadata for the conditons/runs (columns) of the AbstractTopDownSet object.

- `colData(AbstractTopDownSet) <- value`: Setter for the `colData` slot.

  Replaces metadata for the conditons/runs (columns) of the AbstractTopDownSet object.

- `combine(x = AbstractTopDownSet, y = AbstractTopDownSet)`: Combine `AbstractTopDownSet` objects.

  `combine` allows to combine two or more `AbstractTopDownSet` objects. Please note that it uses the default `sampleColumns` to define technical replicates (see `readTopDownFiles()`).and the default by argument to group ion injection times for the calculation of the median time (see `updateMedianInjectionTime()`). Both could be modified after `combine` by calling `updateConditionNames()` (with modified `sampleColumns` argument) and `updateMedianInjectionTime()` (with modified by argument).

- `conditionData(AbstractTopDownSet)`: Accessor for the `colData` slot.

  An alias for `colData`.

- `conditionData(AbstractTopDownSet) <- value`: Setter for the `colData` slot.

  An alias for `colData<-`.

- `conditionNames(AbstractTopDownSet)`: Accessor for condition names.

  Returns a `character` with names for the conditions/runs (columns).

- `dim(AbstractTopDownSet)`: Accessor for dimensions.

  Returns a `numeric` with number of fragments/bonds (rows) and conditions/runs (columns).

- `dimnames(AbstractTopDownSet)`: Accessor for dimension names.

  Returns a `list` with names for the fragments/bonds (rows) and for the conditions/runs (columns).

- `removeEmptyConditions(AbstractTopDownSet)`: Remove empty conditions/runs.

  Removes conditions/runs (columns) without any intensity/coverage information from the AbstractTopDownSet object. It returns a modified AbstractTopDownSet object.

- `rowViews(AbstractTopDownSet)`: Accessor for the `rowViews` slot.

  Depending on the implementation it returns an FragmentViews object for TopDownSet objects or an Biostrings::XStringViews object for NCBSet objects.

- `summary(AbstractTopDownSet)`: Summary statistics.

  Returns a `matrix` with some statistics: number of fragments, total/min/first quartile/median/mean/third quartile/maximum of intensity values.

- `updateConditionNames(AbstractTopDownSet)`: Update condition names.

  Updates condition names based on `sampleColumns` from `conditionData/colData`. Columns with just identical entries are ignored. This method will create/update the `colData(object)$Sample` column that identifies technical replicates and could be used in other methods.

- `updateMedianInjectionTime(AbstractTopDownSet)`: Update median ion injection times.

  Recalculates median ion injection times by a user given grouping variable (default: Mz, AgcTarget). This is useful if you acquire new data and the ion injection time differs across the runs. Use the by argument to provide a `list/data.frame` of grouping variables, e.g. `by=colData(object)[, c("Mz", "AgcTarget", "File")]`.

Slots

`rowViews` Biostrings::XStringViews, information about fragments/bonds (name, type, sequence, mass, charge), see Biostrings::XStringViews and FragmentViews for details.

`colData` S4Vectors::DataFrame, information about the MS2 experiments and the fragmentation conditions.

`assay` Matrix::dgCMatrix, intensity/coverage values of the fragments/bonds. The rows correspond to the fragments/bonds and the columns to the condition/run. It just stores values that are different from zero.

`files character`, files that were imported.

`processing character`, log messages.

Author(s)

Sebastian Gibb <mail@sebastiangibb.de>

See Also

- TopDownSet and NCBSet which both implement/use this interface. These manual pages also provide some example code.
- FragmentViews (and Biostrings::XStringViews) for the row view interface.
- Matrix::dgCMatrix for technical details about the intensity/coverage storage.

Examples

```
## Because AbstractTopDownSet is a VIRTUAL class we could not create any
## object of it. Here we demonstrate the usage with an TopDownSet that
## implements the AbstractTopDownSet interface. See `?"TopDownSet-class"` for
## more details an further examples.

## Example data
data(tds, package="topdownr")

tds

head(summary(tds))

# Accessing slots
rowViews(tds)
colData(tds)
head(assayData(tds))

# Accessing colData
tds$Mz
tds$FilterString

# Subsetting

# First 100 fragments
tds[1:100]
```

```
# All c fragments
tds["c"]

# Just c 152
tds["c152"]

# Condition 1 to 10
tds[, 1:10]
```

`createExperimentsFragmentOptimisation` *Create fragment optimisation experiment*

Description

This function is used to create a tree-like `list` of all combinations of a user-given set of MS1 and TMS2 settings for an fragment optimisation experiment. The list could be written to an Orbitrap Fusion Lumos method xml file using `writeMethodXmls()`.

Usage

```
createExperimentsFragmentOptimisation(
  ms1,
  ...,
  groupBy = c("AgcTarget", "replication"),
  nMs2perMs1 = 10,
  scanDuration = 0,
  replications = 2,
  randomise = TRUE
)
```

Arguments

`ms1`	`data.frame`, MS1 settings.
`...`	further named arguments with `data.frames` containing the TMS2 settings.
`groupBy`	`character`, group experiments by columns in the TMS2 `data.frames`. The columns have to be present in all `data.frames`. Each group will be written to its own XML file.
`nMs2perMs1`	`integer`, how many TMS2 scans should be run after a MS1 scan?
`scanDuration`	`double`, if greater than zero (e.g. `scanDuration=0.5`) the Start/EndTimeMin are overwritten with a duration of `scanDuration`. If `scanDuration` is zero (default) Start/EndTimeMin are not overwritten.
`replications`	`integer`, number of replications.
`randomise`	`logical`, should the TMS2 scan settings randomised?

Value

`list`, able to be written via `xml2::as_xml_document()`

See Also

`writeMethodXmls()`, `expandMs1Conditions()`, `expandTms2Conditions()`

Examples

```
## build experiments within R
ms1 <- expandMs1Conditions(
    FirstMass=400,
    LastMass=1200,
    Microscans=as.integer(10)
)

targetMz <- cbind(mz=c(560.6, 700.5, 933.7), z=rep(1, 3))
common <- list(
    OrbitrapResolution="R120K",
    IsolationWindow=1,
    MaxITTimeInMS=200,
    Microscans=as.integer(40),
    AgcTarget=c(1e5, 5e5, 1e6)
)

cid <- expandTms2Conditions(
    MassList=targetMz,
    common,
    ActivationType="CID",
    CIDCollisionEnergy=seq(7, 35, 7)
)
hcd <- expandTms2Conditions(
    MassList=targetMz,
    common,
    ActivationType="HCD",
    HCDCollisionEnergy=seq(7, 35, 7)
)
etd <- expandTms2Conditions(
    MassList=targetMz,
    common,
    ActivationType="ETD",
    ETDReactionTime=as.double(1:2)
)
etcid <- expandTms2Conditions(
    MassList=targetMz,
    common,
    ActivationType="ETD",
    ETDReactionTime=as.double(1:2),
    ETDSupplementalActivation="ETciD",
    ETDSupplementalActivationEnergy=as.double(1:2)
)
uvpd <- expandTms2Conditions(
```

```
    MassList=targetMz,
    common,
    ActivationType="UVPD"
)

exps <- createExperimentsFragmentOptimisation(
    ms1=ms1, cid, hcd, etd, etcid, uvpd,
    groupBy=c("AgcTarget", "replication"), nMs2perMs1=10, scanDuration=0.5,
    replications=2, randomise=TRUE
)

## use different settings for CID
cid560 <- expandTms2Conditions(
    MassList=cbind(560.6, 1),
    common,
    ActivationType="CID",
    CIDCollisionEnergy=seq(7, 21, 7)
)
cid700 <- expandTms2Conditions(
    MassList=cbind(700.5, 1),
    common,
    ActivationType="CID",
    CIDCollisionEnergy=seq(21, 35, 7)
)

exps <- createExperimentsFragmentOptimisation(
    ms1=ms1, cid560, cid700,
    groupBy=c("AgcTarget", "replication"), nMs2perMs1=10, scanDuration=0.5,
    replications=2, randomise=TRUE
)

## use a CSV (or excel) file as input
myCsvContent <- "
ActivationType, ETDReactionTime, UVPDActivationTime
UVPD,,1000
ETD,1000,
"
myCsvSettings <- read.csv(text=myCsvContent, stringsAsFactors=FALSE)
myCsvSettings
#   ActivationType ETDReactionTime UVPDActivationTime
# 1           UVPD              NA               1000
# 2            ETD            1000                 NA

exps <- createExperimentsFragmentOptimisation(
    ms1 = data.frame(FirstMass=500, LastMass=1000),
    ## TMS2
    myCsvSettings,
    ## other arguments
    groupBy="ActivationType"
)
```

createTngFusionMethFiles
Windows specific functions.

Description

The functions runXmlMethodChanger and runScanHeadsman call XmlMethodChanger.exe and ScanHeadsman.exe with the corresponding arguments. The only work on Windows (maybe on Linux + wine as well but that was never tested).

Usage

```
createTngFusionMethFiles(
  template,
  xml = list.files(pattern = ".*\\.xml$"),
  executable = "XmlMethodChanger.exe",
  verbose = interactive()
)

runXmlMethodChanger(
  template,
  xml = list.files(pattern = ".*\\.xml$"),
  executable = "XmlMethodChanger.exe",
  verbose = interactive()
)

runScanHeadsman(path = ".", executable = "ScanHeadsman.exe")
```

Arguments

template	character, path to template .meth file.
xml	character, vector of path to .xml files.
executable	character, path to the XmlMethodChanger.exe or ScanHeadsman.exe executable.
verbose	logical, if TRUE a progress bar is shown.
path	character, path to the directory containing the .raw files.

Details

runXmlMethodChanger applies 'XmlMethodChanger.exe' on all given XML files generated with writeMethodXmls() to create .meth files from a template.

runScanHeadsman calls ScanHeadsman.exe on a given directory containing .raw files. ScanHeadsman.exe extracts the method and scan header data into .experiments.csv and .txt files, respectively.

Value

Nothing. Used for its side effects.

References

XmlMethodChanger source code: https://github.com/thermofisherlsms/meth-modifications/

ScanHeadsman source code: https://bitbucket.org/caetera/scanheadsman

See Also

writeMethodXmls()

Examples

```
## Not run:
runXmlMethodChanger(templateMeth="TMS2IndependentTemplate240Extended.meth",
                    modificationXml=list.files(pattern="^method.*\\.xml$"),
                    executable="..\\XmlMethodChanger.exe")

## End(Not run)
## Not run:
runScanHeadsman("raw", executable="..\\ScanHeadsman.exe")

## End(Not run)
```

---

expandMs1Conditions *Expand MS Conditions*

Description

Create a data.frame of all possible combinations of the given arguments. It ensures that just arguments are applied that yield a valid MethodModification.xml file.

Usage

```
expandMs1Conditions(..., family = "Calcium", version = "3.2")

expandTms2Conditions(
  ActivationType = c("CID", "HCD", "ETD", "UVPD"),
  ...,
  MassList = NULL,
  family = "Calcium",
  version = "3.2"
)
```

Arguments

...	further named arguments, used to create the combination of conditions.
family	character, currently just Calcium is supported
version	character, currently 3.1, 3.2 (default), 3.3 are supported
ActivationType	character, *ActivationType* for TMS2, either CID, HCD, ETD, or UVPD.

MassList — matrix, 2 columns (mass, z) for targeted mass list, or NULL (default) to not overwrite targeted mass.

Value

data.frame with all possible combinations of conditions/settings.

See Also

validMs1Settings()

validTms2Settings(), expand.grid()

Examples

```
expandMs1Conditions(FirstMass=100, LastMass=400)
expandTms2Conditions(
     ActivationType="CID",
     OrbitrapResolution="R120K",
     IsolationWindow=1,
     MaxITTimeInMS=200,
     Microscans=as.integer(40),
     AgcTarget=c(1e5, 5e5, 1e6),
     CIDCollisionEnergy=c(NA, seq(7, 35, 7)),
     MassList=cbind(mz=c(560.6, 700.5, 933.7), z=rep(1, 3))
)
```

FragmentViews-class *The FragmentViews class*

Description

The FragmentViews class is a basic container for storing a set of views (start/end locations) on the same peptides/protein sequence. Additionally it keeps information about mass, type and charge of the fragments.

Usage

```
FragmentViews(
  sequence,
  mass,
  type,
  z = 1L,
  start = NULL,
  end = NULL,
  width = NULL,
  names = NULL,
  metadata = list()
)
```

```
## S4 method for signature 'FragmentViews,FragmentViews'
combine(x, y)

## S4 method for signature 'FragmentViews'
mz(object, ...)

## S4 method for signature 'FragmentViews'
show(object)
```

Arguments

sequence	character/ Biostrings::AAString, complete protein/peptide sequence.
mass	double, mass of the fragments, same length as start/end/width.
type	character, type of the fragments, same length as start/end/width‘.
z	integer, charge of the fragments, length one or same length as start/end/width‘.
start	integer, start positions of the fragments. At least two of start/end/width‘ has to be given.
end	integer, end positions of the fragments. At least two of start/end/width‘ has to be given.
width	integer, width positions of the fragments. At least two of start/end/width‘ has to be given.
names	character, names of the fragments, same length as start/end/width‘.
metadata	list, metadata like modifications.
object, x, y	FragmentViews
...	arguments passed to internal/other methods.

Details

FragmentViews extends Biostrings::XStringViews. In short it combines an IRanges::IRanges object to store start/end location on a sequence, an Biostrings::AAString object.

Value

An FragmentViews object.

Functions

- `FragmentViews()`: Constructor

  In general it is not necessary to call the constructor manually. See `readTopDownFiles()` instead.

Coercion

`as(object, "data.frame")`: Coerce an FragmentViews object into a `data.frame`.

Author(s)

Sebastian Gibb <mail@sebastiangibb.de>

See Also

Biostrings::XStringViews

Examples

```
# Constructor
fv <- FragmentViews("ACE", start=1, width=1:3, names=paste0("b", 1:3),
                    mass=c(72.04439, 232.07504, 361.11763),
                    type="b", z=1)
fv

# Coercion to data.frame
as(fv, "data.frame")
as(fv, "data.frame")
```

NCBSet-class *The NCBSet class*

Description

The NCBSet class is a container for a top-down proteomics experiment similar to the TopDownSet but instead of intensity values it just stores the information if a bond is covered by a N-terminal (encoded as 1), a C-terminal (encoded as 2) and/or bidirectional fragments (encoded as 3).

Usage

```
## S4 method for signature 'NCBSet'
bestConditions(
  object,
  n = ncol(object),
  minN = 0L,
  maximise = c("fragments", "bonds"),
  ...
)

## S4 method for signature 'NCBSet'
fragmentationMap(
  object,
  nCombinations = 10,
  cumCoverage = TRUE,
  maximise = c("fragments", "bonds"),
  labels = colnames(object),
  alphaIntensity = TRUE,
  ...
```

```
)

## S4 method for signature 'NCBSet'
show(object)

## S4 method for signature 'NCBSet'
summary(object, what = c("conditions", "bonds"), ...)
```

Arguments

object	NCBSet
n	integer, max number of combinations/iterations.
minN	integer, stop if there are less than minN additional fragments
maximise	character, optimisation targeting for the highest number of "fragments" (default) or "bonds".
nCombinations	integer, number of combinations to show (0 to avoid plotting them at all).
cumCoverage	logical, if TRUE (default) cumulative coverage of combinations is shown.
labels	character, overwrite x-axis labels.
alphaIntensity	logical, if TRUE (default) the alpha level of the color is used to show the colData(object)$AssignedIntensity. If FALSE the alpha is set to 1.
what	character, specifies whether "conditions" (columns; default) or "bonds" (rows) should be summarized.
...	arguments passed to internal/other methods. added.

Value

An NCBSet object.

Methods (by generic)

- bestConditions(NCBSet): Best combination of conditions.
  Finds the best combination of conditions for highest coverage of fragments or bonds. If there are two (or more conditions) that would add the same number of fragments/bonds the one with the highest assigned intensity is used. Use n to limit the number of iterations and combinations that should be returned. If minN is set at least minN fragments have to be added to the combinations. The function returns a 7-column matrix. The first column contains the index (Index) of the condition (column number). The next columns contain the newly added number of fragments or bonds (FragmentsAddedToCombination, BondsAddedToCombination), the fragments or bonds in the condition (FragmentsInCondition, BondsInCondition), and the cumulative coverage fragments or bonds (FragmentCoverage, BondCoverage).
- fragmentationMap(NCBSet): Plot fragmentation map.
  Plots a fragmentation map of the Protein. Use nCombinations to add another plot with nCombinations combined conditions. If cumCoverage is TRUE (default) these combinations increase the coverage cumulatively.
- summary(NCBSet): Summary statistics.
  Returns a matrix with some statistics: number of fragments, total/min/first quartile/median/mean/third quartile/maximum of intensity values.

Slots

`rowViews` Biostrings::XStringViews, information about bonds (name, start, end, width, sequence), see Biostrings::XStringViews for details.

`colData` S4Vectors::DataFrame, information about the MS2 experiments and the fragmentation conditions.

`assay` Matrix::dgCMatrix, coverage values of the bonds. The rows correspond to the bonds and the columns to the condition/run. It just stores values that are different from zero. If a bond is covered by an N-terminal fragment its encoded with `1`, by an C-terminal fragmentl with `2` and by both fragment types/bidirectional by `3` respectively.

`files character`, files that were imported.

`processing character`, log messages.

Author(s)

Sebastian Gibb <mail@sebastiangibb.de>

See Also

- An `NCBSet` is generated from an TopDownSet object.
- Biostrings::XStringViews for the row view interface.
- Matrix::dgCMatrix for technical details about the coverage storage.

Examples

```
## Example data
data(tds, package="topdownr")

## Aggregate technical replicates
tds <- aggregate(tds)

## Coercion into an NCBSet object
ncb <- as(tds, "NCBSet")

ncb

head(summary(ncb))

# Accessing slots
rowViews(ncb)
colData(ncb)
head(assayData(ncb))

# Accessing colData
ncb$Mz

# Subsetting

# First 100 bonds
ncb[1:100]
```

```
# Just bond 152
ncb["bond152"]

# Condition 1 to 10
ncb[, 1:10]

# Plot fragmentation map
fragmentationMap(ncb)
```

`readTopDownFiles` *Read top-down files.*

Description

It creates an TopDownSet object and is its only constructor.

Usage

```
readTopDownFiles(
  path,
  pattern = ".*",
  type = c("a", "b", "c", "x", "y", "z"),
  modifications = c("Carbamidomethyl", "Acetyl", "Met-loss"),
  customModifications = data.frame(),
  adducts = data.frame(),
  neutralLoss = PSMatch::defaultNeutralLoss(),
  sequenceOrder = c("original", "random", "inverse"),
  tolerance = 5e-06,
  redundantIonMatch = c("remove", "closest"),
  redundantFragmentMatch = c("remove", "closest"),
  dropNonInformativeColumns = TRUE,
  sampleColumns = c("Mz", "AgcTarget", "EtdReagentTarget", "EtdActivation",
    "CidActivation", "HcdActivation", "UvpdActivation"),
  conditions = "ScanDescription",
  verbose = interactive()
)
```

Arguments

`path`	`character`, path to directory that contains the top-down files.
`pattern`	`character`, a filename pattern, the default `.*` means all files.
`type`	`character`, type of fragments, currently *a-c* and *x-z* are supported, see `PSMatch::calculateFragments(` for details.
`modifications`	`character`, unimod names of modifications that should be applied. Currenlty just *Acetyl* (Unimod:1 but just protein N-term), *Carbamidomethyl* (Unimod:4) and *Met-loss* (Unimod:765) are supported. *Met-loss* removes M (if followed by

A, C, G, P, S, T, or V; (see also http://www.unimod.org/modifications_view.php?editid1=1, http://www.unimod.org/modifications_view.php?editid1=4, and http://www.unimod.org/modifications_vi for details)). Use NULL to disable all modifications.

customModifications
: data.frame, with 4 columns, namely: mass, name, location, variable, see details section.

adducts
: data.frame, with 3 columns, namely: mass, name, to, see details section.

neutralLoss
: list, neutral loss that should be applied, see PSMatch::calculateFragments() and PSMatch::defaultNeutralLoss() for details.

sequenceOrder
: character, order of the sequence before fragment calculation and matching is done. "original" doesn't change anything. "inverse" reverse the sequence and "random" arranges the amino acid sequence at random.

tolerance
: double, tolerance in *ppm* that is used to match the theoretical fragments with the observed ones.

redundantIonMatch
: character, a mz could be matched to one, two or more fragments. If it is matched against more than one fragment the match could be "remove"d or the match to the "closest" fragment could be chosen.

redundantFragmentMatch
: character, one or more mz could be matched to the same fragment, these matches could be "remove"d or the match to the "closest" mz is chosen.

dropNonInformativeColumns
: logical, should columns with just one identical value across all runs be removed?

sampleColumns
: character, column names of the colData() used to define a sample (technical replicate). This is used to add the Sample column (used for easier aggregation, etc.).

conditions
: character/numeric, one of:

  - "ScanDescription" (default): create condition IDs based on the given "Scan Description" parameter (set automatically by createExperimentsFragmentOptimisation()
  - "FilterString": create condition IDs based on mass labels in the *Filter-String* column (included for backward-compatibilty, used in writeMethodXmls() prior version 1.5.2 in Dec 2018).
  - A single numeric value giving the number of conditions.

verbose
: logical, verbose output?

Details

readTopDownFiles reads and processes all top-down files, namely:

- .fasta (protein sequence)
- .mzML (spectra)
- .experiments.csv (method/fragmentation conditions)
- .txt (scan header information)

customModifications: additional to the provided unimod modifications available through the modifications argument customModifications allow to apply user-definied modifications to the output of PSMatch::calculateFragments(). The customModifications argument takes a data.frame with the mass to add, the name of the modification, the location (could be the position of the amino acid or "N-term"/"C-term"), whether the modification is always seen (variable=FALSE) or both, the modified and unmodified amino acid are present (variable=TRUE), e.g. for Activation (which is available via modification="Acetyl") data.frame(mass=42.010565, name="Acetyl", location="N-term", variable=FALSE) or variable one (that could be present or not): data.frame(mass=365.132, name="Custom", location=10, variable=TRUE)

If the customModifications data.frame contains multiple columns the modifications are applied from row one to the last row one each time.

adducts: *Thermo's Xtract* allows some mistakes in deisotoping, mostly it allows +/- C13-C12 and +/- H+. The adducts argument takes a data.frame with the mass to add, the name that should assign to these new fragments and an information to whom the modification should be applied, e.g. for H+ on z, data.frame(mass=1.008, name="zpH", to="z").

Please note: The adducts are added to the output of PSMatch::calculateFragments(). That has some limitations, e.g. neutral loss calculation could not be done in topdownr-package. If neutral loss should be applied on adducts you have to create additional rows, e.g.: data.frame(mass=c(1.008, 1.008), name=c("cpH", "cpH_"), to=c("c", "c_")).

Value

A TopDownSet object.

See Also

PSMatch::calculateFragments(), PSMatch::defaultNeutralLoss()

Examples

```
if (require("topdownrdata")) {
    # add H+ to z and no neutral loss of water
    tds <- readTopDownFiles(
        topdownrdata::topDownDataPath("myoglobin"),
        ## Use an artifical pattern to load just the fasta
        ## file and files from m/z == 1211, ETD reagent
        ## target 1e6 and first replicate to keep runtime
        ## of the example short
        pattern=".*fasta.gz$|1211_.*1e6_1",
        adducts=data.frame(mass=1.008, name="zpH", to="z"),
        neutralLoss=PSMatch::defaultNeutralLoss(
            disableWaterLoss=c("Cterm", "D", "E", "S", "T")),
        tolerance=25e-6
   )
}
```

tds *TopDownSet Example Data*

Description

An example data set for topdownr. It is just a subset of the myoglobin dataset available in topdownrdata::topdownrdata-package.

Usage

```
tds
```

Format

A TopDownSet with 14901 fragments (1949 rows, 351 columns).

Details

It was created as follows:

```
tds <- readTopDownFiles(
   topdownrdata::topDownDataPath("myoglobin"),
   ## Use an artifical pattern to load just the fasta
   ## file and files from m/z == 1211, ETD reagent
   ## target 1e6 and first replicate to keep runtime
   ## of the example short
   pattern=".*fasta.gz$|1211_.*1e6_1",
   adducts=data.frame(mass=1.008, name="zpH", to="z"),
   neutralLoss=PSMatch::defaultNeutralLoss(
       disableWaterLoss=c("Cterm", "D", "E", "S", "T")),
   tolerance=25e-6)
```

Source

Subset taken from the topdownrdata::topdownrdata-package package.

topdownr-deprecated *Deprecated functions in topdownr*

Description

These functions are provided for compatibility with older versions of 'MyPkg' only, and will be defunct at the next release.

Details

The following functions are deprecated and will be made defunct; use the replacement indicated below:

- `defaultMs1Settings`: `expandMs1Conditions()` in combination with `createExperimentsFragmentOptimisation(`
- `defaultMs2Settings`: `expandTms2Conditions()` in combination with `createExperimentsFragmentOptimisation`

`TopDownSet-class` *The TopDownSet class*

Description

The TopDownSet class is a container for a whole top-down proteomics experiment.

Usage

```
## S4 method for signature 'TopDownSet'
aggregate(x, by = x$Sample, ...)

## S4 method for signature 'TopDownSet,TopDownSet'
combine(x, y)

## S4 method for signature 'TopDownSet'
filterCv(object, threshold, by = object$Sample, ...)

## S4 method for signature 'TopDownSet'
filterInjectionTime(
  object,
  maxDeviation = log2(3),
  keepTopN = 2,
  by = object$Sample,
  ...
)

## S4 method for signature 'TopDownSet'
filterIntensity(object, threshold, relative = TRUE, ...)

## S4 method for signature 'TopDownSet'
filterNonReplicatedFragments(object, minN = 2, by = object$Sample, ...)

## S4 method for signature 'TopDownSet'
normalize(object, method = "TIC", ...)

## S4 method for signature 'TopDownSet,missing'
plot(x, y, ..., verbose = interactive())
```

```
## S4 method for signature 'TopDownSet'
show(object)

## S4 method for signature 'TopDownSet'
summary(object, what = c("conditions", "fragments"), ...)
```

Arguments

x, object	TopDownSet
by	list, grouping variable, in general it refers to technical
y	missing, not used.
threshold	double, threshold variable.
maxDeviation	double, maximal allowed deviation in the log2 injection time in ms in comparison to the median ion injection time.
keepTopN	integer, how many technical replicates should be kept?
relative	logical, if relative is TRUE all fragments with intensity below threshold * max(intensity) per fragment are removed, otherwise all fragments below threshold are removed.
minN	numeric, if less than minN of a fragment are found across technical replicates it is removed.
method	character, normalisation method, currently just "TIC" for *T*otal *I*on *C*urrent normalisation of the scans/conditions (column-wise normalisation) is supported.
verbose	logical, verbose output?
what	character, specifies whether "conditions" (columns; default) or "fragments" (rows) should be summarized.
...	arguments passed to internal/other methods. replicates (that's why the default is the "Sample" column in colData).

Details

See vignette("analysis", package="topdownr") for a detailed example how to work with TopDownSet objects.

Value

An TopDownSet object.

Methods (by generic)

- aggregate(TopDownSet): Aggregate conditions/runs.

  Aggregates conditions/runs (columns) in an TopDownSet object by a user-given value (default is the "Sample" column of colData which has the same value for technical replicates). It combines intensity values and numeric metadata of the grouped conditions/runs (columns) by mean and returns a reduced TopDownSet object.

- `combine(x = TopDownSet, y = TopDownSet)`: Combine `TopDownSet` objects.

  `combine` allows to combine two or more `TopDownSet` objects. Please note that it uses the default `sampleColumns` to define technical replicates (see `readTopDownFiles()`).and the default by argument to group ion injection times for the calculation of the median time (see `updateMedianInjectionTime()`). Both could be modified after `combine` by calling `updateConditionNames()` (with modified `sampleColumns` argument) and `updateMedianInjectionTime()` (with modified by argument).

- `filterCv(TopDownSet)`: Filter by CV.

  Filtering is done by coefficient of variation across technical replicates (defined by the `by` argument). All fragments below a given `threshold` are removed. The `threshold` is the maximal allowed CV in percent (`sd/mean * 100 < threshold`).

- `filterInjectionTime(TopDownSet)`: Filter by ion injection time.

  Filtering is done by maximal allowed deviation and just the technical `keepTopN` replicates with the lowest deviation from the median ion injection time are kept.

- `filterIntensity(TopDownSet)`: Filter by intensity.

  Filtering is done by removing all fragments that are below a given (absolute/relative) intensity `threshold`.

- `filterNonReplicatedFragments(TopDownSet)`: Filter by non-replicated fragments.

  Filtering is done by removing all fragments that don't replicate across technical replicates.

- `normalize(TopDownSet)`: Normalise.

  Applies *T*otal *I*on *C*urrent normalisation to a TopDownSet object. The normalisation ist done per scans/conditions (column-wise normalisation).

- `plot(x = TopDownSet, y = missing)`: Plotting.

  Plots an TopDownSet object. The function returns a `list` of `ggplot` objects (one item per condtion). Use `pdf` or another non-interactive device to plot the list of `ggplot` objects (see example section).

- `summary(TopDownSet)`: Summary statistics.

  Returns a `matrix` with some statistics: number of fragments, total/min/first quartile/median/mean/third quartile/maximum of intensity values.

Slots

`rowViews` FragmentViews, information about fragments (name, type, sequence, mass, charge), see FragmentViews for details.

`colData` S4Vectors::DataFrame, information about the MS2 experiments and the fragmentation conditions.

`assay` Matrix::dgCMatrix, intensity values of the fragments. The rows correspond to the fragments and the columns to the condition/run. It just stores values that are different from zero.

`files character`, files that were imported.

`tolerance double`, tolerance in *ppm* that were used for matching the experimental mz values to the theoretical fragments.

`redundantMatching character`, matching strategies for redundant ion/fragment matches. See `redundantIonMatch` and `redundantFragmentMatch` in `readTopDownFiles()` for details.

`processing character`, log messages.

Coercion

'as(object, "MSnSet"): Coerce an TopDownSet object into an MSnbase::MSnSet object.

'as(object, "NCBSet"): Coerce an TopDownSet object into an NCBSet object.

Author(s)

Sebastian Gibb <mail@sebastiangibb.de>

See Also

- FragmentViews for the row view interface.
- Matrix::dgCMatrix for technical details about the intensity storage.
- `?vignette("analysis", package="topdownr")` for a full documented example of an analysis using topdownr.

Examples

```
## Example data
data(tds, package="topdownr")

tds

head(summary(tds))

# Accessing slots
rowViews(tds)
colData(tds)
head(assayData(tds))

# Accessing colData
tds$Mz
tds$FilterString

# Subsetting

# First 100 fragments
tds[1:100]

# All c fragments
tds["c"]

# Just c 152
tds["c152"]

# Condition 1 to 10
tds[, 1:10]

# Filtering
# Filter all intensities that don't have at least 10 % of the highest
# intensity per fragment.
```

```
tds <- filterIntensity(tds, threshold=0.1)

# Filter all conditions with a CV above 30 % (across technical replicates)
tds <- filterCv(tds, threshold=30)

# Filter all conditions with a large deviation in injection time
tds <- filterInjectionTime(tds, maxDeviation=log2(3), keepTopN=2)

# Filter all conditions where fragments don't replicate
tds <- filterNonReplicatedFragments(tds)

# Normalise by TIC
tds <- normalize(tds)

# Aggregate technical replicates
tds <- aggregate(tds)

head(summary(tds))

# Coercion
as(tds, "NCBSet")

if (require("MSnbase")) {
    as(tds, "MSnSet")
}
## Not run:
# plot a single condition
# pseudo-code (replace topdownset with your object)
plot(topdownset[,1])

# plot the whole object
pdf("topdown-spectra.pdf", paper="a4r", width=12)
# pseudo-code (replace topdownset with your object)
plot(topdownset)
dev.off()

## End(Not run)
```

`validMs1Settings` *List valid MS settings.*

Description

These functions list settings for MS1 or TMS2 that are supported by *Thermo's XmlMethodChanger*.

Usage

```
validMs1Settings(family = "Calcium", version = "3.2")

validTms2Settings(
```

```
  type = c("All", "TMS2", "ETD", "CID", "HCD", "UVPD"),
  family = "Calcium",
  version = "3.2"
)
```

Arguments

family	character, currently just Calcium is supported
version	character, currently 3.1, 3.2 (default), 3.3 are supported
type	character, type of activation.

Value

matrix with three columns:

- name: element name
- class: expected R class of the value
- type: MS/ActivationType, e.g. MS1/TMS2/ETD/...

Examples

```
validMs1Settings()
validTms2Settings()
validTms2Settings("TMS2")
validTms2Settings("ETD")
validTms2Settings(c("TMS2", "ETD"))
```

---

writeMethodXmls *Create Orbitrap Fusion Lumos method.xml files.*

Description

This function is used to create Orbitrap Fusion Lumos method files from a tree-like list experiment generated by e.g. createExperimentsFragmentOptimisation().

Usage

```
writeMethodXmls(exps, pattern = "method-%s.xml", verbose = interactive())
```

Arguments

exps	list, generated by e.g. createExperimentsFragmentOptimisation()
pattern	character, file name pattern for the method.xml files.
verbose	logical, verbose output?

Details

- exps: a named tree-like list object generated by e.g. createExperimentsFragmentOptimisation(). Its names are used as filename.
- pattern: The file name pattern used to name different method files. It must contain a "%s" that is replaced by the conditions defined in groupBy.

DEFUNCT options:

- ms1Settings: A list of MS1 settings. This has to be a named list. Valid MS1 settings are: c("FirstMass", "LastMass", "Microscans", "MaxITTimeInMS", "AgcTarget")
- ms2Settings: A list of MS2 settings. This has to be a named list. Valid MS2 settings are: c("ActivationType", "IsolationWindow", "EnableMultiplexIons", "EnableMSXIds", "MaxNoOfMultiplexIons", "OrbitrapResolution", "AgcTarget", "MinAgcTarget", "MaxITTimeInMS", "Microscans", "ETDReactionTime", "ETDReagentTarget", "MaximumETDReagentInjectionTime", "UseInternalCalibratedETD", "ETDSupplementalActivationEnergy", "ETDSupplementalActivation")
- groupBy: The groupBy parameter is used to split methods into different files. Valid entries are all settings that could be used in ms2Settings and "replication".
- massLabeling: The Orbitrap Fusion devices seems not to respect the start and end times of the runs given in the method.xml files. That's why it is nearly impossible to identify the run with its conditions based on the timings. If massLabeling is TRUE (default) the mass values given in mz are rounded to the first decimal place and the second to fourth decimal place is used as numeric identifier.

Author(s)

Sebastian Gibb <mail@sebastiangibb.de>, Pavel V. Shliaha <pavels@bmb.sdu.dk>

See Also

createExperimentsFragmentOptimisation()

Examples

```
ms1 <- expandMs1Conditions(FirstMass=400, LastMass=1200, Microscans=as.integer(10))

targetMz <- cbind(mz=c(560.6, 700.5, 933.7), z=rep(1, 3))
common <- list(
    OrbitrapResolution="R120K",
    IsolationWindow=1,
    MaxITTimeInMS=200,
    Microscans=as.integer(40),
    AgcTarget=c(1e5, 5e5, 1e6)
)
cid <- expandTms2Conditions(
    MassList=targetMz,
    common,
    ActivationType="CID",
    CIDCollisionEnergy=seq(7, 35, 7)
)
hcd <- expandTms2Conditions(
```

```
    MassList=targetMz,
    common,
    ActivationType="HCD",
    HCDCollisionEnergy=seq(7, 35, 7)
)
etd <- expandTms2Conditions(
    MassList=targetMz,
    common,
    ActivationType="ETD",
    ETDReagentTarget=c(1e6, 5e6, 1e7),
    ETDReactionTime=c(2.5, 5, 10, 15, 30, 50),
    ETDSupplementalActivation=c("None", "ETciD", "EThcD"),
    ETDSupplementalActivationEnergy=seq(7, 35, 7)
)
exps <- createExperimentsFragmentOptimisation(ms1=ms1, cid, hcd, etd,
    groupBy=c("AgcTarget", "replication"), nMs2perMs1=10, scanDuration=0.5,
    replications=2, randomise=TRUE
)
writeMethodXmls(exps=exps)
```

Index