Package ‘ChemmineR’

April 11, 2018

Type Package

Title Cheminformatics Toolkit for R

Version 2.30.2

Date 2017-12-12

Author Y. Eddie Cao, Kevin Horan, Tyler Backman, Thomas Girke

Maintainer Thomas Girke <thomas.girke@ucr.edu>

Description ChemmineR is a cheminformatics package for analyzing druglike small molecule data in R. Its latest version contains functions for efficient processing of large numbers of molecules, physicochemical/structural property predictions, structural similarity searching, classification and clustering of compound libraries with a wide spectrum of algorithms. In addition, it offers visualization functions for compound clustering results and chemical structures.

License Artistic-2.0

Depends R (>= 2.10.0), methods

biocViews Cheminformatics, BiomedicalInformatics, Pharmacogenetics, Pharmacogenomics, MicrotitrePlateAssay, CellBasedAssays, Visualization, Infrastructure, DataImport, Clustering, Proteomics

Imports rjson, graphics, stats, RCurl, DBI, digest, BiocGenerics, Rcpp (>= 0.11.0), ggplot2,grid,gridExtra

Suggests RSQLite, scatterplot3d, gplots, fmcsR, snow, RPostgreSQL, BiocStyle, knitr, knitcitations, knitrBootstrap, ChemmineOB (>= 1.3.8), ChemmineDrugs, png,rmarkdown

Enhances ChemmineOB

URL https://github.com/girke-lab/ChemmineR

VignetteBuilder knitr

LinkingTo Rcpp, BH

SystemRequirements GNU make

NeedsCompilation yes

R topics documented:

`addDescriptorType` *Add Descriptor Type*

Description

Add a new descriptor type to the database. Normally descriptor types are added as needed, but if you are doing a parrallel data load you must pre-load the descriptor type to prevent duplicate defintion errors.

Usage

```
addDescriptorType(conn, descriptorType)
```

Arguments

`conn`	Any database connection object.
`descriptorType`	The name of the descriptor.

Value

No return value.

Author(s)

Kevin Horan

Examples

```
## Not run:
conn =  initDb(...)
addDescriptor(conn,"fp")

## End(Not run)
```

addNewFeatures *Add New Features*

Description

Adds new features to a database without adding any data. Note that if you are loading new data anyway, it is much more efficient to use the loadSdf function and include the new features then. This function will have to read all compounds out of the database first.

Usage

```
addNewFeatures(conn, featureGenerator)
```

Arguments

`conn` A database connection object, such as is returned by `initDb`.

`featureGenerator`
A function which returns a data frame containing the new features. It may also contain features which are already in the database, these will simply be ignored. See the description of `fct` in `loadSdf` for details.

Value

No value is returned.

Author(s)

Kevin Horan

See Also

`loadSdf`

Examples

```
   #create and initialize a new SQLite database
   conn = initDb("test.db")

data(sdfsample)

#just load the data with no features or descriptors
ids=loadSdf(conn,sdfsample)
   addNewFeatures(conn, function(sdfset)
data.frame(MW = MW(sdfset),
               rings(sdfset,type="count",upper=6, arom=TRUE))
 )

unlink("test.db")
```

ap — *Return atom pair component of* AP/APset

Description

Returns atom pair component of objects of class AP or APset as list of vectors.

Usage

```
ap(x)
```

Arguments

x	Object of class AP and APset

Details

...

Value

List	with one to many of following components:
numeric	atom pairs

Author(s)

Thomas Girke

References

Chen X and Reynolds CH (2002). "Performance of similarity measures in 2D fragment-based similarity searching: comparison of structural descriptors and similarity coefficients", J Chem Inf Comput Sci.

See Also

Functions: SDF2apcmp, apset2descdb, cmp.search, cmp.similarity

Examples

```
## Instance of SDFset class
data(sdfsample)
sdfset <- sdfsample[1:50]
sdf <- sdfset[[1]]

## Compute atom pair library
ap <- sdf2ap(sdf)
(apset <- sdf2ap(sdfset))
view(apset[1:4])

## Return main components of APset object
cid(apset[1:4]) # compound IDs
```

```
ap(apset[1:4]) # atom pair descriptors

## Return atom pairs in human readable format
db.explain(apset[1])
```

AP-class *Class "AP"*

Description

Container for storing the atom pair descriptors of a single compound as numeric vector. The atom pairs are used as structural similarity measures and for compound similarity searching.

Objects from the Class

Objects can be created by calls of the form `new("AP", ...)`.

Slots

`AP`**:** Object of class `"numeric"`

Methods

ap `signature(x = "AP")`: returns atom pairs as numeric vector

coerce `signature(from = "APset", to = "AP")`: `as(apset, "AP")`

show `signature(object = "AP")`: prints summary of `AP`

Author(s)

Thomas Girke

References

Chen X and Reynolds CH (2002). "Performance of similarity measures in 2D fragment-based similarity searching: comparison of structural descriptors and similarity coefficients", J Chem Inf Comput Sci.

See Also

Related classes: SDF, SDFset, SDFstr, APset.

Functions: `SDF2apcmp`, `apset2descdb`, `cmp.search`, `cmp.similarity`

Examples

```
showClass("AP")

## Instance of SDFset class
data(sdfsample)
sdfset <- sdfsample[1:50]
sdf <- sdfsample[[1]]
```

```
## Compute atom pair library
ap <- sdf2ap(sdf)
(apset <- sdf2ap(sdfset))
view(apset[1:4])

## Return main components of APset object
cid(apset[1:4]) # compound IDs
ap(apset[1:4]) # atom pair descriptors

## Return atom pairs in human readable format
db.explain(apset[1])

## Coerce APset to other objects
apset2descdb(apset) # returns old list-style AP database
tmp <- as(apset, "list") # returns list
as(tmp, "APset") # converst list back to APset

## Compound similarity searching with APset
cmp.search(apset, apset[1], type=3, cutoff=0.2)
plot(sdfset[names(cmp.search(apset, apset[6], type=2, cutoff=0.4))])

## Identify compounds with identical AP sets
cmp.duplicated(apset, type=2)

## Structure similarity clustering
cmp.cluster(db=apset, cutoff = c(0.65, 0.5))[1:20,]
```

---

apfp *Frequent Atom Pairs*

Description

Ranked set of 4096 most frequent atom pairs observed in the compound collection from DrugBank with a MW < 1000. Their atom pairs were generated with the `sdf2ap` function. The provided data frame is sorted row-wise by atom pair frequency and only the 4096 most frequent atom pairs are included. This data set can be used as predefined atom pair selection when computing atom pair fingerprints with the `desc2fp` function.

Usage

```
data(apfp)
```

Format

Object of class `data.frame`. First column contains atom pair (AP) IDs and the second column their frequency in DrugBank compounds.

Details

Object stores 4096 most frequent atom pairs generated from DrugBank compounds.

Source

DrugBank: http://www.drugbank.ca/

References

Chen X and Reynolds CH (2002). "Performance of similarity measures in 2D fragment-based similarity searching: comparison of structural descriptors and similarity coefficients", J Chem Inf Comput Sci.

Examples

```
data(apfp)
apfp[1:4,]
```

apset *Atom pairs stored in* APset *object*

Description

Atom pairs for 100 molecules stored in `sdfsample`.

Usage

```
data(apset)
```

Format

Object of class `apset`

Details

Object stores atom pairs of 100 molecules.

Source

```
apset <- sdf2ap(sdfsample)
```

References

Chen X and Reynolds CH (2002). "Performance of similarity measures in 2D fragment-based similarity searching: comparison of structural descriptors and similarity coefficients", J Chem Inf Comput Sci.

Examples

```
data(apset)
apset[1:4]
view(apset[1:4])
```

APset-class *Class "APset"*

Description

List-like container for storing the atom pair descriptors of a many compounds as objects of class AP. This container is used for structure similarity searching of compounds.

Objects from the Class

Objects can be created by calls of the form new("APset", ...).

Slots

AP: Object of class "list"

ID: Object of class "character"

Methods

[signature(x = "APset"): subsetting of class with bracket operator

[[signature(x = "APset"): returns single component as AP object

[[<- signature(x = "APset"): replacement method for single AP component

[<- signature(x = "APset"): replacement method for several AP components

ap signature(x = "APset"): returns atom pair list from AP slot

c signature(x = "APset"): concatenates two APset containers

cid signature(x = "APset"): returns all compound identifiers from ID slot

cid<- signature(x = "APset"): replacement method for compound identifiers in ID slot

coerce signature(from = "APset", to = "AP"): as(apset, "AP")

coerce signature(from = "APset", to = "list"): as(apset, "list")

coerce signature(from = "list", to = "APset"): as(list, "APset")

length signature(x = "APset"): returns number of entries stored in object

show signature(object = "APset"): prints summary of APset

view signature(x = "APset"): prints extended summary of APset

Author(s)

Thomas Girke

References

Chen X and Reynolds CH (2002). "Performance of similarity measures in 2D fragment-based similarity searching: comparison of structural descriptors and similarity coefficients", in J Chem Inf Comput Sci.

See Also

Related classes: SDF, SDFset, SDFstr, AP, FPset, FP.

Functions: SDF2apcmp, apset2descdb, cmp.search, cmp.similarity

Examples

```
showClass("APset")

## Instance of SDFset class
data(sdfsample)
sdfset <- sdfsample[1:50]
sdf <- sdfsample[[1]]

## Compute atom pair library
ap <- sdf2ap(sdf)
(apset <- sdf2ap(sdfset))
view(apset[1:4])

## Return main components of APset object
cid(apset[1:4]) # compound IDs
ap(apset[1:4]) # atom pair descriptors

## Return atom pairs in human readable format
db.explain(apset[1])

## Coerce APset to other objects
apset2descdb(apset) # returns old list-style AP database
tmp <- as(apset, "list") # returns list
as(tmp, "APset") # converst list back to APset

## Compound similarity searching with APset
cmp.search(apset, apset[1], type=3, cutoff=0.2)
plot(sdfset[names(cmp.search(apset, apset[6], type=2, cutoff=0.4))])

## Identify compounds with identical AP sets
cmp.duplicated(apset, type=2)

## Structure similarity clustering
cmp.cluster(db=apset, cutoff = c(0.65, 0.5))[1:20,]
```

apset2descdb *APset to list-style AP database*

Description

Coerces APset to old list-style descriptor database used by search/cluster functions.

Usage

```
apset2descdb(apset)
```

Arguments

apset	Object of class apset

Details

...

Value

`list`	with following components
`descdb`	list of atom pair sets
`cids`	compound IDs
`sdfsegs`	start/end coordinates for each molecule in SD file; only populated when `cmp.parse` is used for import
`source`	path/name of SD file
`type`	import method

Author(s)

Thomas Girke

References

Chen X and Reynolds CH (2002). "Performance of similarity measures in 2D fragment-based similarity searching: comparison of structural descriptors and similarity coefficients", J Chem Inf Comput Sci.

See Also

Functions: `SDF2apcmp`, `sdf2ap`, `cmp.search`, `cmp.similarity`

Examples

```
## Instance of SDFset class
data(sdfsample)
sdfset <- sdfsample[1:50]
sdf <- sdfsample[[1]]

## Compute atom pair library
ap <- sdf2ap(sdf)
(apset <- sdf2ap(sdfset))
view(apset[1:4])

## Return main components of APset object
cid(apset[1:4]) # compound IDs
ap(apset[1:4]) # atom pair descriptors

## Return atom pairs in human readable format
db.explain(apset[1])

## Coerce APset to other objects
apset2descdb(apset) # returns old list-style AP database
tmp <- as(apset, "list") # returns list
as(tmp, "APset") # converst list back to APset

## Compound similarity searching with APset
cmp.search(apset, apset[1], type=3, cutoff=0.2)
plot(sdfset[names(cmp.search(apset, apset[6], type=2, cutoff=0.4))])

## Identify compounds with identical AP sets
cmp.duplicated(apset, type=2)
```

```
## Structure similarity clustering
cmp.cluster(db=apset, cutoff = c(0.65, 0.5))[1:20,]
```

atomblock *Return atom block*

Description

Returns atom block(s) from an object of class SDF or SDFset.

Usage

```
atomblock(x)
```

Arguments

x	object of class SDF or SDFset

Details

...

Value

matrix if SDF is provided or list of matrices if SDFset is provided

Author(s)

Thomas Girke

References

...

See Also

header, atomcount, bondblock, datablock, cid, sdfid

Examples

```
## SDF/SDFset instances
data(sdfsample)
sdfset <- sdfsample
sdf <- sdfset[[1]]

## Extract atome block
atomblock(sdf)
atomblock(sdfset[1:4])

## Replacement methods
sdfset[[1]][[2]][1,1] <- 999
sdfset[[1]]
atomblock(sdfset)[1:2] <- atomblock(sdfset)[3:4]
atomblock(sdfset[[1]]) == atomblock(sdfset[[3]])
view(sdfset[1:2])
```

atomcount *Molecular property functions*

Description

Functions to compute molecular properties: weight, formula, atom frequencies, etc.

Usage

```
atomcount(x, addH = FALSE, ...)

atomcountMA(x, ...)

MW(x, mw=atomprop, ...)

MF(x, ...)
```

Arguments

`x`	object of class `SDFset` or `SDF`
`mw`	`data.frame` with atomic weights; imported by default with data(atomprop); supports custom data sets
`addH`	'addH = TRUE' should be passed on to any of these function to add hydrogens that are often not specified in SD files
`...`	Arguments to be passed to/from other methods.

Details

...

Value

`named vector`	`MW` and `MF`
`list`	`atomcount`
`matrix`	`atomcountMA`

Author(s)

Thomas Girke

References

Standard atomic weights (2005) from: http://iupac.org/publications/pac/78/11/2051/

See Also

Functions: `datablock`, `datablocktag`

Examples

```
## Instance of SDFset class
data(sdfsample)
sdfset <- sdfsample

## Compute properties; to consider missing hydrogens, set 'addH = TRUE'
MW(sdfset[1:4], addH = FALSE)
MF(sdfset[1:4], addH = FALSE)
atomcount(sdfset[1:4], addH = FALSE)
propma <- atomcountMA(sdfset[1:4], addH = FALSE)
boxplot(propma, main="Atom Frequency")

## Example for injecting a custom matrix/data frame into the data block of an
## SDFset and then writing it to an SD file
props <- data.frame(MF=MF(sdfset), MW=MW(sdfset), atomcountMA(sdfset))
datablock(sdfset) <- props
view(sdfset[1:4])
# write.SDF(sdfset[1:4], file="sub.sdf", sig=TRUE, cid=TRUE)
```

---

atomprop *Standard atomic weights*

Description

Data frame with atom names, symbols, standard atomic weights, group number and period number.

Usage

```
data(atomprop)
```

Format

The format is a data frame with 117 rows and 6 columns.

Source

Columns 1 to 4 from: http://iupac.org/publications/pac/78/11/2051/ Columns 5 to 6 from: http://en.wikipedia.org/wiki/Li

References

Pure Appl. Chem., 2006, Vol. 78, No. 11, pp. 2051-2066

Examples

```
data(atomprop)
atomprop[1:4,]
```

atomsubset *Subset SDF/SDFset Objects by Atom Index to Obtain Substructure*

Description

Function to obtain a substructure from SDF/SDFset objects by providing a row index for the atom block in an SDF referencing the atoms of interest. The function subsets both the atom and bond block(s) accordingly.

Usage

```
atomsubset(x, atomrows, type="new", datablock = FALSE)
```

Arguments

`x`	object of class `SDFset` or `SDF`
`atomrows`	The argument `atomrows` can be assigned a numeric index referencing the atoms in the atom block of `x`. If `x` is of class `SDF`, the index needs to be provided as `vector`. If `x` is of class `SDFset`, the same number of index vectors as molecules stored in `x` need to be passed on in a list with component names identical to the component (molecule) names stored in `x`.
`type`	The argument `type="new"` assigns new atom numbers to a subsetted SDF, while `type="old"` maintains the numbering of the source SDF.
`datablock`	By default the data block(s) in `SDF/SDFset` objects are removed after atom subsetting. The setting `datablock=TRUE` will maintain the data block information in the subsetted result.

Details

...

Value

object of class `SDF` or `SDFset`

Author(s)

Thomas Girke

References

...

See Also

...

Examples

```
## Instance of SDFset class
data(sdfsample)
sdfset <- sdfsample

## Subset one or more molecules with atom index(es) to obtain substructure(s)
atomsubset(sdfset[[1]], atomrows=1:18)
indexlist <- list(1:18, 1:12)
names(indexlist) <- cid(sdfset[1:2])
atomsubset(sdfset[1:2], atomrows=indexlist)
```

`batchByIndex` *Batch by Index*

Description

When doing a select were the condition is a large number of ids it is not always possible to include them in a single SQL statement. This function will break the list of ids into chunks and allow the indexProcessor to deal with just a small number of ids.

Usage

```
batchByIndex(allIndices, indexProcessor, batchSize = 1e+05)
```

Arguments

`allIndices`	A vector of values that will be broken into batches and passed as an argument to the `indexProcessor` function.
`indexProcessor`	A function that takes one batch if indices. It is called once for each batch. The return value from this function is ignored. To accumulate results you can write to a global variable using the "«-" operator.
`batchSize`	The size of each batch. The last batch may be smaller than this value.

Value

No value is returned.

Author(s)

Kevin Horan

See Also

`parBatchByIndex`

Examples

```
## Not run:
result=NA
indices = 1:10000

#run a query on each batch of indexes, appending each result to
# "result" as we go.
batchByIndex(indices, function(indexBatch){
df = dbGetQuery(dbConnection, generateQuery(indexBatch))
result <<- if(is.na(result)) df else  rbind(result,df)
},1000)

## End(Not run)
```

bondblock *Return bond block*

Description

Returns bond block(s) from an object of class SDF or SDFset.

Usage

```
bondblock(x)
```

Arguments

x	object of class SDF or SDFset

Details

...

Value

matrix if SDF is provided or list of matrices if SDFset is provided

Author(s)

Thomas Girke

References

...

See Also

header, atomcount, atomblock, datablock, cid, sdfid

Examples

```
## SDF/SDFset instances
data(sdfsample)
sdfset <- sdfsample
sdf <- sdfset[[1]]

## Extract bond block
bondblock(sdf)
bondblock(sdfset[1:4])

## Replacement methods
sdfset[[1]][[3]][1,1] <- 999
sdfset[[1]]
bondblock(sdfset)[1:2] <- bondblock(sdfset)[3:4]
bondblock(sdfset[[1]]) == bondblock(sdfset[[3]])
view(sdfset[1:2])
```

---

bonds *Bonds, charges and missing hydrogens*

Description

Returns information about bonds, charges and missing hydrogens in `SDF` and `SDFset` objects.

Usage

```
bonds(x, type = "bonds")
```

Arguments

`x`	`SDF` or `SDFset` containers
`type`	If `type="bonds"` (default), a `data.frame` is returned with columns: `atom` (atom labels), `Nbondcount` (observed bond count), `Nbondrule` (bond count according to position in periodic table) and `charge` (charge of each atom). If `type="charge"`, all charged atoms are returned and if `type="addNH"`, the number of missing hydrogens are returned for each molecule.

Details

It is used by many other functions (e.g. `MW`, `MF`, `atomcount`, `atomcuntMA` and `plot`) to correct for missing hydrogens that are often not specified in SD files.

Value

If `x` is of class `SDF`, then a single `data.frame` or `vector` is returned. If `x` is of class `SDFset`, then a `list` of `data.frames` or `vecotors` is returned that has the same length and order as `x`.

Author(s)

Thomas Girke

References

...

See Also

Functions: conMA

Class: SDF and SDFset

Examples

```
## Instances of SDFset class
data(sdfsample)
sdfset <- sdfsample

## Returns data frames with bonds and charges
bonds(sdfset[1:2], type="bonds")

## Returns charged atoms in each molecule
bonds(sdfset[1:2], type="charge")

## Returns the number of missing hydrogens in each molecule
bonds(sdfset[1:2], type="addNH")
```

---

browseJob *Open ChemMine Tools Job in Web Browser*

Description

Launches a web browser to view the results of a ChemMine Tools web job with an interactive online viewer. Note that this reassigns the job to the current logged in user within the browser, so it becomes no longer accessible by the result and status functions. Any results should be saved within R before launching a browser.

Usage

```
browseJob(object)
```

Arguments

object	A jobToken job as returned by the function launchCMTool

Value

Returns an URL string which can be used to access the job results. The function also attempts to open the url with the browseURL function. As this URL can only be used once, the returned string is only useful if the browseURL function fails to open a browser.

Author(s)

Tyler William H Backman

References

See ChemMine Tools at http://chemmine.ucr.edu.

See Also

Functions: toolDetails, listCMTools, launchCMTool, result, status

Examples

```
## Not run:
## list available tools
listCMTools()

## get detailed instructions on using a tool
toolDetails("Fingerprint Search")

## download compound 2244 from PubChem
job1 <- launchCMTool("pubchemID2SDF", 2244)

## check job status and download result
status(job1)
result1 <- result(job1)

## open job in web browser
browseJob(job1)

## End(Not run)
```

---

bufferLines *Buffer File Input*

Description

Buffer the input of files to increase efficiency

Usage

```
bufferLines(fh, batchSize, lineProcessor)
```

Arguments

fh	file handle
batchSize	How many lines to read in each batch
lineProcessor	Each batch of lines will be passed to this function for processing

Value

No return value

Author(s)

Kevin Horan

Examples

```
## Not run:
fh = file("filename")
bufferLines(fh,100,function(lines) {
message("found ",length(lines)," lines")
})

## End(Not run)
```

---

bufferResultSet *Buffer Query Results*

Description

Allow query results to be processed in batches for efficiency.

Usage

```
bufferResultSet(rs, rsProcessor, batchSize = 1000,closeRS=FALSE)
```

Arguments

rs	A DBIResult object, usually from dbSendQuery.
rsProcessor	Each batch will be passed as a data frame to this function for processing.
batchSize	The number of rows to read in each batch
closeRS	Should the result set be closed by this function when it is done?

Value

No value.

Author(s)

Kevin Horan

Examples

```
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (rs, rsProcessor, batchSize = 1000)
{
    while (TRUE) {
        chunk = fetch(rs, n = batchSize)
        if (dim(chunk)[1] == 0)
            break
        rsProcessor(chunk)
    }
  }
```

byCluster *By Cluster*

Description

Re-organize a vector valued clustering into an list which groups cluster members together

Usage

```
byCluster(clustering, excludeSingletons = TRUE)
```

Arguments

`clustering`	A named vector in which the names are cluster members and the values are cluster labels. This is format output by jarvisPatrick.
`excludeSingletons`	If true only clusters with more than 1 member will be in the output, otherwise all clusters will be used.

Value

A list with a slot for each cluster. Each slot of the list is a vector containing the cluster members.

Author(s)

Kevin Horan

See Also

`jarvisPatrick`

Examples

```
data(apset)
cl = jarvisPatrick(nearestNeighbors(apset,cutoff=0.6),k=2)
print(byCluster(cl))
```

canonicalize *Canonicalize*

Description

Canonicalizes the atom numbering of a compound. The implimentation of this function is in Open Babel and requires the ChemmineOB package to function.

Usage

```
canonicalize(sdf)
```

Arguments

`sdf`	Any sdfset object.

Value

A new SDFset in which all compounds have been canonicalized

Author(s)

Kevin Horan

References

http://openbabel.org/api/2.3/canonical_code_algorithm.shtml

See Also

`canonicalNumbering`

Examples

```
data(sdfsample)
canonicalSdf = canonicalize(sdfsample[1])
```

`canonicalNumbering` *Canonical Numbering*

Description

Computes a re-arrangement required to transform the atom numbering of the given compound into the canonical atom numbering. This function uses the OBGraphSym and CanonicalLabels classes of Open Babel to compute the re-arrangement.

Usage

```
canonicalNumbering(sdf)
```

Arguments

`sdf`	Any sdfset object.

Value

A list of vectors of index values. Each item in the list corresponds to one of the given compounds. The values of a list item are the re-arrangement of the atoms. For example, if the value in item 1, column 1 is 25, that means that atom number 1 in the original compound should become atom number 25 in the canonical version of that compound.

Author(s)

Kevin Horan

References

http://openbabel.org/api/2.3/canonical_code_algorithm.shtml

See Also

canonicalize

Examples

```
data(sdfsample)
labels = canonicalNumbering(sdfsample[1])
```

---

cid *Return compound IDs*

Description

Returns the compound identifiers from the ID slot of an SDFset object.

Usage

```
cid(x)
```

Arguments

x	object of class SDFset or APset

Details

...

Value

character vector

Author(s)

Thomas Girke

References

...

See Also

atomblock, atomcount, bondblock, datablock, header, sdfid

Examples

```
## SDFset/APset instances
data(sdfsample)
sdfset <- sdfsample
apset <- sdf2ap(sdfset[1:4])

## Extract compound IDs from SDFset/APset
cid(sdfset[1:4])
cid(apset[1:4])

## Extract IDs defined in SD file
sdfid(sdfset[1:4])

## Assigning compound IDs and keeping them unique
unique_ids <- makeUnique(sdfid(sdfset))
cid(sdfset) <- unique_ids
cid(sdfset[1:4])

## Replacement Method
cid(sdfset) <- as.character(1:100)
```

cluster.sizestat *generate statistics on sizes of clusters*

Description

'cluster.sizestat' is used to do simple statistics on sizes of clusters generated by 'cmp.cluster'. It will return a dataframe which maps a cluster size to the number of clusters with that size. It is often used along with 'cluster.visualize'.

Usage

```
cluster.sizestat(cls, cluster.result=1)
```

Arguments

cls	The clustering result returned by 'cmp.cluster'
cluster.result	If multiple cutoff values are used in clustering process, this argument tells which cutoff value is to be considered here.

Details

'cluster.sizestat' depends on the format that is returned by 'cmp.cluster' - it will treat the first column as the indecies, and the second column as the cluster sizes of effective clustering. Because of this, when multiple cutoffs are used when 'cmp.cluster' is called, 'cluster.sizestat' will only consider the clustering result of the first cutoff. If you want to work on an alternative cutoff, you have to manually reorder/remove columns.

Value

Returns a data frame of two columns.

`cluster size` This column lists cluster sizes

`count` This column lists number of clusters of a cluster size

Author(s)

Y. Eddie Cao

See Also

`cmp.cluster`, `cluster.visualize`

Examples

```
## Load sample SD file
# data(sdfsample); sdfset <- sdfsample

## Generate atom pair descriptor database for searching
# apset <- sdf2ap(sdfset)

## Loads same atom pair sample data set provided by library
data(apset)

## Binning clustering using variable similarity cutoffs.
cluster <- cmp.cluster(db=apset, cutoff = c(0.65, 0.5))

## Statistics on sizes of clusters
cluster.sizestat(cluster[,c(1,2,3)])
cluster.sizestat(cluster[,c(1,4,5)])
```

`cluster.visualize` *visualize clustering result using multi-dimensional scaling*

Description

'cluster.visualize' takes clustering result returned by 'cmp.cluster' and generate multi-dimensional scaling plot for visualization purpose.

Usage

```
cluster.visualize(db, cls, size.cutoff, distmat=NULL, color.vector=NULL, non.interactive="", clus
```

Arguments

`db` The desciptor database, in the format returned by 'cmp.parse'.

`cls` The clustering result returned by 'cmp.cluster'.

`size.cutoff` The cutoff size for clusters considered in this visualization. Clusters of size smaller than the cutoff will not be considered.

`distmat` A distance matrix that corresponds to the 'db'. If not provided, it will be computed on-the-fly in an efficient manner.

`color.vector` Colors to be used in the plot. If the number of colors in the vector is not enough for the plot, colors will be reused. If not provided, color will be generated and randomly sampled from 'rainbow'.

`non.interactive` If provided, will enable the non-interactive mode, and the plot will be in an eps file named after this value.

`cluster.result` Used to select the clustering result if multiple clustering results are present in 'cls'.

`dimensions` Dimensionality to be used in visualization. See details.

`quiet` Whether to supress the progress bar.

`highlight.compounds` A vector of compound IDs, corresponding to compounds to be highlighted in the plot. A highlighted compound is represented as a filled circle.

`highlight.color` Color used for highlighted compounds. If not set, a highlighted compounds will have the same color as that used for other compounds in the same cluster.

`...` Further arguments will be passed to 'cmp.similarity' to calculate similarity matrix.

Details

'cluster.visualize' internally calls the 'cmdscale' function to generate a set of points in 2-D for the compounds in selected clusters. Note that for compounds in clusters smaller than the cutoff size, they will not be considered in this calculation - their entries in 'distmat' will be discarded if 'distmat' is provided, and distances involving them will not be computed if 'distmat' is not provided.

To determine the value for 'size.cutoff', you can use 'cluster.sizestat' to see the size distribution of clusters.

Because 'cmp.cluster' function allows you to perform multiple clustering processes simultaneously with different cutoff values, the 'cls' parameter may point to a data frame containing multiple clustering results. The user can use 'cluster.result' to specify which result to use. By default, this is set to 1, and the first clustering result will be used in visualization. Whatever the value is, in interactive mode (described below), all clustering result will be displayed when a compound is selected in the interactive plot.

If the colors provided in 'color.vector' are not enough to distinguish clusters by colors, the function will silently reuse the colors, resulting multiple clusters colored in the same color. We suggest you use 'cluster.sizestat' to see how many clusters will be selected using your 'size.cutoff', or simply provide no 'color.vector'.

If 'non.interative' is not set, the final plot is interactive. You will be able to select points by clicking them. When you click on any point, information about the compound represented by that point will be displayed. This includes the cluster ID, cluster size, compound index in the SDF and compound name if any. You can then perform another selection. To exit this process, right click on X11 device or press ESC in non-X11 device (Quartz and Windows).

By default, 'dimensions' is set to 2, and the built-in 'plot' function will be used for plotting. If you need to do 3-Dimensional plotting, set 'dimensions' to 3, and pass the returned value to 3D plot utilities, such as 'scatterplot3d' or 'rggobi'. This package does not perform 3D plot on its own.

Value

This function returns a data frame of MDS coordinates and clustering result. This value can be passed to 3D plot utilities such as 'scatterplot3d' and 'rggobi'.

The last column of the output gives whether the compounds have been clicked in the interactive mode.

Author(s)

Y. Eddie Cao

See Also

`cmp.parse`, `cmp.cluster`, `cluster.sizestat`

Examples

```
## Load sample SD file
# data(sdfsample); sdfset <- sdfsample

## Generate atom pair descriptor database for searching
# apset <- sdf2ap(sdfset)

## Loads same atom pair sample data set provided by library
data(apset)
db <- apset

## cluster db with 2 cutoffs
clusters <- cmp.cluster(db, cutoff=c(0.5, 0.4))

## Return size stats
sizestat <- cluster.sizestat(clusters)

## Visualize results, using a cutoff of 3, write to file 'test.eps'
coord <- cluster.visualize(db, clusters, 2, non.interactive="test.eps")

## Not run:
## visualize it in interactive mode, using a cutoff of 3 and the 2nd clustering result
coord <- cluster.visualize(db, clusters, cluster.result=2, 3)

## 3D visualization with scatterplot3d
coord <- cluster.visualize(db, clusters, 3, dimensions=3)
library(scatterplot3d)
scatterplot3d(coord)


## End(Not run)
```

`cmp.cluster` *cluster compounds using a descriptor database*

Description

'cmp.cluster' uses structural compound descriptors and clusters the compounds based on their pairwise distances. `cmp.cluster` uses single linkage to measure distance between clusters when it merges clusters. It accepts both a single cutoff and a cutoff vector. By using a cutoff vector, it can generate results similar to hierarchical clustering after tree cutting.

Usage

```
cmp.cluster(db, cutoff, is.similarity = TRUE, save.distances = FALSE,
        use.distances = NULL, quiet = FALSE, ...)
```

Arguments

db	The desciptor database, in the format returned by 'cmp.parse'.
cutoff	The clustering cutoff. Can be a single value or a vector. The cutoff gives the maximum distance between two compounds in order to group them in the same cluster.
is.similarity	Set when the cutoff supplied is a similarity cutoff. This cutoff is the minimum similarity value between two compounds such that they will be grouped in the same cluster.
save.distances	whether to save distance for future clustering. See details below.
use.distances	Supply pre-computed distance matrix.
quiet	Whether to suppress the progress information.
...	Further arguments to be passed to `cmp.similarity`.

Details

`cmp.cluster` will compute distances on the fly if `use.distances` is not set. Furthermore, if `save.distances` is not set, the distance values computed will never be stored and any distance between two compounds is guaranteed not to be computed twice. Using this method, `cmp.cluster` can deal with large databases when a distance matrix in memory is not feasible. The speed of the clustering function should be slowed when using a transient distance calculation.

When `save.distances` is set, `cmp.cluster` will be forced to compute the distance matrix and save it in memory before the clustering. This is useful when additional clusterings are required in the future without re-computed the distance matrix. Set `save.distances` to TRUE if you only want to force the clustering to use this 2-step approach; otherwise, set it to the filename under which you want the distance matrix to be saved. After you save it, when you need to reuse the distance matrix, you can 'load' it, and supply it to `cmp.cluster` via the `use.distances` argument.

`cmp.cluster` supports a vector of several cutoffs. When you have multiple cutoffs, `cmp.cluster` still guarantees that pairwise distances will never be recomputed, and no copy of distances is kept in memory. It is guaranteed to be as fast as calling `cmp.cluster` with a single cutoff that results in the longest processing time, plus some small overhead linear in processing time.

Value

Returns a `data.frame`. Besides a variable giving compound ID, each of the other variables in the data frame will either give the cluster IDs of compounds under some clustering cutoff, or the size of clusters that the compounds belong to. When N cutoffs are given, in total 2*N+1 variables will be generated, with N of them giving the cluster ID of each compound under each of the N cutoffs, and the other N of them giving the cluster size under each of the N cutoffs. The rows are sorted by cluster sizes.

Author(s)

Y. Eddie Cao, Li-Chang Cheng

See Also

cmp.parse1, cmp.parse, cmp.search, cmp.similarity

Examples

```
## Load sample SD file
# data(sdfsample); sdfset <- sdfsample

## Generate atom pair descriptor database for searching
# apset <- sdf2ap(sdfset)

## Loads atom pair and atom pair fingerprint samples provided by library
data(apset)
db <- apset
fpset <- desc2fp(apset)

## Clustering of 'APset' object with multiple cutoffs
clusters <- cmp.cluster(db=apset, cutoff=c(0.5, 0.85))

## Clustering of 'FPset' object with multiple cutoffs. This method allows to call
## various similarity methods provided by the fpSim function.
clusters2 <- cmp.cluster(fpset, cutoff=c(0.5, 0.7), method="Tversky")

## Saves the distance matrix before clustering:
clusters <- cmp.cluster(db, cutoff=0.65, save.distances="distmat.rda")
# Later one reload the matrix and pass it the clustering function.
load("distmat.rda")
clusters <- cmp.cluster(db, cutoff=0.60, use.distances=distmat)
```

cmp.duplicated *quickly detect compound duplication in a descriptor database*

Description

'cmp.duplicated' detects duplicated compounds from a descriptor database generated by 'cmp.parse'. Two compounds are said to duplicate each other when their descriptors are the same.

Usage

```
cmp.duplicated(db, sort = FALSE, type=1)
```

Arguments

db	The desciptor database, in the format returned by 'cmp.parse'.
sort	Whether to sort the descriptors for a compound. See details.
type	Returns results as vector (type=1) or data frame (type=2).

Details

'cmp.duplicated' will take the descriptors in the descriptor database, concatenate all descriptors for the same compound into a string, and use this string as the identification of a compound. If two compounds share the same identification string, they are said to duplicate each other.

'cmp.duplicated' assume the the database passed in as argument to follow the format generated by 'cmp.parse'. That is, 'db' is a list, 'db$descdb' is a list, and each entry of 'db$descdb' is an array of numeric values that give descriptors for one compound.

By default, 'cmp.duplicated' will assume the descriptors for a compound is already sorted. That is each entry in 'db\$descdb' is a sorted array. This is true for database generated by 'cmp.parse'. If you generate the database using some other tools, you might want to enable sorting.

Value

Returns a logic array, telling whether a compound in the database is a duplication of a compound appearing before this one. For example, if the i-th element of the array is TRUE, it means that the i-th compound in the database is a duplication of a compound listed before this compound in the database.

The returned array can be used to remove duplication. Simply use it to index the descriptor database.

If you are interested in what compound is duplicated, you can do a search in the database with cutoff set to 1.

Author(s)

Y. Eddie Cao

See Also

`cmp.parse`, `cmp.search`

Examples

```
## Load sample SD file
# data(sdfsample); sdfset <- sdfsample

## Generate atom pair descriptor database for searching
# apset <- sdf2ap(sdfset)

## Loads same atom pair sample data set provided by library
data(apset)
db <- apset

## Manually create a duplication (here compound 1 and 10)
db[10] <- db[1]

## Find duplication
dup <- cmp.duplicated(db)
dup
cid(db[dup])

## Remove all duplications
db <- db[!dup]
```

cmp.parse *Parse an SDF file and compute descriptors for all compounds*

Description

'cmp.parse' will take a SDF file, parse all the compounds encoded, compute their atom-pair descriptors, and return the descriptors as a list. The list contains two names, 'descdb' and 'cids'. 'descdb' is a vector of descriptors, and 'cids' is a list of names of compounds found in the SDF file. The returned list is usually used to a database, against which similarity search can be performed using the 'search' function. These two functions will parse all compounds in the SDF file. To parse a single compound, use 'cmp.parse1' instead.

Usage

```
cmp.parse(filename)
```

Arguments

`filename`	The file name of the SDF file

Details

The 'filename' can be a local file or an URL. It is interactive, and will display the parsing progress. Since the parsing will also compute of atom-pair descriptors, it is time consuming. You will be reminded to save the parsing result for future use at the end of parsing.

'type' is either set to the default value 'normal' or 'file-backed'. When set to 'file-backed', the parsing work will be delegated to a separate package called 'ChemmineRpp', and the database will be stored in a file instead of in the primary memory. Therefore, 'file-backed' mode can handle larger compound libraries. In 'file-backed' mode, 'dbname' will be used to name the database file. A suffix '.cdb' will be appended to the given name.

The type of the database is transparent to other part of the package. For example, calling 'cmp.search' against a database in 'file-backed' mode will cause the package to load the descriptors from the database file progressively.

Value

Return a list that can be used as the database against which similarity search can be performed. The 'search' and 'cmp.cluster' functions both expect a database returned by 'cmp.parse'.

`descdb`	A vector containing the descriptors for all the compounds.
`cids`	Compound ID information found in the SDF file. It is the first line of SDF of a compound.

Author(s)

Y. Eddie Cao, Li-Chang Cheng

References

Chen X and Reynolds CH (2002). "Performance of similarity measures in 2D fragment-based similarity searching: comparison of structural descriptors and similarity coefficients", in *J Chem Inf Comput Sci*.

See Also

cmp.parse1, cmp.search, cmp.cluster, cmp.similarity

Examples

```
## Load sample SD file
# data(sdfsample); sdfset <- sdfsample

## Generate atom pair descriptor database for searching
# apset <- sdf2ap(sdfset)

## Loads same atom pair sample data set provided by library
data(apset)
db <- apset
# (optinally) save the db for future use
save(db, file="db.rda", compress=TRUE)
# ...
# later, in a separate session, you can load it back:
load("db.rda")
```

---

cmp.parse1 *Parsing an SDF file and calculate the descriptor for one compound*

Description

Read SDF information from an SDF file or connection, parse the first compound, and calculate the descriptor for that compound. The returned descriptor can be added to database returned by 'cmp.parse' or be used as the query structure when calling 'search'. This function will only parse one compound and return only the descriptor. To parse all compounds in an SDF file, use 'cmp.parse'.

Usage

```
cmp.parse1(filename)
```

Arguments

filename	The file name of the SDF file or a URL or a connection.

Details

'cmp.parse1' can take a file name or a URL or a connection. When a connection is used, the current line must be the first line of SDF of the compound to be parsed. 'cmp.parse1' will skip the header and parse from the 4th line. Therefore, the compound ID information will be skipped. After the parsing is done, if 'filename' is a connection, it will then point to the line after the connection table of SDF. You can use some other procedure to parse the annotation block.

Value

Return the descriptor, which is encoded as a vector.

Author(s)

Y. Eddie Cao, Li-Chang Cheng

References

Chen X and Reynolds CH (2002). "Performance of similarity measures in 2D fragment-based similarity searching: comparison of structural descriptors and similarity coefficients", in *J Chem Inf Comput Sci*.

See Also

`cmp.parse`, `cmp.search`, `cmp.cluster`, `cmp.similarity`

Examples

```
# load an SDF file from web and parse it
## Not run: structure <- cmp.parse1("http://bioweb.ucr.edu/ChemMineV2/compound/Aurora/b32:NNQS2MBRHAZTI===/s
```

`cmp.search` *Search a descriptor database for compounds similar to query compound*

Description

Given descriptor of a query compound and a database of compound descriptors, search for compounds that are similar to the query compound. User can limit the output by supplying a cutoff similarity score or a cutoff that limits the number of returned compounds. The function can also return the scores together with the compounds.

Usage

```
cmp.search(db, query, type=1, cutoff = 0.5, return.score = FALSE, quiet = FALSE,
 mode = 1,visualize = FALSE, visualize.browse = TRUE, visualize.query = NULL)
```

Arguments

`db` The compound descriptor database returned by 'cmp.parse'.

`query` The query descriptor, which is usually returned by 'cmp.parse1'.

`type` Returns results in form of position indices (type=1), named vector with compound IDs (type=2) or data frame (type=3).

`cutoff` The cutoff similarity (when cutoff <= 1) or the number of maximum compounds to be returned (when cutoff > 1).

`return.score` Whether to return similarity scores. If set to TRUE, a data frame will be returned; otherwise, only the compounds' indices in the database will be returned in the order of decreasing scores.

`quiet` Whether to disable progress information.

`mode` Mode used when computing similarity scores. This value is passed to 'cmp.similarity'.

`visualize`

`visualize.browse`

`visualize.query`

Details

'cmp.search' will go through all the compound descriptors in the database and calculate the similarity between the query compound and compounds in the database. When cutoff similarity score is set, compounds having a similarity score higher than the cutoff will be returned. When maximum number of compounds to return is set to N via 'cutoff', the compounds having the highest N similarity scores will be returned.

Value

When 'return.score' is set to FALSE, a vector of matching compounds' indices in the database will be returned. Otherwise, a data frame will be returned:

`ids` The indices of matching compounds in the database.

`scores` The similarity scores between the matching compounds and the query compound

Author(s)

Y. Eddie Cao, Li-Chang Cheng

References

Chen X and Reynolds CH (2002). "Performance of similarity measures in 2D fragment-based similarity searching: comparison of structural descriptors and similarity coefficients", in *J Chem Inf Comput Sci*.

See Also

`cmp.parse1`, `cmp.parse`, `cmp.search`, `cmp.cluster`, `cmp.similarity`, `sdf.visualize`

Examples

```
## Load sample SD file
# data(sdfsample); sdfset <- sdfsample

## Generate atom pair descriptor database for searching
# apset <- sdf2ap(sdfset)

## Loads same atom pair sample data set provided by library
data(apset)
db <- apset
query <- db[1]

## Ooptinally, save the db for future use
save(db, file="db.rda", compress=TRUE)

## Search for similar compounds using similarity cutoff
```

```
cmp.search(db, query, cutoff=0.2, type=1) # returns index
cmp.search(db, query, cutoff=0.2, type=2) # returns named vector
cmp.search(db, query, cutoff=0.2, type=3) # returns data frame

## in the next session, you may use load a saved db and do the search:
load("db.rda")
cmp.search(db, query, cutoff=3)
## you may also use the loaded db to do clustering:
cmp.cluster(db, cutoff=0.35)
```

cmp.similarity *Compute similarity between two compounds using their descriptors*

Description

Given descriptors for two compounds, 'cmp.similarity' returns the similarity measure between the two compounds.

Usage

```
cmp.similarity(a, b, mode = 1, worst = 0)
```

Arguments

a	Descriptor of the first compound.
b	Descriptor of the second compound.
mode	Mode used when computing the distance. See details below.
worst	The worst value you are expecting. If 'cmp.similarity' finds the upper bound of similarity is worse than it, it will return a 0 and potentially save some computation.

Details

'cmp.similarity' uses descriptor information generated by 'cmp.parse' and 'cmp.parse1'. Basically, a descriptor is a vector of numbers. The vector actually reprsents the set of descriptors of structural fragment. Similarity measurement uses Tanimoto coefficient.

'cmp.similarity' supports 3 different modes. In mode 1, normal Tanimoto coefficient is used. In mode 2, it uses the size of descriptor intersection over the size of the smaller descriptor, mainly to deal with compounds that vary a lot in size. In mode 3, it is similar to mode 2, except that it raises the similarity to the power 3 to penalize small values. When mode is 0, 'cmp.similarity' will select mode 1 or mode 3, based on the size differences between the two descriptors.

When 'cmp.similarity' is used in searching compounds with a threshold similarity value, or in clustering with a cutoff distance, the threshold similarity and cutoff distance can be used to decide a 'worse' value. 'cmp.similarity' can compute an upper bound of similarity easier, and by comparing this upper bound to the 'worst' value, it can potentially skip the real computation if it finds the similarity will be below the 'worst' value and will be useless to the caller.

Value

Return a numeric value between 0 and 1 which gives the similarity between the two compounds.

Author(s)

Y. Eddie Cao, Li-Chang Cheng

References

Chen X and Reynolds CH (2002). "Performance of similarity measures in 2D fragment-based similarity searching: comparison of structural descriptors and similarity coefficients", in *J Chem Inf Comput Sci*.

Peter Willett (1998). "Chemical Similarity Searching", in *J. Chem. Inf. Comput. Sci*.

See Also

`cmp.parse1`, `cmp.parse`, `cmp.search`, `cmp.cluster`

Examples

```
## Load sample SD file
# data(sdfsample); sdfset <- sdfsample

## Generate atom pair descriptor database for searching
# apset <- sdf2ap(sdfset)

## Loads same atom pair sample data set provided by library
data(apset)

## Compute similarities among two compounds
cmp.similarity(apset[1], apset[2])

## Search apset database with a query compound
cmp.search(apset, apset[1], type=3, cutoff = 0.3)
```

---

conMA *Bond Matrices*

Description

Creates a bond matrix from `SDF` and `SDFset` objects. The matrix contains the atom labels in the row and column titles and the bond types are given in the data part as follows: 0 is no connection, 1 is a single bond, 2 is a double bond and 3 is a triple bond.

Usage

```
conMA(x, exclude = "none")
```

Arguments

`x` `SDF` or `SDFset` containers

`exclude` if `exclude="none"`, then all atoms will be considered in the resulting connection table; if `exclude=c("H")`, then the H atoms will be excluded. Any number of atom labels to be excluded can be passed on to this argument in form of a `character` vector.

Details

...

Value

If `x` is of class `SDF`, then a single bond `matrix` is returned. If `x` is of class `SDFset`, then a `list` of matrices is returned that has the same length as `x`.

Author(s)

Thomas Girke

References

...

See Also

Functions: `bonds`

Class: `SDF` and `SDFset`

Examples

```
## Instances of SDFset class
data(sdfsample)
sdfset <- sdfsample

## Create bond matrix for first two molecules in sdfset
conMA(sdfset[1:2], exclude=c("H"))

## Return bond matrix for first molecule and plot its structure with atom numbering
conMA(sdfset[[1]], exclude=c("H"))
plot(sdfset[1], atomnum = TRUE, noHbonds=FALSE , no_print_atoms = "", atomcex=0.8)

## Return number of non-H bonds for each atom
rowSums(conMA(sdfset[[1]], exclude=c("H")))
```

`connections` *Database Connections*

Description

Get a connection to one of the pre-build compound databases.

Usage

```
DUD()

DrugBank()
```

Value

A connection object to the ether the DUD or DrugBank database. This object must be passed to other functions which make use of the connection.

Author(s)

Kevin Horan

Examples

```
dbConn = DrugBank()
```

datablock *Return data block*

Description

Returns data block(s) from an object of class SDF or SDFset.

Usage

```
datablock(x)

datablocktag(x, tag)
```

Arguments

`x`	object of class `SDF` or `SDFset`
`tag`	`numeric` position (index) or `character` name of entry in data block vector

Details

...

Value

named `character` vector if `SDF` is provided or `list` of named `character` vectors if `SDFset` is provided

Author(s)

Thomas Girke

References

...

See Also

`atomblock`, `atomcount`, `bondblock`, `header`, `cid`, `sdfid`

Examples

```
## SDF/SDFset instances
data(sdfsample)
sdfset <- sdfsample
sdf <- sdfset[[1]]

## Extract data block
datablock(sdf)
datablock(sdfset[1:4])
datablocktag(sdfset, tag="PUBCHEM_OPENEYE_CAN_SMILES")

## Replacement methods
sdfset[[1]][[1]][1] <- "test"
sdfset[[1]]
datablock(sdfset)[1] <- datablock(sdfset[2])
view(sdfset[1:2])

## Example for injecting a custom matrix/data frame into the data block of an
## SDFset and then writing it to an SD file
props <- data.frame(MF=MF(sdfset), MW=MW(sdfset), atomcountMA(sdfset))
datablock(sdfset) <- props
view(sdfset[1:4])
# write.SDF(sdfset[1:4], file="sub.sdf", sig=TRUE, cid=TRUE)
```

---

datablock2ma *SDF data blocks to matrix*

Description

Convert data blocks in SDFset to character matrix with datablock2ma, then store its numeric columns as numeric matrix and its character columns as character matrix.

Usage

```
datablock2ma(datablocklist, cleanup = " \\(.*", ...)

splitNumChar(blockmatrix)
```

Arguments

datablocklist	list of data block vectors; can be created with datablock(sdfset)
blockmatrix	matrix returned by datablock2ma
cleanup	character pattern to be used to clean up the name fields of the data block vectors; the exact pattern matches are replaced by nothing (deleted).
...	option to pass on additional arguments

Details

...

Value

`datablock2ma`	`character matrix`
`splitNumChar`	`list` with two components, a numeric matrix and a character matrix

Author(s)

Thomas Girke

References

...

See Also

Classes: `SDFset`

Examples

```
## SDFset instance
data(sdfsample)
sdfset <- sdfsample

# Convert data block to matrix
blockmatrix <- datablock2ma(datablocklist=datablock(sdfset))
blockmatrix[1:4, 1:4]

# Split matrix to numeric matrix and character matrix
numchar <- splitNumChar(blockmatrix=blockmatrix)
names(numchar)
numchar[[1]][1:4,]
numchar[[2]][1:4,]
```

`db.explain` *Explain an atom-pair descriptor or an array of atom-pair descriptors*

Description

'db.explain' will take an atom-pair descriptor in numeric or a set of such descriptors, and interpret what they represent in a more human readable way.

Usage

```
db.explain(desc)
```

Arguments

`desc`	The descriptor or the array/vector of descriptors

Details

'desc' can be a single numeric giving a single descriptor or can be any container data type, such as vector or array, such that 'length(desc)' returns 2 or larger.

Value

Return a character vector describing the descriptors.

See Also

`cmp.parse`

Examples

```
## Load sample SD file
# data(sdfsample); sdfset <- sdfsample

## Generate atom pair descriptor database for searching
# apset <- sdf2ap(sdfset)

## Loads same atom pair sample data set provided by library
data(apset)
db <- apset

## Return atom pairs of first compound in human readable format
db.explain(db[1])
```

`db.subset` *Subset a descriptor database and return a sub-database for the selected compounds*

Description

'db.subset' will take a descriptor database generated by 'cmp.parse' and an array of indecies, and return a new database for compounds corresponding to these indecies. The returned value is a descriptor database as returned by the `cmp.parse` function.

Usage

```
db.subset(db, cmps)
```

Arguments

`db`	The database generated by 'cmp.parse'
`cmps`	An array of indecies that correspond to a set of selected compounds from the database

Details

'db.subset' creates a sub-database from 'db' by only including infomration that is relevant to compounds indexed by 'cmps'.

Value

Return a descriptor database for the selected compounds. The format of the database is compatible with the one returned by `cmp.parse`.

See Also

`cmp.parse`, `sdf.subset`

Examples

```
## Note: this functionality has become obsolete since the introduction of the
## 'apset' S4 class.

## Load sample SD file
# data(sdfsample); sdfset <- sdfsample

## Generate atom pair descriptor database for searching
# apset <- sdf2ap(sdfset)

## Loads same atom pair sample data set provided by library
data(apset)
db <- apset
olddb <- apset2descdb(db)

## Create a sub-database for the 1st and 2nd compound in that SDF
db_sub <- db.subset(olddb, c(1, 2))
```

`dbTransaction` *DB Transaction*

Description

Run any db statements inside a transaction. If any error is raised the transaction will be rolled back, otherwise it will be committed at the end.

Usage

```
dbTransaction(conn, expr)
```

Arguments

`conn`	A database connection object, such as is returned by `initDb`.
`expr`	Any block of code.

Value

The value of the given block of code will be returned upon successfully commiting the transaction. Otherwise an error will be raised.

Author(s)

Kevin Horan

Examples

```
   conn = initDb("test15.db")
dbTransaction(conn,{
# any db code here
})
```

desc2fp *Fingerprints from descriptor vectors*

Description

Generates fingerprints from descriptor vectors such as atom pairs stored in APset or list containers. The obtained fingerprints can be used for structure similarity comparisons, searching and clustering. Due to their compact size, computations on fingerprints are often more time and memory efficient than on their much more complex atom pair counterparts.

Usage

```
desc2fp(x, descnames=1024, type = "FPset")
```

Arguments

x	Object of classe APset or list of vectors
descnames	Descriptor set to consider for fingerprint encoding. If a single value from 1-4096 is provided then the function uses the corresponding number of the most frequent atom pairs stored in the apfp data set provided by the package. Alternatively, one can provide here any custom atom pair selection in form of a character vector.
type	return fingerprint set as FPset, matrix or character vector

Details

...

Value

matrix or character vectors

Author(s)

Thomas Girke

References

Chen X and Reynolds CH (2002). "Performance of similarity measures in 2D fragment-based similarity searching: comparison of structural descriptors and similarity coefficients", J Chem Inf Comput Sci.

See Also

Functions: `sdf2ap`, `SDF2apcmp`, `apset2descdb`, `cmp.search`, `cmp.similarity`

Related classes: SDF, SDFset, SDFstr, APset.

Examples

```
## Instance of SDFset class
data(sdfsample)
sdfset <- sdfsample[1:10]

## Compute atom pair library
apset <- sdf2ap(sdfset)

## Compute atom pair fingerprint matrix using internal atom pair
## selection containing 4096 most common atom pairs in DrugBank.
## For details see ?apfp. The following example uses from this
## set the 1024 most frequent atom pairs:
fpset <- desc2fp(x=apset, descnames=1024, type="FPset")

## Alternatively, one can provide any custom atom pair selection. Here
## 1024 most common ones in apset object.
fpset1024 <- names(rev(sort(table(unlist(as(apset, "list")))))[1:1024])
fpset2 <- desc2fp(x=apset, descnames=fpset1024, type="FPset")

## A more compact way of storing fingerprints is as character values
fpchar <- desc2fp(x=apset, descnames=1024, type="character")

## Convert character fingerprints back to FPset or matrix
fpset <- as(fpchar, "FPset")
fpma <- as.matrix(fpset)

## Similarity searching returning Tanimoto similarity coefficients
fpSim(x=fpset[1], y=fpset)

## Clustering example
simMAap <- sapply(cid(fpset), function(x) fpSim(x=fpset[x], fpset, sorted=FALSE))
hc <- hclust(as.dist(1-simMAap), method="single")
plot(as.dendrogram(hc), edgePar=list(col=4, lwd=2), horiz=TRUE)
```

`draw_sdf` *draw_sdf*

Description

Draws an sdf object in the 2D plane using ggplot2 library. Permits customization of bond colors and atom colors.

Usage

```
draw_sdf(sdf, filename = "test.jpg", alpha_edge = 0.5, alpha_node = 1, numbered = FALSE, font_size
```

Arguments

`sdf` An instance of a SDF or list of SDFs

`filename` Filename to save image to. Defaults to 'test.jpg'. If set to NULL, does not save image.

`alpha_edge` alpha of bonds in your image. Defaults to 0.5. 0 is fully transparent, 1 is fully opaque.

`alpha_node` alpha of atoms in your image. Defaults to 1.0.

`numbered` If 1 or TRUE, displays numbering of atoms at their location. If 2, displays a second numbering style.

`font_size` Controls size of text to be displayed at atom locations. Beware when plotting multiple SDFs in one image. Ggplot will still scale fonts as if text is being plotted in one image.

`node_vertical_offset`
Upward shift of atom text. Upward shit is in SDF units, not ggplot units.

`bgcolor` An rgb(r,g,b,alpha) or similar object. produces a background of the specified color.

`node_background_color`
A common color as a text string (e.g. 'white', 'pink') or an rgb(r,g,b,alpha). Draws a filled circle of the color specified before drawing text over each node.

`bgraster` A readPNG object or a path to an object that can be understood using readPNG. Will be used as background.

`node_policy` Mapping that defines how atom strings should be displayed. Simplest would be c('default'='black')

`edge_policy` Mapping that defines how bonds should be displayed. Simplest is c('default'='black'), though this will display all Hydrogen bonds as well.

`bond_dist_offset`
Defines space between double or triple bonds, in SDF units.

`fmcsR_sdf` A second SDF object to run fmcsR on.

Details

Requires ggplot2. Additional features require grid, gridExtra, fmcsR, or png. Most matrix operations vectorized.

Value

Returns a ggplot2 object. Calling draw_sdf(...) rather than assigning it will result in R trying to print a ggplot2 object.

Author(s)

John A. Sharifi

Examples

```
library(ChemmineR)  # if not already imported
data(sdfsample)
draw_sdf(sdfsample[[1]])
```

exactMassOB *Exact Mass (Monoisotopic Mass)*

Description

Computes the exact mass of each compound given.

Usage

```
exactMassOB(sdfset)
```

Arguments

sdfset	Any SDFset object.

Value

A vector of mass values.

Author(s)

Kevin Horan

Examples

```
library(ChemmineR)
data(sdfsample)
mass = exactMassOB(sdfsample)
```

ExtSDF-class *Class "ExtSDF"*

Description

This is a subclass of `SDF` and thus inherits all the slots and methods from that class. It adds a list of extended attributes for atoms and bonds. These attributes can curretnly only be populated from a V3000 formatted SDF file.

Objects from the Class

Objects can be created by calls of the form `new("ExtSDF", ...)`. The function `read.SDFset` will also return objects of this class if the argument `extendedAttributes` is set to "TRUE".

Slots

`extendedAtomAttributes`: Object of class `"list"`

`extendedBondAttributes`: Object of class `"list"`

Methods

getAtomAttr `signature(x = "ExtSDF")`: returns atom attributes as a named list of values

getBondAttr `signature(x = "ExtSDF")`: returns bond attributes as a named list of values

show `signature(object = "ExtSDF")`: prints summary of `SDF` as well as any defined extended attributes for the atoms or bonds

Author(s)

Kevin Horan

References

SDF V3000 format definition: http://www.symyx.com/downloads/public/ctfile/ctfile.jsp

See Also

Related classes: SDF, SDFset, SDFstr, AP, APset

Examples

```
showClass("ExtSDF")
```

`findCompounds` *Find Compounds in Database*

Description

Searches the SQL database using features computed at load time. Each feature used should be specified in the `featureNames` parameter. Then a set of filters can be given to search for specific compounds.

Usage

```
findCompounds(conn, featureNames, tests)
```

Arguments

`conn`	A database connection object, such as is returned by `initDb`.
`featureNames`	A list of all feature names used in any test.
`tests`	A vector of filters that must all be true for a compound to be returned. For example: c("MW <= 400","RINGS > 3") would return all compounds with a molecular weight of 400 or less and a more than 3 rings, assuming these features exist in the database. The syntax for each test is "<feature name> <SQL operator> <value>". These tests will simply be concatenated together with " AND " in-between them and tacked on the end of a WHERE clause of an SQL statement. So any SQL that will work in that context is fine.

Value

Returns a list of compound ids. The actual compounds can be fetched with `getCompounds`.

Author(s)

Kevin Horan

See Also

`getCompounds`

Examples

```
   #create and initialize a new SQLite database
   conn = initDb("test1.db")

data(sdfsample)

#load data and compute 3 features: molecular weight, with the MW function,
# and counts for RINGS and AROMATIC, as computed by rings, which returns a data frame itself.
ids=loadSdf(conn,sdfsample,
  function(sdfset)
data.frame(MW = MW(sdfset),  rings(sdfset,type="count",upper=6, arom=TRUE))
 )
   #search for compounds with molecular weight less than 200
   lightIds = findCompounds(conn,"MW",c("MW < 200"))
   MW(getCompounds(conn,lightIds)) # should find one compound with weight 140
unlink("test1.db")
```

`findCompoundsByName` *Find compound by name*

Description

Find the ids of compounds given the names.

Usage

```
findCompoundsByName(conn, names, keepOrder = FALSE, allowMissing = FALSE)
```

Arguments

`conn` A database connection object, such as is returned by `initDb`.

`names` A list of names of compounds to search for. The names are those that would be returned by `sdfid`. An error will be raised if any names are not found.

`keepOrder` If true, the order of the output compound ids will be the same as the input names. This imposes a performance hit that can be significant for large datasets, thus it should be left FALSE unless needed.

`allowMissing` When this is false an error will be raised when names queried were not found in the database. If true, just those that are found will be returned with no error or warning.

Value

Returns the compound ids for compounds with the given name. The output order is not guaranteed unless keepOrder is set to TRUE. An error will be raised if any name cannot be found.

Author(s)

Kevin Horan

Examples

```
    #create and initialize a new SQLite database
    conn = initDb("test4.db")

data(sdfsample)

#just load the data with no features or descriptors
ids=loadSdf(conn,sdfsample)

    # find id of compound 650003
    findCompoundsByName(conn,c("650003"))
unlink("test4.db")
```

`fingerprintOB` *Fingerprints from OpenBabel*

Description

Generates fingerprints from SDFsets using OpenBabel. The name of the fingerprint can also be set and can be anything available through OpenBabel. You can see what this list is by executing "obabel -L fingerprints". Results are returned as an FPset.

Usage

```
fingerprintOB(sdfSet, fingerprintName)
```

Arguments

`sdfSet` Input compounds to generate fingerprints for.

`fingerprintName` The name of the fingerprint in Open Babel. A list of available names can be found by executing "obabel -L fingerprints". Currently that list is: "FP2", "FP3", "FP4", and "MACCS".

Value

An FPset with an element for each given compound.

Author(s)

Kevin Horan

Examples

```
## Not run:
data(sdfsample)
fpset = fingerprintOB(sdfsample)

## End(Not run)
```

fold *Fold*

Description

Fold a fingerprint. This takes the second half of the fingerprints and combines with the first half with a logical 'OR' operation. The result is a fingerprint with half as many bits.

Usage

```
fold(x, count = 1, bits = NULL)
```

Arguments

`x`	The fingerprint(s) to fold. This can be either an `FP` or an `FPset` object.
`count`	The number of times to fold this fingerprint. Folding will stop early if the fingerprint is reduced down to 1 bit before reaching the requested fold count.
`bits`	Fold this fingerprint until it is `bits` bits long. An exception will be thrown if `bits` is not reachable.

Value

The new, folded, fingerprint.

Author(s)

Kevin Horan

Examples

```
fp = new("FP",fp=c(1,0,1,1, 0,0,1,0))
foldedFp = fold(fp,bits=4)
```

foldCount *foldCount*

Description

Returns the number of times this fingerprint has been folded.

Usage

```
foldCount(x)
```

Arguments

`x`	Either an `FP` or an `FPset` object.

Value

Returns the number of times this fingerprint has been folded.

Author(s)

Kevin Horan

Examples

```
fp = new("FP",fp=c(1,0,1,1, 0,0,1,0))
foldedFp=fold(fp)
fc = foldCount(foldedFp) # == 1
```

FP-class *Class* "FP"

Description

Container for storing the fingerprint of a single compound. The FPset class is used for storing the fingerprints of many compounds.

Objects from the Class

Objects can be created by calls of the form new("FP", ...).

Slots

fp**:** Object of class "numeric"

foldCount**:** Object of class "numeric"

type**:** Object of class "character"

Methods

as.character signature(x = "FP"): returns fingerprint as character string

as.numeric signature(x = "FP"): returns fingerprint as numeric vector

as.vector signature(x = "FP"): returns fingerprint as numeric vector

coerce signature(from = "FPset", to = "FP"): coerce FPset component to list with many FP objects

coerce signature(from = "numeric", to = "FP"): construct FP object from numeric vector

show signature(object = "FP"): prints summary of FP

c signature(x = "FP"): concatenates any number of FP objects

fold signature(x = "FP"): fold fingerprint in half

foldCount signature(x = "FP"): number of times this object has been folded

fptype signature(x = "FP"): the type of this fingerprint

numBits signature(x = "FP"): the number of bits in this fingerprint

Author(s)

Thomas Girke

References

Chen X and Reynolds CH (2002). "Performance of similarity measures in 2D fragment-based similarity searching: comparison of structural descriptors and similarity coefficients", in J Chem Inf Comput Sci.

See Also

Related classes: SDF, SDFset, SDFstr, AP, APset, FPset.

Examples

```
showClass("FP")

## Instance of FP class
data(apset)
fpset <- desc2fp(apset)
(fp <- fpset[[1]])

## Class usage
fpc <- as.character(fp)
fpn <- as.numeric(fp)
as(fpn, "FP")
as(fpset[1:4], "FP")
```

---

fp2bit *Convert base 64 fingerprints to binary*

Description

The function converts the base 64 encoded PubChem fingerprints to a binary `matrix` or a `character` vector. If applied to a `SDFset` object, then its data block needs to contain the PubChem fingerprint information.

Usage

```
fp2bit(x, type = 3, fptag = "PUBCHEM_CACTVS_SUBSKEYS")
```

Arguments

x	Object of class `SDFset`, `matrix` or `character`
type	If set to `1`, the results are returned as binary `matrix`. If set to `2`, the results are returned as `character` strings in a named *vector*. If set to `3` (default), the results are returned as `FPset` object.
fptag	Name tag in SDF data block where the PubChem fingerprints are stored. Default is set to "PUBCHEM_CACTVS_SUBSKEYS".

Details

...

Value

matrix, character or FPset

Author(s)

Thomas Girke

References

See PubChem fingerprint specification at: ftp://ftp.ncbi.nih.gov/pubchem/specifications/pubchem_fingerprints.txt

See Also

Functions: fpSim

Examples

```
## Load PubChem SDFset sample
data(sdfsample); sdfset <- sdfsample
cid(sdfset) <- sdfid(sdfset)

## Convert base 64 encoded fingerprints to FPset object
fpset <- fp2bit(sdfset)

## Pairwise compound structure comparisons
fpSim(fpset[1], fpset[2])

## Structure similarity searching: x is query and y is fingerprint database
fpSim(x=fpset[1], y=fpset, method="Tanimoto", cutoff=0, top="all")

## Compute fingerprint based Tanimoto similarity matrix
simMA <- sapply(cid(fpset), function(x) fpSim(x=fpset[x], fpset, sorted=FALSE))

## Hierarchical clustering with simMA as input
hc <- hclust(as.dist(1-simMA), method="single")

## Plot hierarchical clustering tree
plot(as.dendrogram(hc), edgePar=list(col=4, lwd=2), horiz=TRUE)
```

FPset-class *Class* "FPset"

Description

Container for storing fingerprints of many compounds. This container is used for structure similarity searching of compounds.

Objects from the Class

Objects can be created by calls of the form new("FPset", ...).

Slots

`fpma`**:** Object of class `"matrix"` with compound identifiers stored in row names

`foldCount`**:** Object of class `"numeric"`

`type`**:** Object of class `"character"`

Methods

[`signature(x = "FPset")`: subsetting of class with bracket operator

[[`signature(x = "FPset")`: returns single component as `FP` object

[<- `signature(x = "FPset")`: replacement method for several components

as.character `signature(x = "FPset")`: returns content as named character vector

as.matrix `signature(x = "FPset")`: returns content as numeric matrix

c `signature(x = "FPset")`: concatenates any number of `FPset` containers

cid `signature(x = "FPset")`: returns all compound identifiers from row names

cid<- `signature(x = "FPset")`: replacement method for compound identifiers

coerce `signature(from = "FPset", to = "FP")`: `as(fpset, "FP")`

coerce `signature(from = "matrix", to = "FPset")`: `as(fpma, "FPset")`

coerce `signature(from = "character", to = "FPset")`: `as(fpchar, "FPset")`

length `signature(x = "FPset")`: returns number of entries stored in object

show `signature(object = "FPset")`: prints summary of `FPset`

view `signature(x = "FPset")`: prints extended summary of `FPset`

fold `signature(x = "FPset")`: fold fingerprint in half

foldCount `signature(x = "FPset")`: number of times this object has been folded

fptype `signature(x = "FPset")`: the type of these fingerprints

numBits `signature(x = "FPset")`: the number of bits in these fingerprints

Author(s)

Thomas Girke

References

Chen X and Reynolds CH (2002). "Performance of similarity measures in 2D fragment-based similarity searching: comparison of structural descriptors and similarity coefficients", in J Chem Inf Comput Sci.

See Also

Related classes: SDF, SDFset, SDFstr, AP, APset, FP.

Examples

```
showClass("FPset")

## Instance of FPset class
data(apset)
(fpset <- desc2fp(apset))
view(fpset)

## Class usage
fpset[1:4] # behaves like a list
fpset[[1]] # returns FP object
length(fpset) # number of compounds
cid(fpset) # returns compound ids
fpset[1] <- 0 # replacement
cid(fpset) <- 1:length(fpset) # replaces compound ids
c(fpset[1:4], fpset[11:14]) # concatenation

## Coerce FPset from/to other objects
fpma <- as.matrix(fpset) # coerces to matrix
fpchar <- as.character(fpset) # coerces to character strings
as(fpma, "FPset")
as(fpchar, "FPset")

## Compound similarity searching with FPset
fpSim(x=fpset[1], y=fpset, method="Tanimoto", cutoff=0.4, top=4)
```

`fpSim` *Fingerprint Search*

Description

Search function for fingerprints, such as PubChem or atom pair fingerprints. Enables structure similarity comparisons, searching and clustering.

Usage

```
fpSim(x, y, sorted=TRUE, method="Tanimoto",
addone=1, cutoff=0, top="all", alpha=1, beta=1,
parameters=NULL,scoreType="similarity")
```

Arguments

`x` Query molecule of class `numeric`, `FP` or `FPset` (of length one) containing binary fingerprint data. Both `x` and `y` need to have the same number of bits and should contain the same type of fingerprints.

`y` Subject molecule(s) of class `numeric`, `matrix`, `FP` or `FPset` containing binary fingerprint data.

`sorted` return results sorted or unsorted

`method` Similarity coefficient to return. One can choose here from several predefined similarity measures: "Tanimoto" (default), "Euclidean", "Tversky" or "Dice". Alternatively, one can pass on any custom similarity function containing the arguments a, b, c and d. For instance, one can define "myfct <- function(a, b, c,

d) c/(alpha*a + beta*b + c)" and then pass on `method=myfct`. The variable 'c' is the number of "on-bits" common in both compounds, 'd' is the number of "off-bits" common in both compounds, and 'a' and 'b' are the number of "on-bits" that are unique in one or the other compound, respectively.

The predefined methods will run a C++ version of this function which is about twice as fast as the R version. When a custom similarity function is given however, it will fall back to using the R version.

`addone` Value to add to numerator and denominator of similarity coefficient to avoid devision by zero when fingerprint(s) contain only "off-bits" (zeros). Note: if `addone > 0` then fingerprints with no "on-bits" will receive the highest similarity score. Typically, this occurs only with extremely small molecules.

`cutoff` allows to restrict results to hits above a similarity cutoff value; default `cutoff=0` returns results for all compounds in `y`.

`top` allows to restrict number of subject molecules to return; default `top="all"` returns results for all compounds in `y` above `cutoff` value.

`alpha` Only used when method="Tversky". Allows to specify the weighting variable 'alpha' of the Tversky index: c/(alpha*a + beta*b + c)

`beta` Only used when method="Tversky". Allows to specify the weighting variable 'beta' of the Tversky index.

`parameters` Parameters for computing Z-scores, E-values, and p-values. Pass this data if you want these scores returned. This data can be generated with the `genParameters` function.

`scoreType` If using the `parameters` argument, this argument specified which type of score the `cutoff` and `sorted` arguments should be applied to. It should be one of "similarity" (default), "zscore", "evalue", or "pvalue".

Value

Returns `numeric vector` with similarity coefficients as values and compound identifiers as names.

Author(s)

Thomas Girke, Kevin Horan

References

Tanimoto similarity coefficient: Tanimoto TT (1957) IBM Internal Report 17th Nov see also Jaccard P (1901) Bulletin del la Societe Vaudoisedes Sciences Naturelles 37, 241-272.

PubChem fingerprint specification: ftp://ftp.ncbi.nih.gov/pubchem/specifications/pubchem_fingerprints.txt

See Also

Functions: `fp2bit`

Examples

```
## Load PubChem SDFset sample
data(sdfsample); sdfset <- sdfsample
cid(sdfset) <- sdfid(sdfset)

## Convert base 64 encoded fingerprints to character vector or binary matrix
```

```
fpset <- fp2bit(sdfset)

## Alternatively, one can use atom pair fingerprints
## Not run:
fpset <- desc2fp(sdf2ap(sdfset))

## End(Not run)

## Pairwise compound structure comparisons
fpSim(x=fpset[1], y=fpset[2], method="Tanimoto")

## Structure similarity searching: x is query and y is fingerprint database
fpSim(x=fpset[1], y=fpset)

## Controlling the output
fpSim(x=fpset[1], y=fpset, method="Tversky", cutoff=0.4, top=4, alpha=0.5, beta=1)

## Use custom distance function
myfct <- function(a, b, c, d) c/(a+b+c+d)
fpSim(x=fpset[1], y=fpset, method=myfct)

## Compute fingerprint-based Tanimoto similarity matrix
simMA <- sapply(cid(fpset), function(x) fpSim(x=fpset[x], fpset, sorted=FALSE))

## Hierarchical clustering with simMA as input
hc <- hclust(as.dist(1-simMA), method="single")

## Plot hierarchical clustering tree
plot(as.dendrogram(hc), edgePar=list(col=4, lwd=2), horiz=TRUE)
```

fptype *fptype*

Description

Returns the type label of this fingerprint

Usage

```
fptype(x)
```

Arguments

x Either an FP or an FPset object.

Value

The type label of this fingerprint.

Author(s)

Kevin Horan

Examples

```
fp = new("FP",fp=c(1,0,1,1, 0,0,1,0),type="testFP")
type = fptype(fp) # == "testFP"
```

fromNNMatrix *From Nearest Neighbor Matrix*

Description

Converts a nearest neighbor matrix into a list that can be used with the jarvisPatrick function.

Usage

```
fromNNMatrix(data, names = rownames(data))
```

Arguments

data	A matrix containing integer valued indexes which represent items to be clustered. The index values contained in the matrix must be smaller than the number of rows in the matrix. Each row in the matrix represents one item and the columns are the nearest neighbors of that item.
names	The names for each row. The rownames of data will be used if not given.

Value

A list containing the slots "indexes" and "names".

Author(s)

Kevin Horan

See Also

jarvisPatrick

Examples

```
data(apset)

nn = nearestNeighbors(apset,cutoff=0.6)
nnMatrix = nn$indexes

cl = jarvisPatrick(fromNNMatrix(nnMatrix),k=2)
```

genAPDescriptors *Generate AP Descriptors*

Description

Generates Atom Pair descriptors using a fast C function.

Usage

```
genAPDescriptors(sdf,uniquePairs=TRUE)
```

Arguments

sdf	A single SDF object.
uniquePairs	When the same atom pair occurs more than once in a single compound, should the names be unique or not? Setting this to true will take slightly longer to compute.

Value

A vector of descriptors for the compound given. An AP object can be generated as shown in the example below.

Author(s)

Kevin Horan

Examples

```
library(ChemmineR)
data(sdfsample)
sdf = sdfsample[[2]]
ap = new("AP", AP=genAPDescriptors(sdf))
```

generate3DCoords *Generate 3D Coords*

Description

Uses Open Babel to compute 3D coordinates given an SDFset with only 2D coordinates.

Usage

```
generate3DCoords(sdf)
```

Arguments

sdf	Any sdfset object.

Value

A new SDFset in which all compounds have 3D coordinates.

Author(s)

Kevin Horan

Examples

```
## Not run:
data(sdfsample)
sdf3D = generate3DCoords(sdfsample[1])

## End(Not run)
```

genParameters *Generate Parameters*

Description

Generate statistics from a fingerprint database for use in caluclating z-scores, E-values, and p-values later.

Usage

```
genParameters(fpset, similarity = fpSim, sampleFraction = 1, ...)
```

Arguments

`fpset`	The database of fingerprints. Needs to be in the format expected by the similarity function. For the default similarity function, this would be an `FPset`.
`similarity`	A function to compute the similarity between two fingerprints. The first argument should be a single query and the second argument should be a set of fingerprints.
`sampleFraction`	The fraction of all pairs to use for estimating parameters. See Details section.
`...`	Extra parameters will be passed on to the similarity function.

Details

A beta function will be fit to the distribution of similarity scores produced by the given similarity function. By default, all pairwise similarities will be computed. Since this can be expensive for large databases, one can also sample pairs to use. This can be done by setting `sampleFraction` to the fraction of all pairwise similarities to use. For example, for a database of 100 fingerprints, there are 10,000 pairs. Setting `sampleFraction` to 0.5 will result in only 5,000 pairs being used to estimate the parameters.

Parameters are conditioned on the number of set bits. This function therefore groups fingerprints by the number of set bits they have and then estimates parameters for each group. A set of global parameters is also estimated and returned for use in cases where there was not enough data to estimate the parameters for a particular number of set bits.

Value

A data frame with the following columns:

count	The number of similarities used to estimate these parameters
avg	the mean
variance	the variance
alpha	The alpha parambe of the Beta function
beta	The beta parameter of the Beta function

There will be a row for each possible count of 1 bits. So for a database of 1024 bit fingerprints, there will be 1025 rows for the possible values of 0-1024 bits. There will also be one additional row at the end with the global parameters. This can be used for cases where there are no parameters estimated for the current query 1-bit count.

Author(s)

Kevin Horan

References

Pierre Baldi and Ramzi Nasr, "When is Chemical Similarity Significant? The Statistical Distribution of Chemical Similarity Scores and Its Extreme Values" Journal of Chemical Information and Modeling 2010 50 (7), 1205-1222

See Also

fpSim

Examples

```
library(ChemmineR)
data(apset)
fpset=desc2fp(apset) #get a fingerprint database

params = genParameters(fpset)
scores = fpSim(fpset[[1]],fpset,parameters=params,top=10)
```

---

getAllCompoundIds *Get ALl Compound Ids*

Description

Return a vector of every compound id in the given database.

Usage

```
getAllCompoundIds(conn)
```

Arguments

conn	A database connection object, such as is returned by initDb.

Value

A vector of compound_id numbers

Author(s)

Kevin Horan

Examples

```
   #create and initialize a new SQLite database
   conn = initDb("test1.db")

data(sdfsample)

#load data
ids=loadSdf(conn,sdfsample)
ids2=getAllCompoundIds(conn)
#ids == ids2

unlink("test1.db")
```

`getCompoundFeatures` *Get Compound Features*

Description

Get feature values for specific compounds.

Usage

```
getCompoundFeatures(conn, compoundIds, featureNames, filename = NA, keepOrder = FALSE, allowMissi
```

Arguments

`conn` A database connection object, such as is returned by `initDb`.

`compoundIds` A vector of compound_id numbers from this database. These are not compound names. Features will be fetched for each compound given here.

`featureNames` A vector of features to fetch the value for, for each given compound.

`filename` If given, dump the results into a comma seperated values (CSV) file instead of returning a data frame. This can avoid some potential memory limits when fetching large sets of data.

`keepOrder` Ensure that the output order of values matches the order in which the compound ids where given. This will make things a little slower, so should only be used where required.

`allowMissing` If false, raise an exception if a compound cannot be found, otherwise just silently ignore it and return data for whatever compound were found.

`batchSize` The number of compounds to fetch in a single query. If you find your running out of memory you can try reducing this values, as well as try writing the result to a file using the `filename` parameter.

Value

If filename is not given, returns a data frame with the compound_id and any given feature names. Each row represents one compound. If filename is given a filename then no value is returned, but the given file is created.

Author(s)

Kevin Horan

Examples

```
#create and initialize a new SQLite database
   conn = initDb("test1.db")

data(sdfsample)

#load data
ids=loadSdf(conn,sdfsample,
  function(sdfset)
data.frame(MW = MW(sdfset),  rings(sdfset,type="count",upper=6, arom=TRUE))
 )

f = getCompoundFeatures(conn,ids,c("mw","rings"))

unlink("test1.db")
```

getCompoundNames *Get Compound Names*

Description

Fetch the names of the given compound ids, if they exist

Usage

```
getCompoundNames(conn, compoundIds, keepOrder = FALSE, allowMissing = FALSE)
```

Arguments

conn A database connection object, such as is returned by initDb.

compoundIds A vector of compound ids.

keepOrder If true, the order of the output compound ids will be the same as the input names. This imposes a performance hit that can be significant for large datasets, thus it should be left FALSE unless needed.

allowMissing When this is false an error will be raised when compound ids queried were not found in the database. If true, just those that are found will be returned with no error or warning.

Value

Returns a vector of compound names.The rownames will be the compound ids. Compound ids not found, or for which a name is not defined, will be represented as NA.

Author(s)

Kevin Horan

Examples

```
   #create and initialize a new SQLite database
   conn = initDb("test2.db")

data(sdfsample)

#just load the data with no features or descriptors
ids=loadSdf(conn,sdfsample)

   getCompoundNames(conn,ids[1:3])
unlink("test3.db")
```

---

getCompounds   *Get Compounds From Database*

Description

Create SDF objects from the given set of compound ids. Id numbers can be found using the findCompounds function.

Usage

```
getCompounds(conn,compoundIds,filename=NA, keepOrder = FALSE, allowMissing = FALSE)
```

Arguments

`conn`	A database connection object, such as is returned by `initDb`.
`compoundIds`	A vector of compound ids, as returned by `loadSdf` or `findCompounds`.
`filename`	If given, writes the compounds directly to the file named.
`keepOrder`	If true, the order of the output compound ids will be the same as the input names. This imposes a performance hit that can be significant for large datasets, thus it should be left FALSE unless needed.
`allowMissing`	When this is false an error will be raised when compound ids queried were not found in the database. If true, just those that are found will be returned with no error or warning.

Value

An SDFset with the requested compounds or nothing if `filename` was specified. A warning will be raised if not all compounds could be found.

Author(s)

Kevin Horan

See Also

loadSdf findCompounds.

Examples

```
    #create and initialize a new SQLite database
    conn = initDb("test3.db")

data(sdfsample)

#just load the data with no features or descriptors
ids=loadSdf(conn,sdfsample)

    #returns a SDFset with 3 compounds
    getCompounds(conn, ids[1:3])

unlink("test3.db")
```

---

getIds *Import Compounds from PubChem*

Description

Accepts one or more PubChem compound ids and downloads the corresponding compounds from PubChem Power User Gateway (PUG) returning results in an SDFset container. The ChemMine Tools web service is used as an intermediate, to translate queries from plain HTTP POST to a PUG SOAP query.

Usage

```
getIds(cids)
```

Arguments

cids	A numeric object which contains one or more PubChem cids

Value

SDFset	for details see ?"SDFset-class"

Author(s)

Tyler Backman

References

PubChem PUG SOAP: http://pubchem.ncbi.nlm.nih.gov/pug_soap/pug_soap_help.html

Chemmine web service: http://chemmine.ucr.edu

PubChem help: http://pubchem.ncbi.nlm.nih.gov/search/help_search.html

Examples

```
## Not run:
## fetch 2 compounds from PubChem
compounds <- getIds(c(111,123))
## End(Not run)
```

grepSDFset *String search in* SDFset

Description

Convenience grep function for string searching in SDFset containers.

Usage

```
grepSDFset(pattern, x, field = "datablock", mode = "subset", ignore.case = TRUE, ...)
```

Arguments

pattern	search pattern
x	SDFset
field	delimits search to specific section in SDF; can be header, atomblock, bondblock or datablock
mode	if mode = "index", then the match positions are returned as vector; if mode = "subset", a list with SDF components is returned where every entry has at least one query match
ignore.case	TRUE turns off case sensitivity
...	option to pass on additional arguments

Details

...

Value

numeric	index vector where the name field contains the component positions in the SDFset and the values the row positions in each sub-component.
list	if mode = "subset"

Author(s)

Thomas Girke

References

...

See Also

Class: SDFset

Examples

```
## Instances of SDFset class
data(sdfsample)
sdfset <- sdfsample

## String Searching in SDFset
q <- grepSDFset("65000", sdfset, field="datablock", mode="subset")
as(q, "SDFset")
grepSDFset("65000", sdfset, field="datablock", mode="index")
```

---

groups *Enumeration of Functional Groups and Atom Neighbors*

Description

Returns frequency information of functional groups in molecules provided as SDF or SDFset objects. Alternatively, the function can return for each atom its atom/bond neighbor information.

Usage

```
groups(x, groups = "fctgroup", type = "countMA")
```

Arguments

x	SDF or SDFset containers
groups	if groups="fctgroup", frequencies of functional groups are returned; if groups="neighbors", atom/bond neighbor information is returned.
type	if type="all", then the complete neighbor information is generated for each atom in a molecule; if type="count", the neighbors are enumerated in a list and if type="countMA", then the counts of atom neighbors or functional groups are returned in a frequency matrix.

Details

At this point this function is in an experimental stage.

Value

...

Author(s)

Thomas Girke

References

...

See Also

...

Examples

```
## Instances of SDFset class
data(sdfsample)
sdfset <- sdfsample

## Enumerate functional groups
groups(sdfset[1:20], groups="fctgroup", type="countMA")

## Report atom/bond neighbors
groups(sdfset[1:4], groups="neighbors", type="countMA")
groups(sdfset[1:4], groups="neighbors", type="count")
groups(sdfset[1:4], groups="neighbors", type="all")
```

---

header *Return header block*

Description

Returns header block(s) from an object of class SDF or SDFset.

Usage

```
header(x)
```

Arguments

x	object of class SDF or SDFset

Details

...

Value

named character vector if SDF is provided or list of named character vectors if SDFset is provided

Author(s)

Thomas Girke

References

...

See Also

atomblock, atomcount, bondblock, datablock, cid, sdfid

Examples

```
## SDF/SDFset instances
data(sdfsample)
sdfset <- sdfsample
sdf <- sdfset[[1]]

## Extract header block
header(sdf)
header(sdfset[1:4])

## Replacement methods
sdfset[[1]][[1]][1] <- "test"
sdfset[[1]]
header(sdfset)[1] <- header(sdfset[2])
view(sdfset[1:2])
```

---

initDb *Iinitialize SQL Database*

Description

This will ensure that the database connection given is ready for use. If it does not find the tables it needs, it will try to create them.

Usage

```
initDb(handle)
```

Arguments

handle	This can be either a filename, in which case we assume it is the name of an SQLite database and use RSQLite to connect to it, or else any DBI Connection.

Value

Returns a connection object that can be used with other database oriented functions.

Author(s)

Kevin Horan

See Also

RSQLite

Examples

```
   #create and initialize a new SQLite database
   conn = initDb("test.db")
```

jarvisPatrick *Jarvis-Patrick Clustering*

Description

Function to perform Jarvis-Patrick clustering. The algorithm requires a nearest neighbor table, which consists of neighbors for each item in the dataset. This information is then used to join items into clusters with the following requirements: (a) they are contained in each other's neighbor list (b) they share at least 'k' nearest neighbors The nearest neighbor table can be computed with `nearestNeighbors`. For standard Jarvis-Patrick clustering, this function takes the number of neighbors to keep for each item. It also has the option of passing a cutoff similarity value instead of the number of neighbors. In this mode, all neighbors which meet the cutoff criteria will be included in the table. This is a setting that is not part of the original Jarvis-Patrick algorithm. It allows to generate tighter clusters and to minimize some limitations of this method, such as joining completely unrelated items when clustering small data sets. Other extensions, such as the `linkage` parameter, can also help improve the clustering quality.

Usage

```
jarvisPatrick(nnm,  k, mode="a1a2b", linkage="single")
```

Arguments

`nnm` A nearest neighbor table, as produced by `nearestNeighbors`.

`k` Minimum number of nearest neighbors two rows (items) in the nearest neighbor table need to have in common to join them into the same cluster.

`mode` If `mode = "a1a2b"` (default), the clustering is run with both requirements (a) and (b); if `mode = "a1b"` then (a) is relaxed to a unidirectional requirement; and if `mode = "b"` then only requirement (b) is used. The size of the clusters generated by the different methods increases in this order: "a1a2b" < "a1b" < "b". The run time of method "a1a2b" follows a close to linear relationship, while it is nearly quadratic for the much more exhaustive method "b". Only methods "a1a2b" and "a1b" are suitable for clustering very large data sets (e.g. >50,000 items) in a reasonable amount of time.

`linkage` Can be one of "single", "average", or "complete", for single linkage, average linkage and complete linkage merge requirements, respectively. In the context of Jarvis-Patrick, average linkage means that at least half of the pairs between the clusters under consideration must pass the merge requirement. Similarly, for complete linkage, all pairs must pass the merge requirement. Single linkage is the normal case for Jarvis-Patrick and just means that at least one pair must meet the requirement.

Details

...

Value

Depending on the setting under the `type` argument, the function returns the clustering result in a `named vector` or a nearest neighbor table as `matrix`.

Note

...

Author(s)

Thomas Girke

References

Jarvis RA, Patrick EA (1973) Clustering Using a Similarity Measure Based on Shared Near Neighbors. IEEE Transactions on Computers, C22, 1025-1034. URLs: http://davide.eynard.it/teaching/2012_PAMI/JP.pdf, http://www.btluke.com/jpclust.html, http://www.daylight.com/dayhtml/doc/cluster/index.pdf

See Also

Functions: `cmp.cluster` `trimNeighbors` `nearestNeighbors`

Examples

```
## Load/create sample APset and FPset
data(apset)
fpset <- desc2fp(apset)

## Standard Jarvis-Patrick clustering on APset/FPset objects
jarvisPatrick(nearestNeighbors(apset,numNbrs=6), k=5, mode="a1a2b")
jarvisPatrick(nearestNeighbors(fpset,numNbrs=6), k=5, mode="a1a2b")

## Jarvis-Patrick clustering only with requirement (b)
jarvisPatrick(nearestNeighbors(fpset,numNbrs=6), k=5, mode="b")

## Modified Jarvis-Patrick clustering with minimum similarity 'cutoff'
## value (here Tanimoto coefficient)
jarvisPatrick(nearestNeighbors(fpset,cutoff=0.6, method="Tanimoto"), k=2 )

## Output nearest neighbor table (matrix)
nnm <- nearestNeighbors(fpset,numNbrs=6)

## Perform clustering on precomputed nearest neighbor table
jarvisPatrick(nnm, k=5)
```

`jarvisPatrick_c` *Jarvis Patrick Clustering in C code*

Description

This not meant to be used directly, use `jarvisPatrick` instead. It is exposed so other libraries can make use of it.

Usage

```
jarvisPatrick_c(neighbors,minNbrs,fast=TRUE,bothDirections=FALSE,linkage = "single")
```

Arguments

`neighbors`	A matrix of integers. Non integer matricies will be coerced. Each row represensts one element, indexed 1 to N. The values in row i should be the index value of the neighbors of i. Thus, each value should itself be a valid row index.
`minNbrs`	The minimum number of common neibhbors needed for two elements to be merged.
`fast`	If true, only the neibhors given in each row are checked to see if they share `minNbrs` neighbors in common. If false, all pairs of elements are compared. For a matrix of size NxM, the first method yeilds a running time of O(NM), while the second yeilds a running time of O(N^2).
`bothDirections`	If true, two elements must contain each other in their neighbor list in order to be merged. If false and fast is true, then only one element must contain the other as a neighbor. If false and fast is false, than neither element must contain the other as a neighbor, though in all cases there must still be at least `minNbrs` neibhros in common.
`linkage`	See `jarvisPatrick` for details.

Value

A cluster array with no names.

Author(s)

Kevin Horan

`jobToken-class` *Class* `"jobToken"`

Description

Container for storing a reference to a remote job ran on the ChemMine Tools web server.

Objects from the Class

Objects can be created by calls of the form `new("jobToken", ...)`.

Slots

`tool_name`**:** Object of class `"character"`

`jobId`**:** Object of class `"character"`

Methods

show `signature(object = "jobToken")`: check the status of a launched job

Author(s)

Tyler William H Backman

References

See ChemMine Tools at http://chemmine.ucr.edu.

See Also

Functions: `launchCMTool`, `toolDetails`, `listCMTools`, `result`, `browseJob`, `status`

Examples

```
showClass("jobToken")
## Not run:
## launch a job on the server and obtain jobToken back
job1 <- launchCMTool("pubchemID2SDF", 2244)

## check status of the job
status(job1)

## obtain results
result1 <- result(job1)
result1

## End(Not run)
```

`launchCMTool` *Launch a Tool on ChemMine Tools*

Description

Accepts a tool name (string), input options, and input data to launch a remote web tool on the ChemMine Tools website.

Usage

```
launchCMTool(tool_name, input = "", ...)
```

Arguments

`tool_name`	A tool name matching verbatim an existing tool name as listed by `listCMTools`.
`input`	Input data in the format required for this tool as listed by `listCMTools`.
`...`	Additional options as mentioned by running `toolDetails` on the tool specified.

Details

By running the function `toolDetails` on a tool of choice, you can see a pre-generated example function call for this tool.

Value

`jobToken`	for details see ?"jobToken-class"

Author(s)

Tyler William H Backman

References

See ChemMine Tools at http://chemmine.ucr.edu.

See Also

Functions: toolDetails, listCMTools, result, browseJob, status

Examples

```
## Not run:
## list available tools
listCMTools()

## get detailed instructions on using a tool
toolDetails("Fingerprint Search")

## download compound 2244 from PubChem
job1 <- launchCMTool("pubchemID2SDF", 2244)

## check job status and download result
status(job1)
result1 <- result(job1)

## End(Not run)
```

listCMTools *List all available ChemMine Tools*

Description

Connects to the ChemMine Tools web service and obtains a list of all available tools, and their input and output formats.

Usage

```
listCMTools()
```

Value

data.frame	A four column data.frame which describes a tool on each row

Author(s)

Tyler William H Backman

References

See ChemMine Tools at http://chemmine.ucr.edu.

See Also

Functions: toolDetails, launchCMTool, result, browseJob, status

Examples

```
## Not run:
## list available tools
listCMTools()

## get detailed instructions on using a tool
toolDetails("Fingerprint Search")

## download compound 2244 from PubChem
job1 <- launchCMTool("pubchemID2SDF", 2244)

## check job status and download result
status(job1)
result1 <- result(job1)

## End(Not run)
```

---

listFeatures *List Features*

Description

List the available features in the given database. These features can be used in the findCompounds function.

Usage

```
listFeatures(conn)
```

Arguments

conn	Database connection

Value

A vector of character feature names.

Author(s)

Kevin Horan

See Also

findCompounds

Examples

```
   #create and initialize a new SQLite database
   conn = initDb("test7.db")

data(sdfsample)

#just load the data with no features or descriptors
```

```
ids=loadSdf(conn,sdfsample,fct=function(sdfset) cbind(mw=MW(sdfset)))
listFeatures(conn) # produces c("mw")
unlink("test7.db")
```

loadSdf — *Load SDF and SMILES Data*

Description

Load an SDF or SMILES formatted file or SDFSet objects into the database. This will also load arbitrary features from the data as well as descriptor data. The `fct` parameter can be used to specify a function which will compute features which will then be indexed and stored in the database. These features can later be used to quickly search for compounds. Descriptors can also be computed and stored in another table.

Usage

```
loadSdf(conn, sdfFile, fct = function(x) data.frame(), descriptors=function(x) data.frame(descrip
Nlines = 50000, startline = 1, restartNlines = 1e+05,updateByName=FALSE)
loadSmiles(conn, smileFile, ...)
```

Arguments

`conn` A database connection object, such as is returned by `initDb`.

`sdfFile` Either the filename of an SDF formated file, or and SDFSet object.

`smileFile` The filename of an SMILES formated file.

`...` When calling loadSmiles, any of the arguments for loadSdf can be used and will be passed to loadSdf internally.

`fct` A function to extract features from the data. It will be handed an SDFSet generated from the data being loaded. This may be done in batches, so there is no guarantee that the given SDFSset will contain the whole dataset. This function should return a data frame with a column for each feature and a row for each compound given, and in the same order. Each of these features will become a new, indexed, table in the database which can be used later to search for compounds. The column name will become the feature name. If not given, no features are computed.

`descriptors` This function will also be given an SDFSet object, which may be done in batches. It should return a data frame with the following two columns: "descriptor" and "descriptor_type". The "descriptor" column should contain a string representation of the descriptor, and "descriptor_type" is the type of the descriptor. Our convention for atom pair is "ap" and "fp" for finger print. The order should be maintained. If not given no descriptors are computed.

`Nlines` When reading data from a file, the number of lines to read at a time. See also `sdfStream`.

`startline` When reading data from a file, the line number to start reading from.See also `sdfStream`

`restartNlines` When reading data from a file and startline > 1, the number of lines to look forward to find the start of the next compound. See also `sdfStream` updateByName If true we make the assumption that all compounds, both in the existing database and the given dataset, have unique names. This function will then avoid re-adding existing, identical compounds, and will update existing compounds with a new definition if a new compound definition with an existing name is given.

If false, we allow duplicate compound names to exist in the database, though not duplicate definitions. So identical compounds will not be re-added, but if a new version of an existing compound is added it will not update the existing one, it will add the modified one as a completely new compound with a new compound id.

Details

Arguments to loadSmiles are the same as those to loadSdf. LoadSmiles will convert its input into an SDFSet and then call loadSdf.

New features can also be added using this function. However, all compounds must have all features so if new features are added to a new set of compounds, all existing features must be computable by the fct function given. If new features are detected, all existing compounds will be run through fct in order to compute the new features for them as well.

For example, if dataset X is loaded with features F1 and F2, and then at a later time we load dataset Y with new feature F3, the fct function used to load dataset Y must compute and return features F1, F2, and F3. loadSdf will call fct with both datasets X and Y so that all features are available for all compounds. If any features are missing an error will be raised.

If just new features are being added, but no new compounds, use the addNewFeatures function.

Value

Returns the compound id numbers of each compound loaded. These can be used to retrieve compounds later. These are id numbers computed by the database and are not extracted from the compound data itself.

Author(s)

Kevin Horan

See Also

sdfStream

Examples

```
   #create and initialize a new SQLite database
   conn = initDb("test6.db")

data(sdfsample)

#just load the data with no features or descriptors
ids=loadSdf(conn,sdfsample)
unlink("test6.db")

   conn = initDb("test5.db")
#load data and compute 3 features: molecular weight, with the MW function,
# and counts for RINGS and AROMATIC, as computed by rings, which returns a data frame itself.
ids=loadSdf(conn,sdfsample,
  function(sdfset)
data.frame(MW = MW(sdfset),  rings(sdfset,type="count",upper=6, arom=TRUE))
 )
unlink("test5.db")
```

makeUnique *Uniquify CMP names*

Description

Creates unique CMP names by appending a counter to each duplicatation set. The function can be used for any character vector.

Usage

```
makeUnique(x, silent = FALSE)
```

Arguments

x	character vector
silent	silent = TRUE suppresses message about duplicate count

Details

The function is important to maintain unique compound names in the ID slot of SDFset containers.

Value

character	of same length as x but without duplications

Author(s)

Thomas Girke

References

...

See Also

Functions: cid, sdfid

Examples

```
## SDFset instance
data(sdfsample)
sdfset <- sdfsample

## Create unique compound IDs
unique_ids <- makeUnique(sdfid(sdfset))
cid(sdfset) <- unique_ids
cid(sdfset[1:4])
```

`maximallyDissimilar` *Maximally Dissimilar*

Description

Find a set of compounds that are far away from each other.

Usage

```
maximallyDissimilar(compounds, n, similarity = cmp.similarity)
```

Arguments

`compounds`	The set of items from which to pick `n` dissimlar items. This can be a list of anything that the similarity function will accept. By default this will be an APset.
`n`	The number of dissimilar items to return.
`similarity`	The similarity function to use. By default Tanimoto will be used on APset objects. Internally, this will be converted to a distance function using `1-similarity(a,b)`, so whatever similarity function you use should return a value between 0 and 1.

Details

This will run in O(length(compounds)n) time. Based on the algorithm described in (Higgs,1997).

Value

A vector of indexes of the dissimilar items.

Author(s)

Kevin Horan

References

Higgs, R.E., Bemis, K.G., Watson, I.A., and Wikel, J.H. 1997. Experimental designs for selecting molecules from large chemical databases. J. Chem. Inf. Comput. Sci. 37, 861-870

Examples

```
data(apset)
maximallyDissimilar(apset,10)
```

nearestNeighbors *Nearest Neighbors*

Description

Computes the nearest neighbors of descriptors in an FPset or APset object for use with the jarvisPatrick clustering function. Only one of numNbrs or cutoff should be given, cutoff will take precedence if both are given. If numNbrs is given, then that many neighbors will be returned for each item in the set. If cutoff is given, then, for each item X, every neighbor that has a similarity value greater than or equal to the cutoff will be returned in the neighbor list for X.

Usage

```
nearestNeighbors(x, numNbrs = NULL, cutoff = NULL, ...)
```

Arguments

x	Either an FPset or an APset.
numNbrs	Number of neighbors to find for each item. If not enough neighbors can be found the matrix will be padded with NA.
cutoff	The minimum similarity value an item must have to another item in order to be included in that items neighbor list. This parameter takes precedence over numNbrs. This parameter allows to obtain tighter clustering results.
...	These parameters will be passed into the distance function used, either cmp.similarity or fpSim, for APset and FPset, respectively.

Value

The return value is a list with the following components:

indexes	index values of nearest neighbors, for each item. If cutoff is used, this will be a list of lists, otherwise it will be a matrix
names	The names of each item in the set, as returned by cid
similarities	The similarity values of each neighbor to the item for that row. This will also be either a list of lists or a matrix, depending on whether or not cutoff was used. Each similarity values corresponds to the id number in the same position in the indexes entry

Author(s)

Kevin Horan

See Also

jarvisPatrick trimNeighbors

Examples

```
data(sdfsample)
ap = sdf2ap(sdfsample)
nnm = nearestNeighbors(ap,cutoff=0.5)
clustering = jarvisPatrick(nnm,k=2,mode="a1b")
```

numBits *numBits*

Description

Returns the number of bits in a fingerprint.

Usage

```
numBits(x)
```

Arguments

x	Either an FP or an FPset object.

Value

The number of bits in this fingerprint object.

Author(s)

Kevin Horan

Examples

```
fp = new("FP",fp=c(1,0,1,1, 0,0,1,0))
n = numBits(fp) # == 8
```

obmol *obmol*

Description

Return reference to an OBMol from OpenBabel, if available. Operates on SDF or SDFset objects.

Usage

```
obmol(x)
```

Arguments

x	object of class SDF or SDFset

Value

A pointer to an OBMol object, or a vector of pointers for an SDFset.

Author(s)

Kevin Horan

See Also

header, atomcount, bondblock, datablock, cid, sdfid

Examples

```
## SDF/SDFset instances
if(require(ChemmineOB)){
data(sdfsample)
sdfset <- sdfsample
sdf <- sdfset[[1]]

obmolRef = obmol(sdf)
}
```

---

parBatchByIndex *Parallel Batch By Index*

Description

Takes an index set, breaks it into batches and runs the given function on each batch in parallel using the given cluster. See batchByIndex for the non-parallel version.

When doing a select were the condition is a large number of ids it is not always possible to include them in a single SQL statement. This function will break the list of ids into chunks and allow the indexProcessor to deal with just a small number of ids.

Usage

```
parBatchByIndex(allIndices, indexProcessor, reduce, cl, batchSize = 1e+05)
```

Arguments

allIndices	A vector of values that will be broken into batches and passed as an argument to the indexProcessor function.
indexProcessor	A function that takes one batch if indices. It is called once for each batch, possibly in parallel. The return value of this function is collected into a list and passed to the reduce function after all jobs have finished.
reduce	This function is run after all jobs have finished. It is called with a list of return values from the indexProcessor function runs. The order of batchs is maintained. The return value of the reduce function is then returned. The idea is that this function merges all the results together into one result.
cl	A SNOW cluster to run jobs on.
batchSize	The size of each batch. The last batch may be smaller than this value.

Value

The return value of the reduce function is returned.

Author(s)

Kevin Horan

See Also

batchByIndex

Examples

```
## Not run:

cl = makeCluster(2) # create a SNOW cluster

#function to run a query for each batch of indexes
job = function(indexBatch)
dbGetQuery(dbConnection, paste("SELECT weight FROM table WHERE id IN (",paste(indexBatch,collapse=","),")"))

# function to combine all the results, in this case by summing them up
reduce = function(results) sum(unlist(results))

indices = 1:10000

#run queries in parallel and then sum the results
totalWeight = parBatchByIndex(indices,job,reduce,cl, 1000)


## End(Not run)
```

plotStruc *Plot compound structures*

Description

Plots compound structure(s) for molecules stored in SDF and SDFset containers.

Usage

```
## Convenience plot method
# plot(x, griddim, print_cid=cid(x), print=TRUE, ...)

## Less important for user
plotStruc(sdf, atomcex = 1.2, atomnum = FALSE, no_print_atoms = c("C"),
          noHbonds = TRUE, bondspacer = 0.12, colbonds=NULL, bondcol="red",
 regenCoords=FALSE, ...)
```

Arguments

`sdf` Object of class SDF

`atomcex` Font size for atom labels

`atomnum` If `TRUE`, then the atom numbers are included in the plot. They are the position numbers of each atom in the atom block of an SDF.

`no_print_atoms` Excludes specified atoms from being plotted.

`noHbonds` If `TRUE`, then the C-hydrogens and their bonds - explicitly defined in an SDF - are excluded from the plot.

`bondspacer` Numeric value specifying the plotting distance for double/triple bonds.

colbonds Highlighting of subgraphs in main structure by providing a numeric vector of atom numbers, here position index in atom block. The bonds of connected atoms will be plotted in the color provided under bondcol.

bondcol A character or numeric vector of length one to specify the color to use for substructure highlighting under colbonds.

regenCoords If ChemmineOB is installed and this option is TRUE, then Open Babel will be used to re-generate the 2D coords for each compound before plotting it. This often results in a nicer layout. If you want to save the results of the coord regeneration, call the regenerateCoords function first yourself and save the result.

... Arguments to be passed to/from other methods.

Details

The function plotStruc depicts a single 2D compound structure based on the XY-coordinates specified in the atom block of an SDF. The generic method plot can be used as a convenient shorthand to plot one or many structures at once. Both functions depend on the availability of the XY-coordinates in the source SD file and only 2D (not 3D) representations are plotted correctly.

Additional arguments that can only be passed on to the plot function when supplied with an SDFset object:

griddim: numeric vector of length two to define the dimensions for arranging several structures in one plot.

print_cid: character vector for printing custom compound labels. Default is print_cid=cid(sdfset).

print: if print=TRUE, then a summary of the SDF content for each supplied compound is printed to the screen. This behavior is turned off with print=TRUE.

Value

Prints summary of SDF/SDFset to screen and plots their structures to graphics device.

Note

The compound depictions created by this function are not as pretty as the structure representations generated with the sdf.visualize function. This will be improved in the future.

Author(s)

Thomas Girke

References

...

See Also

sdf.visualize

Examples

```
## Import SDFset sample set
data(sdfsample)
(sdfset <- sdfsample)

## Plot single compound structure
plotStruc(sdfset[[1]])

## Plot several compounds structures
plot(sdfset[1:4])

## Highlighting substructures (here all rings)
myrings <- as.numeric(gsub(".*_", "", unique(unlist(rings(sdfset[1])))))
plot(sdfset[1], colbonds=myrings)

## Customize plot
plot(sdfset[1:4], griddim=c(2,2), print_cid=letters[1:4], print=FALSE, noHbonds=FALSE)
```

---

propOB *Properties from OpenBabel*

Description

Generates the following descriptors: "cansmi", "cansmiNS", "formula", "HBA1", "HBA2", "HBD", "InChI", "InChIKey", "logP", "MR", "MW", "nF","title", "TPSA".

Usage

```
propOB(sdfSet)
```

Arguments

sdfSet	An SDFset object.

Value

A data frame with a row for each compound in the given data frame and a named column for each property.

Author(s)

Kevin Horan

Examples

```
## Not run:
library(ChemmineR)
data(sdfsample)
propOB(sdfsample)

## End(Not run)
```

`pubchemFPencoding` *Enncoding of PubChem Fingerprints*

Description

Data frame with bit positions and substructure specifications.

Usage

```
data(pubchemFPencoding)
```

Format

The format is a data frame with 881 rows and 2 columns.

Source

From: ftp://ftp.ncbi.nih.gov/pubchem/specifications/pubchem_fingerprints.txt

References

See: ftp://ftp.ncbi.nih.gov/pubchem/specifications/pubchem_fingerprints.txt

Examples

```
data(pubchemFPencoding)
pubchemFPencoding[1:4,]
```

`read.AP` *Read Atom Pair/Fingerprint Strings*

Description

Function to convert atom pairs (AP) or fingerprints (e.g. AP fingerprints) stored as character strings to `APset` or `FPset` objects (e.g. generated by `sdfStream`). Alternatively, one can provide the AP or fingerprint strings in a named character vector.

Usage

```
read.AP(x, type, colid, isFile = class(x) == "character" & length(x) == 1)
```

Arguments

`x`	name of file from where to read the AP/APFP character strings; or named character vector containing the AP/APFP strings
`type`	`type="ap"` for AP character string input, and `type="fp"` for fingerprint character string input
`colid`	column containing AP/FP character strings if `x` is a file
`isFile`	Is `x` a file name or not?

Details

...

Value

object of class APset or FPset

Author(s)

Thomas Girke

References

...

See Also

sdf2ap, sdfStream

Examples

```
## Load sample data
library(ChemmineR)
data(sdfsample); sdfset <- sdfsample
## Not run: write.SDF(sdfset, "test.sdf")

## Define descriptor set in a simple function
desc <- function(sdfset) {
        cbind(SDFID=sdfid(sdfset),
              # datablock2ma(datablocklist=datablock(sdfset)),
              MW=MW(sdfset),
              groups(sdfset),
              APFP=desc2fp(x=sdf2ap(sdfset), descnames=1024, type="character"),
              AP=sdf2ap(sdfset, type="character"),
              rings(sdfset, type="count", upper=6, arom=TRUE)
        )
}

## Run sdfStream with desc function and write results to a file called 'matrix.xls'
sdfStream(input="test.sdf", output="matrix.xls", fct=desc, Nlines=1000)

## Select molecules from SD File using line index from sdfStream
indexDF <- read.delim("matrix.xls", row.names=1)[,1:4]
indexDFsub <- indexDF[indexDF$MW < 400, ] # Selects molecules with MW < 400
sdfset <- read.SDFindex(file="test.sdf", index=indexDFsub, type="SDFset")

## Write result directly to SD file without storing larger numbers of molecules in memory
read.SDFindex(file="test.sdf", index=indexDFsub, type="file", outfile="sub.sdf")

## Read AP/APFP strings from file into APset or FP object
apset <- read.AP(x="matrix.xls", type="ap", colid="AP")
apfp <- read.AP(x="matrix.xls", type="apfp", colid="APFP")

## Alternatively, one can provide the AP/APFP strings in a named character vector
apset <- read.AP(x=sdf2ap(sdfset[1:20], type="character"), type="ap")
apfp <- read.AP(x=desc2fp(x=sdf2ap(sdfset[1:20]), descnames=1024, type="character"), type="apfp")
```

```
## End(Not run)
```

read.SDFindex *Extract Molecules from SD File by Line Index*

Description

Extracts specific molecules from SD File based on a line position index computed by the `sdfStream` function.

Usage

```
read.SDFindex(file, index, type = "SDFset", outfile)
```

Arguments

`file`	file name of source SD file used to generate `index`
`index`	data frame containing in the first two columns the start and end positions (index) of molecules in an SD File, respectively. Typically, this index would be imported with `read.table/read.delim` from a tabular descriptor file generated by the `sdfStream` function.
`type`	if `type="file"`, the SDF output will be written to a file named as specified under `outfile`; if `type="SDFset"`, the SDF data is collected will be a `SDFset` container.
`outfile`	name of output file when `type="file"`

Details

...

Value

Writes molecules in SDF format to file or collects them in `SDFset` container.

Author(s)

Thomas Girke

References

SDF format definition: http://www.symyx.com/downloads/public/ctfile/ctfile.jsp

See Also

Import/export functions: `read.SDFset`, `read.SDFstr`, `read.SDFstr`, `read.SDFset`, `write.SDFsplit`

Examples

```
## Load sample data
library(ChemmineR)
data(sdfsample); sdfset <- sdfsample
## Not run: write.SDF(sdfset, "test.sdf")

## Define descriptor set in a simple function
desc <- function(sdfset) {
        cbind(SDFID=sdfid(sdfset),
              # datablock2ma(datablocklist=datablock(sdfset)),
              MW=MW(sdfset),
              groups(sdfset),
              # AP=sdf2ap(sdfset, type="character"),
              rings(sdfset, type="count", upper=6, arom=TRUE)
        )
}

## Run sdfStream with desc function and write results to a file called 'matrix.xls'
sdfStream(input="test.sdf", output="matrix.xls", fct=desc, Nlines=1000)

## Select molecules from SD File using line index from sdfStream
indexDF <- read.delim("matrix.xls", row.names=1)[,1:4]
indexDFsub <- indexDF[indexDF$MW < 400, ] # Selects molecules with MW < 400
sdfset <- read.SDFindex(file="test.sdf", index=indexDFsub, type="SDFset")

## Write result directly to SD file without storing larger numbers of molecules in memory
read.SDFindex(file="test.sdf", index=indexDFsub, type="file", outfile="sub.sdf")

## End(Not run)
```

---

read.SDFset *SD file to* SDFset

Description

Imports one or many molecules from an SD/MOL file and stores it in an SDFset container. Supports both the V2000 and V3000 formats.

Usage

```
read.SDFset(sdfstr = sdfstr, ...)
```

Arguments

sdfstr	path/name to an SD file; alternatively an SDFstr object can be provided
...	option to pass on additional arguments. Possible arguments are: datablock: true or false, whether to include the data fields or not. Defaults to TRUE. tail2vec: true or false, whether to return data feilds as a vector or not. Defaults to TRUE. extendedAttributes: true or false, whether to parse the extended attributes available on the V3000 format. Defaults to FALSE.

Details

...

Value

`SDFset`	for details see ?"SDFset-class"

Author(s)

Thomas Girke

References

SDF format defintion: http://www.symyx.com/downloads/public/ctfile/ctfile.jsp

See Also

Functions: `read.SDFstr`

Examples

```
## Write instance of SDFset class to SD file
data(sdfsample); sdfset <- sdfsample
# write.SDF(sdfset[1:4], file="sub.sdf")

## Import SD file
# read.SDFset("sub.sdf")

## Pass on SDFstr object
sdfstr <- as(sdfset, "SDFstr")
read.SDFset(sdfstr)
```

`read.SDFstr` *SD file to* `SDFstr`

Description

Imports one or many molecules from an SD/MOL file and stores it in an `SDFstr` container.

Usage

```
read.SDFstr(sdfstr)
```

Arguments

`sdfstr`	path/name to an SD file; alternatively one can pass on a `character` vector containing lines of an SD file

Details

...

Value

SDFstr	for details see ?"SDFstr-class"

Author(s)

Thomas Girke

References

SDF format defintion: http://www.symyx.com/downloads/public/ctfile/ctfile.jsp

See Also

Functions: read.SDFset

Examples

```
## Write instance of SDFstr class to SD file
data(sdfsample); sdfset <- sdfsample
sdfstr <- as(sdfset, "SDFstr")
# write.SDF(sdfset[1:4], file="sub.sdf")

## Import SD file
# read.SDFstr("sub.sdf")

## Pass on SDFstr object
sdfstr <- as(sdfset, "SDFstr")
read.SDFset(sdfstr)
```

read.SMIset *SMILES file to* SMIset

Description

Imports one or many molecules from a SMILES file and stores content in a SMIset container. The input file is expected to contain one SMILES string per row with tab-separated compound identifiers at the end of each line. The compound identifiers are optional.

Usage

```
read.SMIset(file, removespaces = TRUE, ...)
```

Arguments

file	path/name to a SMILES file
removespaces	if set to TRUE spaces will be removed
...	option to pass on additional arguments

Details

...

Value

SMIset	for details see ?"SMIset-class"

Author(s)

Thomas Girke

References

SMILES (Simplified molecular-input line-entry system) format definition: http://en.wikipedia.org/wiki/Simplified_molec
input_line-entry_system

See Also

Functions: read.SDFset

Examples

```
## Write instance of SMIset class to SMILES file
data(smisample); smiset <- smisample
# write.SMI(smiset[1:4], file="sub.smi")

## Import SMILES file
# read.SMIset("sub.smi")
```

---

regenerateCoords *Re-generate 2D Coordinates*

Description

This uses Open Babel (requires ChemmineOB package) to re-generate the 2D coordinates of compounds. This often results in a nicer layout of the compound when plotting.

Usage

```
regenerateCoords(sdf)
```

Arguments

sdf	A SDF or SDFset object whose coordinates will be re-generated.

Value

Either an SDF object if given an SDF, or else and SDFset.

Author(s)

Kevin Horan

See Also

plotStruc

Examples

```
## Not run:
data(sdfsample)
prettySdfset = regenerateCoords(sdfsample[1:4])

## End(Not run)
```

---

result *Obtain the resulting output data from a ChemMine Tools Job*

Description

Accepts a `jobToken` job as returned by the function `launchCMTool` and returns the final result. If the job is still running, the function will loop until the job is ready.

Usage

```
result(object)
```

Arguments

`object`	A `jobToken` job as returned by the function `launchCMTool`

Value

Output will be in the format specified for this tool, as listed with the `listCMTools` function.

Author(s)

Tyler William H Backman

References

See ChemMine Tools at http://chemmine.ucr.edu.

See Also

Functions: `toolDetails`, `listCMTools`, `launchCMTool`, `browseJob`, `status`

Examples

```
## Not run:
## list available tools
listCMTools()

## get detailed instructions on using a tool
toolDetails("Fingerprint Search")

## download compound 2244 from PubChem
job1 <- launchCMTool("pubchemID2SDF", 2244)

## check job status and download result
```

```
status(job1)
result1 <- result(job1)

## End(Not run)
```

rings *Ring and Aromaticity Perception*

Description

Identifies all possible rings in molecules using the exhaustive ring perception algorithm from Hanser et al (1996). In addition, the function can return all smallest possible rings as well as aromaticity information for each ring.

Usage

```
rings(x, upper = Inf, type = "all", arom = FALSE, inner = FALSE)
```

Arguments

x	SDF or SDFset containers
upper	allows to specify an upper length limit for ring predictions. The default setting upper=Inf will return all possible rings. Smaller length limits will reduce the search space resulting in shortened compute times.
type	if type="all", the function returns each ring of a compound as character vector of atom symbols that are numbered by their position in the atom block of an SDF/SDFset object. Note: the example below shows how to plot structures with the same numbering information for visual inspection. If type="arom", only aromatic rings are returned, while type="count" returns the ring and/or aromaticity counts for each compound in a matrix.
arom	if arom="TRUE", ring aromaticity information will be computed. If type="all", the output is a logical vector where 'TRUE' values indicate aromatic rings in the associated ring list. If type="arom", then the function returns only aromatic rings. A ring is considered aromatic if it meets the following requirements: (i) all atoms in the ring need to be sp2 hybridized. This means each atom has to have a double bond or at least one lone electron pair and it needs to be attached to an sp2 hybridized atom. (ii) In addition, Hueckel's rule '4n + 2' needs to be true, where 'n' is either zero or any positive integer.
inner	if inner="TRUE", only inner (smallest possible) rings will be returned. They are identified by first computing all possible rings and then selecting only the inner rings. Note: this requires the setting upper=Inf. If only rings below a certain size limit (e.g. 6) are of interest, then it will be more time efficient to set this limmit under the upper argument than identifying all smallest rings.

Details

...

Value

The settings type="all" and type="arom" return lists, and type="count" returns a matrix.

Author(s)

Thomas Girke

References

Hanser, Jauffret and Kaufmann (1996) A New Algorithm for Exhaustive Ring Perception in a Molecular Graph. Journal of Chemical Information and Computer Sciences, 36: 1146-1152. URL: http://pubs.acs.org/doi/abs/10.1021/ci960322f

See Also

...

Examples

```
## Instances of SDFset class
data(sdfsample)
sdfset <- sdfsample

## Return all possible rings for a single compound
rings(sdfset[1], upper=Inf, type="all", arom=FALSE, inner=FALSE)
plot(sdfset[1], print=FALSE, atomnum=TRUE, no_print_atoms="H")

## Return all possible rings for several compounds plus their
## aromaticity information
rings(sdfset[1:4], upper=Inf, type="all", arom=TRUE, inner=FALSE)

## Return rings with no more than 6 atoms
rings(sdfset[1:4], upper=6, type="all", arom=TRUE, inner=FALSE)

## Return rings with no more than 6 atoms that are also armomatic
rings(sdfset[1:4], upper=6, type="arom", arom=TRUE, inner=FALSE)

## Return shortest possible rings (no complex rings)
rings(sdfset[1:4], upper=Inf, type="all", arom=TRUE, inner=TRUE)

## Count shortest possible rings
rings(sdfset[1:4], upper=Inf, type="count", arom=TRUE, inner=TRUE)
```

SDF-class *Class "SDF"*

Description

Container for storing every element of a single molecule defined in an SD/MOL file without information loss in a list-like container. The import occurs via the `SDFstr` container class. The header block is stored as named character vector, the atom/bond blocks as matrices and the data block as named character vector.

Objects from the Class

Objects can be created by calls of the form `new("SDF", ...)`.

Slots

`header`: Object of class `"character"`

`atomblock`: Object of class `"matrix"`

`bondblock`: Object of class `"matrix"`

`datablock`: Object of class `"character"`

`obmolRef`: Object of class `"ExternalReferenceOrNULL"`

`version`: Object of class `"character"`

Methods

[`signature(x = "SDF")`: subsetting of class with bracket operator

[[`signature(x = "SDF")`: returns one of the four object components

[[<- `signature(x = "SDF")`: replacement method for the four sub-components

[<- `signature(x = "SDF")`: replacement method for the four sub-components

atomblock `signature(x = "SDF")`: returns atom block as matrix

atomcount `signature(x = "SDF")`: returns atom frequency

bondblock `signature(x = "SDF")`: returns bond block as matrix

obmol `signature(x = "SDF")`: returns an OBMol pointer

coerce `signature(from = "character", to = "SDF")`: `as(character, "SDF")`

coerce `signature(from = "list", to = "SDF")`: `as(list, "SDF")`

coerce `signature(from = "SDF", to = "character")`: `as(sdf, "character")`

coerce `signature(from = "SDF", to = "list")`: `as(sdf, "list")`

coerce `signature(from = "SDF", to = "SDFset")`: `as(sdf, "SDFset")`

coerce `signature(from = "SDF", to = "SDFstr")`: `as(SDF, "SDFstr")`

coerce `signature(from = "SDFset", to = "SDF")`: `as(sdfset, "SDF")`

datablock `signature(x = "SDF")`: returns data block as named character vector

datablocktag `signature(x = "SDF")`: returns data block as named character vector with subsetting support

header `signature(x = "SDF")`: returns header block as named character vector

plot `signature(x = "SDF")`: plots molecule structure for `SDF` object

sdf2list `signature(x = "SDF")`: returns `SDF` object as `list`

sdf2str `signature(sdf = "SDF")`: returns `SDF` object as `character` vector

sdfid `signature(x = "SDF")`: returns molecule ID field from header block

show `signature(object = "SDF")`: prints summary of `SDF`

Author(s)

Thomas Girke

References

SDF format definition: http://www.symyx.com/downloads/public/ctfile/ctfile.jsp

See Also

Related classes: SDFset, SDFstr, AP, APset

Examples

```
showClass("SDF")

## Instances of SDF class
data(sdfsample); sdfset <- sdfsample
(sdf <- sdfset[[1]]) # returns first molecule in sdfset as SDF object

## Accessing SDF components
header(sdf); atomblock(sdf); bondblock(sdf); datablock(sdf)
sdfid(sdf)

## Plot molecule structure of SDF
plot(sdf) # plots to R graphics device
# sdf.visualize(sdf) # viewing in browser
```

---

sdf.subset *Subset a SDF and return SDF segements for selected compounds*

Description

'sdf.subset' will take a descriptor database generated by 'cmp.parse' and an array of indices, and return an SDF string consisting of SDFs for compounds corresponding to that list of indices. The returned value is a character string.

Usage

```
sdf.subset(db, cmps)
```

Arguments

db	The database generated by 'cmp.parse'
cmps	An array of indecies that correspond to a set of selected compounds from the database

Details

'sdf.subset' depends on information embedded in the descriptor database returned by 'cmp.parse'. It also relies on the availability of the original SDF where the database has been generated from. Basically, when 'cmp.parse' parses the original SDF file, it will store the path of that SDF file as well as offset information for SDF segment in that file. Therefore, if the SDF file has been changed or deleted, 'sdf.subset' cannot function properly.

The result SDF will also have names added to compounds if they are not present in the original SDF.

Value

Return a character string whose content is the concatenation of SDFs for the selected compounds.

See Also

cmp.parse, sdf.visualize

Examples

```
## Note: this functionality has become obsolete since the introduction of the
## 'SDFset' and 'apset' S4 classes.

# load sample database from web
# db <- cmp.parse("http://bioweb.ucr.edu/ChemMineV2/static/example_db.sdf")
# select SDF for 1st and 2nd compound in that SDF
# sdf_segments <- sdf.subset(db, c(1, 2))
# now sdf_segments containt the 2 SDFs for those 2 compounds
```

sdf.visualize *Visualize an SDFset online using ChemMine Tools*

Description

'sdf.visualize' will take an SDFset object and send the compounds to the ChemMine Tools website, for visualization and futher analysis. The results are launched in the users web browser.

Usage

```
sdf.visualize(sdf)
```

Arguments

sdf	A SDFset object which containing the given compounds

Value

Returns the URL of the webpage containing all the SDFs and 2D images corresponding to the selected compounds.

Author(s)

Tyler Backman

References

ChemMine Tools web service: http://chemmine.ucr.edu

See Also

cmp.parse, sdf.subset, plotStruc

Examples

```
## Load sample SD file
data(sdfsample)
sdfset <- sdfsample

## Not run:
## Plot structures using web service ChemMine Tools
sdf.visualize(sdfset[1:4])

## End(Not run)
```

---

sdf2ap *Atom pair library*

Description

Creates from a SDFset a searchable atom pair library that is stored in a container of class APset.

Usage

```
sdf2ap(sdfset, type = "AP",uniquePairs=TRUE)
```

Arguments

sdfset	Objects of classes SDFset or SDF
type	if type="AP", the function returns APset/AP objects; if type="character", it returns the result as a character vector of length one. The latter is useful for storing AP data in tabular files.
uniquePairs	When the same atom pair occurs more than once in a single compound, should the names be unique or not? Setting this to true will take slightly longer to compute.

Details

...

Value

APset	if input is SDFset
AP	if input is SDF

Author(s)

Thomas Girke

References

Chen X and Reynolds CH (2002). "Performance of similarity measures in 2D fragment-based similarity searching: comparison of structural descriptors and similarity coefficients", J Chem Inf Comput Sci.

See Also

Functions: desc2fp, SDF2apcmp, apset2descdb, cmp.search, cmp.similarity

Related classes: SDF, SDFset, SDFstr, APset.

Examples

```
## Instance of SDFset class
data(sdfsample)
sdfset <- sdfsample[1:50]
sdf <- sdfsample[[1]]

## Compute atom pair library
ap <- sdf2ap(sdf)
(apset <- sdf2ap(sdfset))
view(apset[1:4])

## Return main components of APset object
cid(apset[1:4]) # compound IDs
ap(apset[1:4]) # atom pair descriptors

## Return atom pairs in human readable format
db.explain(apset[1])

## Coerce APset to other objects
apset2descdb(apset) # returns old list-style AP database
tmp <- as(apset, "list") # returns list
as(tmp, "APset") # converst list back to APset

## Compound similarity searching with APset
cmp.search(apset, apset[1], type=3, cutoff=0.2)
plot(sdfset[names(cmp.search(apset, apset[6], type=2, cutoff=0.4))])

## Identify compounds with identical AP sets
cmp.duplicated(apset, type=2)

## Structure similarity clustering
cmp.cluster(db=apset, cutoff = c(0.65, 0.5))[1:20,]
```

SDF2apcmp SDF *to* list *for AP generation*

Description

Returns SDF class as list containing the components for generating atom pair descriptors.

Usage

```
SDF2apcmp(SDF)
```

Arguments

SDF	SDF

Details

...

Value

`list`	with atom and bond components

Author(s)

Thomas Girke

References

Chen X and Reynolds CH (2002). "Performance of similarity measures in 2D fragment-based similarity searching: comparison of structural descriptors and similarity coefficients", J Chem Inf Comput Sci.

See Also

Functions: `sdf2ap`, `apset2descdb`, `cmp.search`, `cmp.similarity`

Examples

```
## Instances of SDFset class
data(sdfsample)
sdf <- sdfsample[[1]]

## Return list
cmp <- SDF2apcmp(sdf)
```

`sdf2list` `SDF` *to* `list`

Description

Returns objects of class `SDF` as `list`.

Usage

```
sdf2list(x)
```

Arguments

`x`	object of class `SDF`

Details

...

Value

list	with following components:
character	SDF header block
matrix	SDF bond block
matrix	SDF atom block
character	SDF data block

Author(s)

Thomas Girke

References

SDF format definition: http://www.symyx.com/downloads/public/ctfile/ctfile.jsp

See Also

Functions: `sdfstr2list`, `sdf2str`, `SDFset2list`, `SDFset2SDF`

Examples

```
## Instance of SDF class
data(sdfsample); sdfset <- sdfsample
sdf <- sdfset[[1]]

## Return as list
sdf2list(sdf)
as(sdf, "list") # similar result
```

`sdf2smiles` `SDFset` *to* `character` *Convert* `SDFset` *to SMILES (*`character`*)*

Description

Accepts compounds in an `SDFset` container and returns the corresponding SMILES (Simplified Molecular Input Line Entry Specification) strings as `SMIset` object. If ChemineOB is available then OpenBabel for the format conversion. Otherwise the compound is submitted to the ChemMine Tools web service for conversion with the Open Babel Open Source Chemistry Toolbox. If the input object contains multiple items, only the first is converted.

Usage

```
sdf2smiles(sdf)
```

Arguments

sdf	A `SDFset` object which containing the given compounds

Value

character	for details see ?"character"

Author(s)

Tyler Backman, Kevin Horan

References

Chemmine web service: http://chemmine.ucr.edu

Open Babel: http://openbabel.org

SMILES Format: http://en.wikipedia.org/wiki/Chemical_file_format#SMILES

Examples

```
## Not run:
## get a sample compound
data(sdfsample); sdfset <- sdfsample[1]
## convert to smiles
(smiles <- sdf2smiles(sdfset))
as.character(smiles)

## End(Not run)
```

---

sdf2str SDF *to* SDFstr

Description

Converts SDF to SDFstr. Its main use is to facilitate the export to SD files. It contains optional arguments to generate custom SDF output.

Usage

```
sdf2str(sdf, head, ab, bb, db, cid = NULL, sig = FALSE, ...)
```

Arguments

sdf	object of class SDF
head	optional character vector to supply custom header block
ab	optional matrix to supply custom atom block
bb	optional matrix to supply custom bond block
db	optional character vector to supply custom data block
cid	character can be provided to inject custom compound ID into header block
sig	if = TRUE then the ChemmineR signature will be injected into the header block for tracking purposes
...	option to pass on additional arguments

Details

If the export function write.SDF is supplied with an SDFset object, then sdf2str is used internally to customize the export of many molecules to a single SD file using the same optional arguments.

Value

sdfstr SDF data of one molecule collapsed to character vector

Author(s)

Thomas Girke

References

SDF format definition: http://www.symyx.com/downloads/public/ctfile/ctfile.jsp

See Also

Coerce functions: sdfstr2list, sdf2str, SDFset2list, SDFset2SDF

Export function: write.SDF

Examples

```
## Instance of SDF class
data(sdfsample); sdfset <- sdfsample
sdf <- sdfset[[1]]

## Customize SDF blocks for export to SD file
sdf2str(sdf=sdf, sig=TRUE, cid=TRUE) # uses default SDF components
sdf2str(sdf=sdf, head=letters[1:4], db=NULL) # uses custom components for header and datablock

## The same arguments can be supplied to the write.SDF function for
## batch export of custom SDFs
# write.SDF(sdfset[1:4], file="sub.sdf", sig=TRUE, cid=TRUE, db=NULL)
```

---

sdfid *Return SDF compound IDs*

Description

Returns the compound identifiers from the header block of SDF or SDFset objects.

Usage

```
sdfid(x, tag = 1)
```

Arguments

x object of class SDFset or SDF

tag values from 1-4 to extract different header block fields; SDF ID is in first one (default)

Details

...

Value

character vector

Author(s)

Thomas Girke

References

...

See Also

atomblock, atomcount, bondblock, datablock, header, cid

Examples

```
## SDF/SDFset instances
data(sdfsample)
sdfset <- sdfsample
sdf <- sdfset[[1]]

## Extract IDs from header block
sdfid(sdf, tag=1)
sdfid(sdfset[1:4])

## Extract compound IDs from ID slot in SDFset container
cid(sdfset[1:4])

## Assigning compound IDs and keeping them unique
unique_ids <- makeUnique(sdfid(sdfset))
cid(sdfset) <- unique_ids
cid(sdfset[1:4])
```

---

sdfsample *SD file in* SDFset *object*

Description

First 100 compounds from PubChem SD file: Compound_00650001_00675000.sdf.gz

Usage

```
data(sdfsample)
```

Format

Object of class sdfset

Details

Object stores 100 molecules from a sample SD file.

Source

ftp://ftp.ncbi.nih.gov/pubchem/Compound/CURRENT-Full/SDF/Compound_00650001_00675000.sdf.gz

References

SDF format definition: http://www.symyx.com/downloads/public/ctfile/ctfile.jsp

Examples

```
data(sdfsample)
sdfset <- sdfsample
view(sdfset[1:4])
```

SDFset-class *Class "SDFset"*

Description

List-like container for storing one or many objects of class SDF each containing the structure definition information of molecules provided by an SD/MOL file. The SDFset is the most important class in the ChemmmineR package for accessing and manipulating information stored in SD files.

Objects from the Class

Objects can be created by calls of the form `new("SDFset", ...)`.

Slots

`SDF`: Object of class `"list"` storing SDF components

`ID`: Object of class `"character"` storing compound identifiers

Methods

[`signature(x = "SDFset")`: subsetting of class with bracket operator

[[`signature(x = "SDFset")`: returns single component as SDF object

[[<- `signature(x = "SDFset")`: replacement method for single SDF component

[<- `signature(x = "SDFset")`: replacement method for several SDF components

atomblock `signature(x = "SDFset")`: returns all atom blocks as list

atomcount `signature(x = "SDFset")`: returns all atom frequencies as list

bondblock `signature(x = "SDFset")`: returns all bond blocks as list

obmol `signature(x = "SDFset")`: returns pointers to OBMol objects as a vector

c `signature(x = "SDFset")`: concatenates two SDFset containers

cid `signature(x = "SDFset")`: returns all compound identifiers from ID slot

header<- `signature(x = "SDFset")`: replacement method for header block

atomblock<- `signature(x = "SDFset")`: replacement method for atom block

bondblock<- `signature(x = "SDFset")`: replacement method for bond block

datablock<- `signature(x = "SDFset")`: replacement method for data block

coerce `signature(from = "list", to = "SDFset")`: `as(list, "SDFset")`

coerce `signature(from = "SDF", to = "SDFset")`: `as(sdf, "SDFset")`

coerce `signature(from = "SDFset", to = "list")`: `as(sdfset, "list")`

coerce `signature(from = "SDFset", to = "SDF")`: `as(sdfset, "SDF")`

coerce `signature(from = "SDFset", to = "SDFstr")`: `as(sdfset, "SDFstr")`

coerce `signature(from = "SDFstr", to = "SDFset")`: `as(sdfstr, "SDFset")`

datablock `signature(x = "SDFset")`: returns all data blocks as list

datablocktag `signature(x = "SDFset")`: returns all data blocks as named as list with subsetting support

header `signature(x = "SDFset")`: returns all header blocks as list

length `signature(x = "SDFset")`: returns number of entries stored in object

plot `signature(x = "SDFset")`: plots one or many molecule structures from `SDFset` object

sdfid `signature(x = "SDFset")`: returns molecule ID field from header block

SDFset2list `signature(x = "SDFset")`: returns `SDFset` object as `list`

SDFset2SDF `signature(x = "SDFset")`: returns `SDFset` object as `list` with `SDF` components

SDFset2SDF<- `signature(x = "SDFset")`: replacement method for `SDFset` component in `SDFset` using accessor method

show `signature(object = "SDFset")`: prints summary of `SDFset`

view `signature(x = "SDFset")`: prints extended summary of `SDFset`

SDFset `SDFset(SDF, ID)`: interface to `SDFset` constructor

Author(s)

Thomas Girke

References

SDF format definition: http://www.symyx.com/downloads/public/ctfile/ctfile.jsp

See Also

Related classes: SDF, SDFstr, AP, APset

Import function: read.SDFset("some_SDF_file")

Export function: write.SDF(sdfset, "some_file.sdf")

Examples

```
showClass("SDFset")

## Instances of SDFset class
data(sdfsample); sdfset <- sdfsample
sdfset; view(sdfset[1:4])
sdfset[[1]]

## Import and store SD File in SDFset container
# sdfset <- read.SDFset("some_SDF_file")

## Miscellaneous accessor methods
```

```
header(sdfset[1:4])
atomblock(sdfset[1:4])
atomcount(sdfset[1:4])
bondblock(sdfset[1:4])
datablock(sdfset[1:4])

## Assigning compound IDs and keeping them unique
cid(sdfset); sdfid(sdfset)
unique_ids <- makeUnique(sdfid(sdfset))
cid(sdfset) <- unique_ids

## Convert data block to matrix
blockmatrix <- datablock2ma(datablocklist=datablock(sdfset)) # Converts data block to matrix
numchar <- splitNumChar(blockmatrix=blockmatrix) # Splits to numeric and character matrix
numchar[[1]][1:4,]; numchar[[2]][1:4,]

## Compute atom frequency matrix, molecular weight and formula
propma <- data.frame(MF=MF(sdfset), MW=MW(sdfset), atomcountMA(sdfset))
propma[1:4, ]

## Assign matrix data to data block
datablock(sdfset) <- propma
view(sdfset[1:4])

## String Searching in SDFset
grepSDFset("650001", sdfset, field="datablock", mode="subset") # To return index, set mode="index")

## Export SDFset to SD file
# write.SDF(sdfset[1:4], file="sub.sdf", sig=TRUE)

## Plot molecule structure of SDF
plot(sdfset[1:4]) # plots to R graphics device
# sdf.visualize(sdfset[1:4]) # viewing in browser
```

SDFset2list SDFset *to* list

Description

Returns object of class SDFset as list where each component conists of a list of the four SDF sub-components: header block, atom block, bond block and data block.

Usage

```
SDFset2list(x)
```

Arguments

x	object of class SDFset

Details

...

Value

`list` containing one or many lists each with following components:

`character` SDF header block

`matrix` SDF bond block

`matrix` SDF atom block

`character` SDF data block

Author(s)

Thomas Girke

References

SDF format definition: http://www.symyx.com/downloads/public/ctfile/ctfile.jsp

See Also

Functions: `sdfstr2list`, `sdf2str`, `sdf2list`, `SDFset2SDF`

Examples

```
## Instance of SDFset class
data(sdfsample); sdfset <- sdfsample
sdfset

## Returns sdfset as list
SDFset2list(sdfset[1:4])
as(sdfset, "list")[1:4] # similar result
```

`SDFset2SDF` `SDFset` *to list with many* `SDF`

Description

Returns object of class `SDFset` as `list` were each component consists of an `SDF` object.

Usage

```
SDFset2SDF(x)
```

Arguments

`x` object of class `SDFset`

Details

...

Value

`list` containing one or many `SDF` objects

Author(s)

Thomas Girke

References

SDF format definition: http://www.symyx.com/downloads/public/ctfile/ctfile.jsp

See Also

Functions: `sdfstr2list`, `sdf2str`, `sdf2list`, `SDFset2list`

Examples

```
## Instance of SDFset class
data(sdfsample); sdfset <- sdfsample
sdfset

## Returns sdfset as list
SDFset2SDF(sdfset[1:4])
as(sdfset, "SDF")[1:4] # similar result
view(sdfset[1:4]) # same result
```

`SDFstr-class` *Class "SDFstr"*

Description

List-like container for storing one or many molecules from an SD (or MOL) file. Each component of an `SDFstr` object stores the SD data line by line from a single molecule in a character vector. The `SDFstr` class is an intermediate container to import SD files into the more important `SDFset` object or to export the data back from an `SDFset` container to a valid SD file.

Objects from the Class

Objects can be created by calls of the form `new("SDFstr", ...)`.

Slots

`a`**:** Object of class `"list"` with `character` components

Methods

[`signature(x = "SDFstr")`: subsetting of class with bracket operator

[[`signature(x = "SDFstr")`: returns single component as character vector

[[<- `signature(x = "SDFstr")`: replacement method for single `SDFstr` component

[<- `signature(x = "SDFstr")`: replacement method for several `SDFstr` components

coerce `signature(from = "character", to = "SDFstr")`: `as(character, "SDFstr")`

coerce `signature(from = "list", to = "SDFstr")`: `as(list, "SDFstr")`

coerce `signature(from = "SDF", to = "SDFstr")`: `as(sdf, "SDFstr")`

coerce `signature(from = "SDFset", to = "SDFstr")`: `as(sdfset, "SDFstr")`

coerce `signature(from = "SDFstr", to = "list")`: `as(sdfstr, "list")`

coerce `signature(from = "SDFstr", to = "SDFset")`: `as(sdfstr, "SDFset")`

length `signature(x = "SDFstr")`: returns length of `SDFstr`

sdfstr2list `signature(x = "SDFstr")`: accessor method to return `SDFstr` as `list`

sdfstr2list<- `signature(x = "SDFstr")`: replacement method for several `SDFstr` components

show `signature(object = "SDFstr")`: prints summary of `SDFstr`

Author(s)

Thomas Girke

References

SDF format definition: http://www.symyx.com/downloads/public/ctfile/ctfile.jsp

See Also

Related classes: SDFset, AP, APset

Import function: read.SDFstr("some_SDF_file")

Examples

```
showClass("SDFstr")

## Instances of SDFstr class
data(sdfsample); sdfset <- sdfsample
sdfstr <- as(sdfset, "SDFstr")
sdfstr[1:4] # print summary of container content
sdfstr[[1]] # returns character vector

## Import: sdfstr <- read.SDFstr("some_SDF_file")
## Export: write.SDF(sdfstr, "some_file.sdf")
```

`sdfstr2list` `SDFstr` *to* `list`

Description

Returns objects of class `SDFstr` as `list`.

Usage

```
sdfstr2list(x)
```

Arguments

`x`	object of class `SDFstr`

Details

...

Value

`list` with many of the following components:

`character` SDF content of one molecule vectorized line by line

Author(s)

Thomas Girke

References

SDF format definition: http://www.symyx.com/downloads/public/ctfile/ctfile.jsp

See Also

Functions: `sdf2list`, `sdf2str`, `SDFset2list`, `SDFset2SDF`

Examples

```
## Instance of SDFstr class
data(sdfsample); sdfset <- sdfsample
sdfstr <- as(sdfset, "SDFstr")

## Return as list
sdfstr2list(sdfstr)
as(sdfstr, "list") # similar result
```

`sdfStream` *Streaming through large SD files*

Description

Streaming function to compute descriptors for large SD Files without consuming much memory. In addition to descriptor values, it returns a line index that defines the positions of each molecule in the source SD File. This line index can be used by the `read.SDFindex` function to retrieve specific compounds of interest from large SD Files without reading the entire file into memory.

Usage

```
sdfStream(input, output, append=FALSE, fct, Nlines = 10000, startline=1, restartNlines=10000, sile
```

Arguments

`input` file name of input SD file

`output` file name of tabular descriptor file

`append` if `append=FALSE`, a new output file will be created, if one with the same name exists it will be overwritten; whereas `append=TRUE` will appended to this file.

`fct` Function to select descriptor sets; any combination of descriptors, supported by `ChemmineR`, can be chosen here, as long as they can be represented in tabular format.

`Nlines` Number of lines to read from input SD File at a time; the memory consumption will be proportional to this value.

`startline` For restarting sdfStream at specific line assigned to `startline` argument. If assigned `startline` value does not match the first line of a molecule in the SD file then it will be reset to the start position of the next molecule in the SD file.

`restartNlines` Number of lines to parse when `startline > 1` in order to identify proper molecule start position. The default value of 10,000 is usually a good choice.

`silent` if `silent=FALSE`, the processing status will be printed to the screen, while `silent=TRUE` suppresses this output.

`...` Arguments to be passed to/from other methods.

Details

...

Value

Writes a descriptor matrix to a tabular file. The first and last line number (position index) of each molecule is specified in the first two columns of the tabular output file, respectively.

Author(s)

Thomas Girke

References

SDF format definition: http://www.symyx.com/downloads/public/ctfile/ctfile.jsp

See Also

Import/export functions: `read.AP`, `read.SDFset`, `read.SDFstr`, `read.SDFstr`, `read.SDFset`, `write.SDFsplit`

Examples

```
## Load sample data
library(ChemmineR)
data(sdfsample); sdfset <- sdfsample
## Not run: write.SDF(sdfset, "test.sdf")

## Define descriptor set in a simple function
desc <- function(sdfset) {
        cbind(SDFID=sdfid(sdfset),
              # datablock2ma(datablocklist=datablock(sdfset)),
              MW=MW(sdfset),
              groups(sdfset),
              # AP=sdf2ap(sdfset, type="character"),
              rings(sdfset, type="count", upper=6, arom=TRUE)
        )
}

## Run sdfStream with desc function and write results to a file called 'matrix.xls'
```

```
sdfStream(input="test.sdf", output="matrix.xls", append=FALSE, fct=desc, Nlines=1000)

## Same as before but starting in SD file at line number 950
sdfStream(input="test.sdf", output="matrix.xls", append=FALSE, fct=desc, Nlines=1000, startline=950)

## Select molecules from SD File using line index from sdfStream
indexDF <- read.delim("matrix.xls", row.names=1)[,1:4]
indexDFsub <- indexDF[indexDF$MW < 400, ] # Selects molecules with MW < 400
sdfset <- read.SDFindex(file="test.sdf", index=indexDFsub, type="SDFset")

## Write result directly to SD file without storing larger numbers of molecules in memory
read.SDFindex(file="test.sdf", index=indexDFsub, type="file", outfile="sub.sdf")

## Read atom pair string representation from file into APset
apset <- read.AP(file="matrix.xls", colid="AP")
cid(apsdf) <- as.character(indexDF$SDFID)

## End(Not run)
```

searchSim *PubChem Similarity (Fingerprint) Search*

Description

Accepts one SDFset container and performs a >0.9 similarity PubChem fingerprint search, returning up to 200 hits in an SDFset container. The ChemMine Tools web service is used as an intermediate, to translate queries from plain HTTP POST to a PubChem Power User Gateway (PUG) query. If the input object contains multiple items, only the first is used as a query.

Usage

```
searchSim(sdf)
```

Arguments

sdf	A SDFset object which contains one compound

Value

SDFset	for details see ?"SDFset-class"

Author(s)

Tyler Backman

References

PubChem PUG SOAP: http://pubchem.ncbi.nlm.nih.gov/pug_soap/pug_soap_help.html

Chemmine web service: http://chemmine.ucr.edu

PubChem help: http://pubchem.ncbi.nlm.nih.gov/search/help_search.html

SMILES Format: http://en.wikipedia.org/wiki/Chemical_file_format#SMILES

Examples

```
## Not run:
## get a sample compound
data(sdfsample); sdfset <- sdfsample[2]
## search a compound on PubChem
compounds <- searchSim(sdfset)
## End(Not run)
```

---

searchString *PubChem Similarity (Fingerprint) SMILES Search*

Description

Accepts one SMILES string (Simplified Molecular Input Line Entry Specification) and performs a >0.95 similarity PubChem fingerprint search, returning the hits in an SDFset container. The ChemMine Tools web service is used as an intermediate, to translate queries from plain HTTP POST to a PubChem Power User Gateway (PUG) query.

Usage

```
searchString(smiles)
```

Arguments

smiles	A character object which contains one SMILES string

Value

SDFset	for details see ?"SDFset-class"

Author(s)

Tyler Backman

References

PubChem PUG SOAP: http://pubchem.ncbi.nlm.nih.gov/pug_soap/pug_soap_help.html

Chemmine web service: http://chemmine.ucr.edu

PubChem help: http://pubchem.ncbi.nlm.nih.gov/search/help_search.html

SMILES Format: http://en.wikipedia.org/wiki/Chemical_file_format#SMILES

Examples

```
## Not run:
## search a compound on PubChem
compounds <- searchString("CC(=O)OC1=CC=CC=C1C(=O)O")
## End(Not run)
```

selectInBatches *Select in Batches*

Description

When doing a select were the condition is a large number of ids it is not always possible to include them in a single SQL statement. This function will break the list of ids into chunks and send the query for each batch. The resutls are appended and returned as one data frame.

Usage

```
selectInBatches(conn, allIndices, genQuery, batchSize = 1e+05)
```

Arguments

`conn`	Database connection object
`allIndices`	A vector of indices to pass to the genQuery function in batches.
`genQuery`	A function which takes a vector of indices and constructs an SQL SELECT statement returning records for the given indicies.
`batchSize`	How many indicies to put in each batch.

Value

A data frame with the results of the query as if all inidices had been included in a single SELEcT statement.

Author(s)

Kevin Horan

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (conn, allIndices, genQuery, batchSize = 1e+05)
{
    batchByIndex(allIndices, function(indexBatch) {
        df = dbGetQuery(conn, genQuery(indexBatch))
        result = rbind(result, df)
    }, batchSize)
    result
  }
```

setPriorities *Set Priorities*

Description

This function should be run after loading a complete set of data. It will find each group of compounds which share the same descriptor and call the given function, `priorityFn`, with the compound_id numbers of the group. This function should then assign priorities to each compound-descriptor pair, however it wishes. Priorities are integer values with lower values being used in preference of higher values.

It is important that this function be called after all data is loaded. It may be that a compound loaded at the beginning of a data set shares a descriptor with a compound loaded near the end of the data set. If the priorities were set at some point in between these then it would not see all the compounds for that one descriptor.

If a SNOW cluster and connection source function are given, it will run in parallel.

Some pre-defined functions that can be use for `priorityFn` are:

`randomPriorities`: Set the priorities of compounds within a descriptor group randomly.

`forestSizePriorities`: Set the priority based on the number of disconnected components (trees) within the compound. Compounds with fewer trees will have a higher priority (lower numerical value) than compounds with more trees.

Usage

```
setPriorities(conn,priorityFn,descriptorIds=c(),cl=NULL,connSource=NULL)
forestSizePriorities(conn,compIds)
randomPriorities(conn,compIds)
```

Arguments

`conn` A database connection object.

`priorityFn` This function will be called with the compound_id numbers associated with the same descriptor. It should use the id numbers to lookup whatever data it wants to assign a priority to each compound. These priority values will be used to pick a compound to represent the group in cases where only one compound is needed for each descriptor.

The function should return a data.frame with the fields "compound_id" and "priority". The order of the rows is not important.

`descriptorIds` If given then only re-compute priorities for groups involving descriptors in this list. This is useful for updating priorities after adding new compounds to an existing database.

`cl` A SNOW cluster on which to run jobs on.

`connSource` A function to create a new database connection with. This will be run once for each new job created. It must return a newly created connection, not a reference to an existing connection.

`compIds` The compound_id values for each group.

Value

For `setPriorities`, no value is returned. `randomPriorities` and `forestSizePriorities` return a data.frame with columns "compound_id" and "priority".

Author(s)

Kevin Horan

Examples

```
## Not run:
data(sdfsample)
conn = initDb("sample.db")
sdfLoad(conn,sdfsample)
setPriorities(conn,forestSizePriorities)

## End(Not run)
```

`smartsSearchOB` *SMARTS Search OB*

Description

Perform searches for SMARTS patterns using Open Babel (requires ChemmineOB package to be installed).

Usage

```
smartsSearchOB(sdfset, smartsPattern, uniqueMatches = TRUE)
```

Arguments

`sdfset`	An SDFset of the compounds you want to search
`smartsPattern`	The SMARTS pattern as a string.
`uniqueMatches`	If true, only return the number of distinct matches, otherwise return the number of all matches.

Value

Returns a vector of counts, one for each input compound.

Author(s)

Kevin Horan

Examples

```
## Not run:
library(ChemmineOB)
data(sdfsample)
#look for rotable bonds
rotableBonds = smartsSearchOB(sdfsample[1:5],"[!$(*#*)&!D1]-!@[!$(*#*)&!D1]",uniqueMatches=FALSE)

## End(Not run)
```

SMI-class *Class* "SMI"

Description

Container for storing the SMILES string of a single molecule.

Objects from the Class

Objects can be created by calls of the form new("SMI", ...).

Slots

smiles: Object of class "character" of length one

Methods

as.character signature(x = "SMI"): returns content as character vector

coerce signature(from = "character", to = "SMI"): as(smi, "SMI")

coerce signature(from = "SMIset", to = "SMI"): as(smiset, "SMI")

show signature(object = "SMI"): prints summary of SMI

Author(s)

Thomas Girke

References

SMILES (Simplified molecular-input line-entry system) format definition: http://en.wikipedia.org/wiki/Simplified_molec input_line-entry_system

See Also

Related classes: SMIset, SDF, SDFset

Examples

```
showClass("SMI")

## Instances of SMI class
data(smisample); smiset <- smisample
(smi <- smiset[[1]]) # returns first molecule in smiset as SMI object
```

smiles2sdf *Convert SMILES (*`character`*) to* `SDFset`

Description

Accepts a named vector or `SMIset` of SMILES (Simplified Molecular Input Line Entry Specification) strings and returns its equivalent as an `SDFset` container.

This function runs in two modes. If ChemmineOB is available then it will use OpenBabel to convert all the given smiles into an SDFset with 2D coordinates. Otherwise the compound is submitted to the ChemMine Tools web service for conversion with the Open Babel Open Source Chemistry Toolbox. In this case only the first element will be used since this is a very slow operation.

Usage

```
smiles2sdf(smiles)
```

Arguments

`smiles`	A named vector of SMILES strings. The names will be used to name the SDF objects.

Value

`SDFset`	for details see ?"SDFset-class"

Author(s)

Tyler Backman, Kevin Horan

References

Chemmine web service: http://chemmine.ucr.edu

Open Babel: http://openbabel.org

SMILES Format: http://en.wikipedia.org/wiki/Chemical_file_format#SMILES

Examples

```
## Not run:
## convert to sdf
data(smisample)
(sdf <- smiles2sdf(smisample[1:4]))

## End(Not run)
```

smisample *SMILES file in* SMIset *object*

Description

First 100 compounds from PubChem SD file (Compound_00650001_00675000.sdf.gz) converted to SMILES format

Usage

```
data(smisample)
```

Format

Object of class `smiset`

Details

Object stores 100 molecules from a sample SMILES file.

Source

ftp://ftp.ncbi.nih.gov/pubchem/Compound/CURRENT-Full/SDF/Compound_00650001_00675000.sdf.gz

References

SMILES (Simplified molecular-input line-entry system) format definition: http://en.wikipedia.org/wiki/Simplified_molec input_line-entry_system

Examples

```
data(smisample)
smiset <- smisample
view(smiset[1:4])
```

SMIset-class *Class* "SMIset"

Description

List-like container for storing SMILES strings of many compounds.

Objects from the Class

Objects can be created by calls of the form `new("SMIset", ...)`.

Slots

`smilist`: Object of class `"list"` with compound identifiers stored in name slots

Methods

[signature(x = "SMIset"): subsetting of class with bracket operator

[[signature(x = "SMIset"): returns single component as SMI object

[<- signature(x = "SMIset"): replacement method for one or many entries

as.character signature(x = "SMIset"): returns content as named character vector

c signature(x = "SMIset"): concatenates two SMIset containers

cid signature(x = "SMIset"): returns compound identifiers

cid<- signature(x = "SMIset"): replacement method for compound identifiers

coerce signature(from = "character", to = "SMIset"): as(character, "SMIset")

coerce signature(from = "list", to = "SMIset"): as(list, "SMIset")

coerce signature(from = "SMIset", to = "SMI"): as(smiset, "SMI")

length signature(x = "SMIset"): returns number of entries stored in object

show signature(object = "SMIset"): prints summary of SMIset

view signature(x = "SMIset"): prints extended summary of SMIset

Author(s)

Thomas Girke

References

SMILES (Simplified molecular-input line-entry system) format definition: http://en.wikipedia.org/wiki/Simplified_molec input_line-entry_system

See Also

Related classes: SMI, SDF, SDFset

Import function: read.SMIset("some_SMILES_file")

Export function: write.SMI(smiset, "some_file.smi")

Examples

```
showClass("SMIset")

## Instances of SMIset class
data(smisample); smiset <- smisample
smiset; view(smiset[1:4])
smiset[[1]]

## Import and store SMILES file in SMIset container
# smiset <- read.SMIset("some_SMILES_file")

## Miscellaneous accessor methods
cid(smiset[1:4])
(smivec <- as.character(smiset[1:4]))

## Construct SMIset from named vector
as(smivec, "SMIset")

## Assigning compound IDs and keeping them unique
```

```
unique_ids <- makeUnique(cid(smiset))
cid(smiset) <- unique_ids

## Export SMIset to SMILES file
# write.SMI(smiset[1:4], file="sub.smi", cid=TRUE)
```

status *Get Status of a ChemMine Tools Job*

Description

Returns the status of a launched ChemMine Tools job as represented by a jobToken object.

Usage

```
status(object)
```

Arguments

object	A jobToken job as returned by the function launchCMTool

Value

The status of the specified job is returned as a string. Possible values include "RUNNING", "FINISHED", or "FAILED".

Author(s)

Tyler William H Backman

References

See ChemMine Tools at http://chemmine.ucr.edu.

See Also

Functions: toolDetails, listCMTools, launchCMTool, browseJob, result

Examples

```
## Not run:
## list available tools
listCMTools()

## get detailed instructions on using a tool
toolDetails("Fingerprint Search")

## download compound 2244 from PubChem
job1 <- launchCMTool("pubchemID2SDF", 2244)

## check job status and download result
status(job1)
result1 <- result(job1)
```

```
## End(Not run)
```

toolDetails *Detailed instructions for each ChemMine Tools web tool*

Description

Connects to the ChemMine Tools web service, and provides detailed instructions and example function calls for any tool.

Usage

```
toolDetails(tool_name)
```

Arguments

tool_name	A tool name matching verbatim an existing tool name as listed by listCMTools.

Details

Prints instructions to console.

Author(s)

Tyler William H Backman

References

See ChemMine Tools at http://chemmine.ucr.edu.

See Also

Functions: launchCMTool, listCMTools, result, browseJob, status

Examples

```
## Not run:
## list available tools
listCMTools()

## get detailed instructions on using a tool
toolDetails("Fingerprint Search")

## download compound 2244 from PubChem
job1 <- launchCMTool("pubchemID2SDF", 2244)

## check job status and download result
status(job1)
result1 <- result(job1)

## End(Not run)
```

trimNeighbors *Trim Neighbors*

Description

Further reduce the cutoff value of a nearest neighbor (NN) table, as produced by `nearestNeighbors`. This allows one to compute a very relaxed NN table initially, and then quickly restrict it later without having to re-compute all the similarities.

Usage

```
trimNeighbors(nnm, cutoff)
```

Arguments

`nnm`	A nearest neighbor table, as produced by `nearestNeighbors`.
`cutoff`	The new similarities cutoff value. All pairs with a similarity less than this value will be removed from the table.

Value

The return value has the same structure as `nnm`, with some neighbors removed from the `indexes` and `similarties` entries.

Author(s)

Kevin Horan

See Also

`jarvisPatrick` `nearestNeighbors`

Examples

```
data(sdfsample)
ap = sdf2ap(sdfsample)
nnm = nearestNeighbors(ap,numNbrs=20)
nnm = trimNeighbors(nnm,cutoff=0.5)
clustering = jarvisPatrick(nnm,k=2,mode="a1b")
```

validSDF *Validity check of SDFset*

Description

Performs validity check of SDFs stored in SDFset objects. Currently, the function tests whether the atom block and the bond block in each SDF component of an SDFset have at least Nabcol and Nbbcol columns (default is 3 for both). In additions, it tests for the presence of NA values in the atom and bond blocks. The function returns a logical vector with TRUE values for valid compounds and FALSE values for invalid ones.

Usage

```
validSDF(x, Nabcol = 3, Nbbcol = 3, logic = "&", checkNA=TRUE)
```

Arguments

x	x object of class SDFset
Nabcol	minimum number of columns in atom block
Nbbcol	minimum number of columns in bond block
logic	logical connection (& or |) among Nabcol and Nbbcol cutoffs
checkNA	checks for NA values in atom and bond blocks

Details

The function is important to remove invalid compounds from SDFset containers.

Value

logical vector of length x with TRUE for valid compounds and FALSE for invalid compounds.

Author(s)

Thomas Girke

References

...

See Also

Functions: read.SDFset

Examples

```
## SDFset instance
data(sdfsample)
sdfset <- sdfsample

## Detect and remove invalid SDFs in SDFset.
valid <- validSDF(sdfset)
which(!valid) # Returns index for invalid SDFs
sdfset <- sdfset[valid] # Returns only valid SDFs.
```

view *Viewing of complex objects*

Description

Convenience function for viewing the content of complex objects like SDFset and APset containers. The function is a shorthand wrapper for `as(sdfset, "SDF")` and `as(apset, "AP")`.

Usage

```
view(x)
```

Arguments

`x`	object of class `SDFset` or `APset`

Details

...

Value

List populated with `SDF` and `AP` components.

Author(s)

Thomas Girke

References

...

See Also

Classes: SDF, SDFset, AP, APset

Examples

```
## Viewing content of SDFset
data(sdfsample); sdfset <- sdfsample
view(sdfset[1:4])

## Viewing content of APset
apset <- sdf2ap(sdfset[1:10])
view(apset)
```

write.SDF *SDF export function*

Description

Writes one or many molecules stored in a `SDFset`, `SDFstr` or `SDF` object to SD file.

Usage

```
write.SDF(sdf, file, cid = FALSE, ...)
```

Arguments

`sdf`	object of class `SDFset`, `SDFstr` or `SDF`
`file`	name of SD file to write to
`cid`	if `cid = TRUE` and an `SDFset` object is provide as input, then the compound IDs in the ID slot of the `SDFset` are used for compound naming
`...`	the optional arguments of the `sdf2str` function can be provided here, including `head, ab, bb, db`; details are provided in the help page for the `sdf2str` function

Details

If the `write.SDF` function is supplied with an `SDFset` object, then it uses internally the `sdf2str` function to allow customizing the resulting SD file. For this all optional arguments of the `sdf2str` function can be passed on to `write.SDF`.

Author(s)

Thomas Girke

References

SDF format definition: http://www.symyx.com/downloads/public/ctfile/ctfile.jsp

See Also

Import function: `read.SDFset`, `read.SDFstr`

Examples

```
## Instance of SDFset class
data(sdfsample); sdfset <- sdfsample

## Write objects of classes SDFset/SDFstr/SDF to file
# write.SDF(sdfset[1:4], file="sub.sdf")

## Example for writing customized SDFset to file containing
## ChemmineR signature, IDs from SDFset and no data block
# write.SDF(sdfset[1:4], file="sub.sdf", sig=TRUE, cid=TRUE, db=NULL)

## Example for injecting a custom matrix/data frame into the data block of an
## SDFset and then writing it to an SD file
props <- data.frame(MF=MF(sdfset), MW=MW(sdfset), atomcountMA(sdfset))
datablock(sdfset) <- props
view(sdfset[1:4])
# write.SDF(sdfset[1:4], file="sub.sdf", sig=TRUE, cid=TRUE)
```

write.SDFsplit *SDF split function*

Description

Splits SD Files into any number of smaller SD Files

Usage

```
write.SDFsplit(x, filetag, nmol)
```

Arguments

x	object of class SDFset, SDFstr
filetag	string to prepend to file names
nmol	integer specifying number of molecules in split SD files

Details

To split an SD File into smaller ones, one can read the source file into R with `read.SDFstr` and write out smaller ones with `write.SDFsplit`. Note: when importing big SD Files, `read.SDFstr` will be much faster than `read.SDFset`, and there is no need to go through an SDFset object instance in this case.

Author(s)

Thomas Girke

References

SDF format definition: http://www.symyx.com/downloads/public/ctfile/ctfile.jsp

See Also

Import/export functions: `read.SDFset`, `read.SDFstr`, `read.SDFstr`, `read.SDFset`

Examples

```
## Load sample data
library(ChemmineR)
data(sdfsample)

## Not run: ## Create sample SD File with 100 molecules
write.SDF(sdfsample, "test.sdf")

## Read in sample SD File
sdfstr <- read.SDFstr("test.sdf")
```

```
## Run export on SDFstr object
write.SDFsplit(x=sdfstr, filetag="myfile", nmol=10)

## Run export on SDFset object
write.SDFsplit(x=sdfsample, filetag="myfile", nmol=10)


## End(Not run)
```

`write.SMI` *SMI export function*

Description

Writes one or many molecules stored in a `SMIset` object to a SMILES file.

Usage

```
write.SMI(smi, file, cid = TRUE, ...)
```

Arguments

`smi` object of class `SMIset`

`file` name of SMILES file to write to

`cid` if `cid = TRUE` the compound identifiers will be exported by appending them in tab-separated format to each SMILES string

`...` option to pass on additional arguments

Details

...

Author(s)

Thomas Girke

References

SMILES (Simplified molecular-input line-entry system) format definition: http://en.wikipedia.org/wiki/Simplified_molec
input_line-entry_system

See Also

Functions: `write.SDF`

Examples

```
## Instance of SMIset class
data(smisample); smiset <- smisample

## Write objects of classes SMIset to file with and
## without compound identifiers
# write.SMI(smiset[1:4], file="sub.smi", cid=TRUE)
# write.SMI(smiset[1:4], file="sub.smi", cid=FALSE)
```

Index