Package 'sdcTable'

July 23, 2025

Version 0.33.0

Date 2025-06-26

Title Methods for Statistical Disclosure Control in Tabular Data

Description Methods for statistical disclosure control in
tabular data such as primary and secondary cell suppression as described for example
in Hundepol et al. (2012) <doi:10.1002/9781118348239> are covered in this package.

URL https://github.com/sdcTools/sdcTable

BugReports https://github.com/sdcTools/userSupport/issues

Depends R (>= 3.5.0), Rcpp (>= 0.11.0), sdcHierarchies (>= 0.19.1)

Imports data.table, knitr, rlang, stringr, methods, slam, progress,
utils, Matrix (>= 1.3-0), SSBtools, highs

Suggests testthat (>= 0.3), rmarkdown, webshot, digest, RegSDC

LinkingTo Rcpp

License GPL (>= 2)

LazyData true

SystemRequirements GLPK library, including -dev or -devel part

Encoding UTF-8

VignetteBuilder knitr

RoxygenNote 7.3.2

NeedsCompilation yes

Author Bernhard Meindl [aut, cre]

Maintainer Bernhard Meindl <bernhard.meindl@gmail.com>

Repository CRAN

Date/Publication 2025-06-26 08:50:02 UTC

Contents

argusVersion . . . 3
attack . . . 3
calc.sdcProblem . . . 5
cell_info . . . 8
change_cellstatus . . . 9
contributing_indices . . . 11
createArgusInput . . . 12
createJJFormat . . . 15
createRegSDCInput . . . 16
cutList-class . . . 18
dataObj-class . . . 19
dimInfo-class . . . 20
dimVar-class . . . 20
get.dimInfo . . . 21
get.problemInstance . . . 22
get.sdcProblem . . . 24
getInfo . . . 25
linProb-class . . . 27
makeProblem . . . 28
microdata1 . . . 30
microdata2 . . . 31
primarySuppression . . . 31
print,dimVar-method . . . 33
print,sdcProblem-method . . . 34
problemInstance-class . . . 34
protectLinkedTables . . . 35
protectTable . . . 38
runArgusBatchFile . . . 42
safeObj-class . . . 43
sdcProb2df . . . 44
sdcProblem-class . . . 45
sdc_testproblem . . . 46
set.dimInfo . . . 47
set.problemInstance . . . 48
set.sdcProblem . . . 49
setInfo . . . 50
show,sdcProblem-method . . . 51
simpleTriplet-class . . . 52
summary,sdcProblem-method . . . 52
writeJJFormat . . . 53

Index **55**

argusVersion *argusVersion*

Description

returns the version and build number of a given tau-argus executable specified in argument exe.

Usage

```
argusVersion(exe, verbose = FALSE)
```

Arguments

exe	a path to a tau-argus executable
verbose	(logical) if TRUE, the version info and build number of the given tau-argus executable will be printed.

Value

a list with two elements being the tau-argus version and the build-number.

Examples

```
## Not run:
argusVersion(exe="C:\\Tau\\TauArgus.exe", verbose=TRUE)

## End(Not run)
```

attack *Attacking primary suppressed cells*

Description

Function [attack()] is used to compute lower and upper bounds for a given sdcProblem instance. For all calculations the current suppression pattern is used when calculating solutions of the attacker's problem.

Usage

```
attack(object, to_attack = NULL, verbose = FALSE, ...)
```

Arguments

object	an object of class ‘sdcProblem‘
to_attack	if ‘NULL‘ all current primary suppressed cells are attacked; otherwise either an integerish (indices) or character-vector (str-ids) of the cells that should be attacked.
verbose	a logical scalar determing if additional output should be displayed
...	placeholder for possible additional input, currently unused;

Value

a ‘data.frame‘ with the following columns: - ‘prim_supps‘: index of primary suppressed cells - ‘status‘: the original sdc-status code - ‘val‘ the original value of the cell - ‘low‘: computed lower bound of the attacker’s problem - ‘up‘: computed upper bound of the attacker’s problem - ‘protected‘ shows if a given cell is accordingly protected

Author(s)

Bernhard Meindl <bernhard.meindl@statistik.gv.at>

Examples

```
## Not run:
dims <- list(
  v1 = sdcHierarchies::hier_create("tot", letters[1:4]),
  v2 = sdcHierarchies::hier_create("tot", letters[5:8])
)

N <- 150
df <- data.frame(
  v1 = sample(letters[1:4], N, replace = TRUE),
  v2 = sample(letters[5:8], N, replace = TRUE)
)

sdc <- makeProblem(data = df, dimList = dims)

# set primary suppressions
specs <- data.frame(
  v1 = c("a", "b", "a"),
  v2 = c("e", "e", "f")
)
sdc <- change_cellstatus(sdc, specs = specs, rule = "u")

# attack all primary sensitive cells
# the cells can be recomputed exactly
attack(sdc, to_attack = NULL)

# protect the table and attack again
sdc <- protectTable(sdc, method = "SIMPLEHEURISTIC")
attack(sdc, to_attack = NULL)
```

```
# attack only selected cells
attack(sdc, to_attack = c(7, 12))

## End(Not run)
```

`calc.sdcProblem` *perform calculations on* `sdcProblem`*-objects depending on argument* `type`

Description

perform calculations on `sdcProblem`-objects depending on argument `type`

Usage

```
calc.sdcProblem(object, type, input)

## S4 method for signature 'sdcProblem,character,list'
calc.sdcProblem(object, type, input)
```

Arguments

`object`	an object of class `sdcProblem`
`type`	a character vector of length 1 defining what to calculate|return|modify. Allowed types are:

- rule.freq: modify suppression status within `object` according to frequency suppression rule
- heuristicSolution: obtain a heuristic (greedy) solution to the problem defined by `object`
- cutAndBranch: solve a secondary cell suppression problem defined by `object` using cut and branch
- anonWorker: is used to solve the suppression problem depending on information provided with argument `input`
- ghmiter: solve a secondary cell suppression problem defined by `object` using hypercube algorithm
- preprocess: perform a preprocess procedure by trying to identify primary suppressed cells that are already protected due to other primary suppressed cells
- cellID: find index of cell defined by information provided with argument `input`
- finalize: create an object of class `safeObj`
- ghmiter.diagObj: calculate codes required to identify diagonal cells given a valid cell code - used for ghmiter-algorithm only
- ghmiter.calcInformation: calculate information for quaders identified by diagonal indices - used for ghmiter-algorithm only
- ghmiter.suppressQuader: suppress a quader based on indices

- ghmiter.selectQuader: select a quader for suppression depending on information provided with argument `input` - used for ghmiter-algorithm only
- ghmiter.suppressAdditionalQuader: select and suppress an additional quader (if required) based on information provided with argument `input` - used for ghmiter-algorithm only
- contributingIndices: calculate indices within the current problem that contribute to a given cell
- reduceProblem: reduce the problem given by `object` using a vector of indices
- genStructuralCuts: calculate cuts that are absolute necessary for a valid solution of the secondary cell suppression problem

`input` a list depending on argument `type`.

- a list (typically generated using genParaObj()) specifying parameters for primary cell suppression if argument `type` matches 'rule.freq'
- a list if argument `type` matches 'heuristicSolution' having the following elements:
  - element 'aProb': an object of class `linProb` defining the attacker's problem
  - element 'validCuts': an object of class `cutList` representing a list of constraints
  - element 'solver': a character vector of length 1 specifying a solver to use
  - element 'verbose': a logical vector of length 1 setting if verbose output is desired
- a list (typically generated using genParaObj()) specifying parameters for the secondary cell suppression problem if argument `type` matches 'cutAndBranch', 'anonWorker', 'ghmiter', 'preprocess'
- a list of length 3 if argument `type` matches 'cellID' having following elements
  - first element: character vector specifying variable names that need to exist in slot 'dimInfo' of `object`
  - second element: character vector specifying codes for each variable that define a specific table cell
  - third element: logical vector of length 1 with TRUE setting verbosity and FALSE to turn verbose output off
- a list of length 3 if argument `type` matches 'ghmiter.diagObj' having following elements
  - first element: numeric vector of length 1
  - second element: a list with as many elements as dimensional variables have been specified and each element being a character vector of dimension-variable specific codes
  - third element: logical vector of length 1 defining if diagonal indices with frequency == 0 should be allowed or not
- a list of length 4 if argument `type` matches 'ghmiter.calcInformation' having following elements
  - first element: a list object typically generated with method `calc.sdcProblem` and type=='ghmiter.diagObj'
  - second element: a list with as many elements as dimensional variables have been specified and each element being a character vector of dimension-variable specific codes
  - third element: numeric vector of length 1 specifying a desired protection level
  - fourth element: logical vector of length 1 defining if quader containing empty cells should be allowed or not

- a list of length 1 if argument type matches 'ghmiter.suppressQuader' having following element
  - first element: numeric vector of indices that should be suppressed
- a list of length 2 if argument type matches 'ghmiter.selectQuader' having following elements
  - first element: a list object typically generated with method calc.sdcProblem and type=='ghmiter.calcInformation'
  - second element: a list (typically generated using genParaObj())
- a list of length 4 if argument type matches 'ghmiter.suppressAdditionalQuader' having following elements
  - first element: a list object typically generated with method calc.sdcProblem and type=='ghmiter.diagObj'
  - second element: a list object typically generated with method calc.sdcProblem and type=='ghmiter.calcInformation'
  - third element: a list object typically generated with method calc.sdcProblem and type=='ghmiter.selectQuader'
  - fourth element: a list (typically generated using genParaObj())
- a list of length 1 if argument type matches 'contributingIndices' having following element
  - first element: character vector of length 1 being an ID for which contributing indices should be calculated
- a list of length 1 if argument type matches 'reduceProblem' having following element
  - first element: numeric vector defining indices of cells that should be kept in the reduced problem
- an empty list if argument type matches 'genStructuralCuts'

Value

information from objects of class sdcProblem depending on argument type

- an object of class sdcProblem if argument type matches 'rule.freq', 'cutAndBranch', 'anonWorker', 'ghmiter', 'ghmiter.supressQuader', 'ghmiter.suppressAdditionalQuader' or 'reduceProblem'
- a numeric vector with elements being 0 or 1 if argument type matches 'heuristicSolution'
- a list if argument type matches 'preprocess' having following elements:
  - element 'sdcProblem': an object of class sdcProblem
  - element 'aProb': an object of class linProb
  - element 'validCuts': an object of class cutList
- a numeric vector of length 1 specifying the index of the cell of interest if argument type matches 'cellID'
- an object of class safeObj if argument type matches 'finalize'
- a list if argument type matches 'ghmiter.diagObj' having following elements:
  - element 'cellToProtect': character vector of length 1 defining the ID of the cell to protect
  - element 'indToProtect': numeric vector of length 1 defining the index of the cell to protect
  - element 'diagIndices': numeric vector defining indices of possible cells defining cubes
- a list containing information about each quader that could possibly be suppressed if argument type matches 'ghmiter.calcInformation'

- a list containing information about a single quader that should be suppressed if argument `type` matches 'ghmiter.selectQuader'
- a numeric vector with indices that contribute to the desired table cell if argument `type` matches 'contributingIndices'
- an object of class `cutList` if argument `type` matches 'genStructuralCuts'

Note

internal function

Author(s)

Bernhard Meindl <bernhard.meindl@statistik.gv.at>

---

cell_info *Get information about specific cells*

Description

Function `cellInfo()` can be used to query information of a single cell from a sdcProblem object. If the instance has already been protected using `protectTable()`, the information is retrieved from the final protected dataset, otherwise from the current state of the instance.

Usage

```
cell_info(object, specs, ...)
```

Arguments

`object`	an object of class sdcProblem
`specs`	input that defines which cells to query; the function expects either (see examples below) • a named character vector: with names referring to the names of the dimensional variables and the values to its labels. In this case each vector-element must contain a single value (label) • a `data.frame` where the column-names refer to the names of the dimensional variables and the values to the labels
`...`	additional parameters for potential future use, currently unused.

Value

a `data.frame` with a row for each of the queried cells; the object contains the following columns:

- id: numeric vector of length 1 specifying the numerical index of the cell
- a column `strID` if `object` has not yet been protected
- one column for each dimensional variable

- a column `freq` containing the cell-frequencies
- if available, one column for each (possible) numerical value that was tabulated
- a column `sdcStatus` with the current sdc code
- is_primsupp: is `TRUE` if the cell is a primary sensitive cell
- is_secondsupp: is `TRUE` if the cell is a secondary suppressed cell

Author(s)

Bernhard Meindl <bernhard.meindl@statistik.gv.at>

Examples

```
# as in makeProblem() with a single primary suppression
p <- sdc_testproblem(with_supps = TRUE)
sdcProb2df(p)

# vector input
specs_vec <- c(region = "D", gender = "male")
cell_info(p, specs = specs_vec)

# data.frame input
specs_df <- data.frame(
  region = c("A", "D", "A"),
  gender = c("male", "female", "female")
)
cell_info(p, specs = specs_df)

# protect the table
p_safe <- protectTable(p, method = "SIMPLEHEURISTIC")

# re-apply
cell_info(p_safe, specs = specs_df)
```

---

change_cellstatus *Change anonymization status of a specific cell*

Description

Function `change_cellstatus()` allows to change|modify the anonymization state of single table cells for objects of class sdcProblem.

Usage

```
change_cellstatus(object, specs, rule, verbose = FALSE, ...)
```

Arguments

object	an object of class sdcProblem
specs	input that defines which cells to query; the function expects either (see examples below) • a named character vector: with names referring to the names of the dimensional variables and the values to its labels. In this case each vector-element must contain a single value (label) • a data.frame where the column-names refer to the names of the dimensional variables and the values to the labels
rule	scalar character vector specifying a valid anonymization code ('u', 'z', 'x', 's') to which all the desired cells under consideration should be set.
verbose	scalar logical value defining verbosity, defaults to FALSE
...	additional parameters for potential future use, currently unused.

Value

a sdcProblem object

Author(s)

Bernhard Meindl <bernhard.meindl@statistik.gv.at>

Examples

```
# load example-problem
# (same as example from ?makeProblem)
p <- sdc_testproblem(with_supps = FALSE)

# goal: set cells with region = "D" and gender != "total" as primary sensitive

# using a data.frame as input
specs <- data.frame(
  region = "D",
  gender = c("male", "female", "total")
)

# marking the cells as sensitive
p <- change_cellstatus(
  object = p,
  specs = specs,
  rule = "u"
)

# check
cell_info(p, specs = specs)

# using a named vector for a single cell to revert
# setting D/total as primary-sensitive
```

```
specs <- c(gender = "total", region = "D")

p <- change_cellstatus(
  object = p,
  specs = specs,
  rule = "s"
)

# and check again
cell_info(p, specs = specs)
```

`contributing_indices` *Compute contributing units to table cells*

Description

This function computes (with respect to the raw input data) the indices of all contributing units to given cells identified by `ids`.

Usage

```
contributing_indices(prob, ids = NULL)
```

Arguments

`prob`	a sdcProblem object created with `makeProblem()`
`ids`	a character vector containing default ids (strIDs) that define table cells. Valid inputs can be extracted by using `sdcProb2df()` and looking at column `strID`. If this argument is `NULL`, the corresponding units are computed for all cells in the table.

Value

a named `list where names correspond to the given ids`‘ and the values to the row numbers within the raw input data.

Examples

```
# loading test problem
p <- sdc_testproblem(with_supps = FALSE)
dt <- sdcProb2df(p, dimCodes = "original")

# question: which units contribute to cell region = "A" and gender = "female"?

# compute the id ("0102")
dt[region == "A" & gender == "female", strID]

# which indices contribute to the cell?
ids <- contributing_indices(prob = p, ids = "0101")
```

```
# check
dataObj <- get.sdcProblem(p, "dataObj")
rawData <- slot(dataObj, "rawData")
rawData[ids[["0101"]]]

# compute contributing ids for all cells
contributing_indices(p)
```

---

createArgusInput *Create input files for tauArgus*

Description

create required input-files and batch-file for tau-argus given an sdcProblem object

Usage

```
createArgusInput(
  obj,
  typ = "microdata",
  verbose = FALSE,
  path = getwd(),
  solver = "FREE",
  method,
  primSuppRules = NULL,
  responsevar = NULL,
  shadowvar = NULL,
  costvar = NULL,
  requestvar = NULL,
  holdingvar = NULL,
  ...
)
```

Arguments

obj	an object of class sdcProblem from sdcTable
typ	(character) either "microdata" or "tabular"
verbose	(logical) if TRUE, the contents of the batch-file are written to the prompt
path	path, into which (temporary) files will be written to (amongst them being the batch-files). Each file written to this folder belonging to the same problem contains a random id in its filename.
solver	which solver should be used. allowed choices are • "FREE" • "CPLEX" • "XPRESS"

In case "CPLEX" is used, it is also mandatory to specify argument licensefile which needs to be the absolute path the the cplex license file

method — secondary cell suppression algorithm, possible choices include:

- "MOD": modular approach. If specified, the following arguments in ... can additionally be set:
  - MaxTimePerSubtable: number specifiying max. time (in minutes) spent for each subtable
  - SingleSingle: 0/1 (default=1)
  - SingleMultiple: 0/1 (default=1)
  - MinFreq: 0/1 (default=1)
- "GH": hypercube. If specified, the following arguments in ... can additionally be set:
  - BoundPercentage: Default percentage to proctect primary suppressed cells, default 75
  - ModelSize: are we dealing with a small (0) or large (1) model? (default=1)
  - ApplySingleton: should singletons be additionally protected? 0/1 (default=1)
- "OPT": optimal cell suppression. If specified, the following arguments in ... can additionally be set:
  - MaxComputingTime: number specifiying max. allowed computing time (in minutes)

primSuppRules — rules for primary suppression, provided as a list. For details, please have a look at the examples below.

responsevar — which variable should be tabulated (defaults to frequencies). For details see tau-argus manual section 4.4.4.

shadowvar — if specified, this variable is used to apply the safety rules, defaults to responsevar. For details see tau-argus manual section 4.4.4.

costvar — if specified, this variable describes the costs of suppressing each individual cell. For details see tau-argus manual section 4.4.4.

requestvar — if specified, this variable (0/1-coded) contains information about records that request protection. Records with 1 will be protected in case a corresponding request rule matches. It is ignored, if tabular input is used.

holdingvar — if specified, this variable contains information about records that should be grouped together. It is ignored, if tabular input is used.

... — allows to specify additional parameters for selected suppression-method as described above as well as licensefile in clase "CPLEX" was specified in argument solver.

Value

the filepath to the batch-file

Examples

```
## Not run:
# loading micro data from sdcTable
utils::data("microdata1", package="sdcTable")
microdata1$num1 <- rnorm(mean = 100, sd = 25, nrow(microdata1))
microdata1$num2 <- round(rnorm(mean = 500, sd=125, nrow(microdata1)),2)
microdata1$weight <- sample(10:100, nrow(microdata1), replace = TRUE)

dim_region <- hier_create(root = "Total", nodes = LETTERS[1:4])

dim_region_dupl <- hier_create(root = "Total", nodes = LETTERS[1:4])
dim_region_dupl <- hier_add(dim_region_dupl, root = "B", nodes = c("b1"))
dim_region_dupl <- hier_add(dim_region_dupl, root = "D", nodes = c("d1"))

dim_gender <- hier_create(root = "Total", nodes = c("male", "female"))

dimList <- list(region = dim_region, gender = dim_gender)
dimList_dupl  <- list(region = dim_region_dupl, gender = dim_gender)
dimVarInd <- 1:2
numVarInd <- 3:5
sampWeightInd <- 6

# creating an object of class \code{\link{sdcProblem-class}}
obj <- makeProblem(
  data = microdata1,
  dimList = dimList,
  dimVarInd = dimVarInd,
  numVarInd = numVarInd,
  sampWeightInd = sampWeightInd)

# creating an object of class \code{\link{sdcProblem-class}} containing "duplicated" codes
obj_dupl <- makeProblem(
  data = microdata1,
  dimList = dimList_dupl,
  dimVarInd = dimVarInd,
  numVarInd = numVarInd,
  sampWeightInd = sampWeightInd)

## create primary suppression rules
primSuppRules <- list()
primSuppRules[[1]] <- list(type = "freq", n = 5, rg = 20)
primSuppRules[[2]] <- list(type = "p", n = 5, p = 20)
# other supported formats are:
# list(type = "nk", n=5, k=20)
# list(type = "zero", rg = 5)
# list(type = "mis", val = 1)
# list(type = "wgt", val = 1)
# list(type = "man", val = 20)

## create batchInput object
bO_md1 <- createArgusInput(
  obj = obj,
```

```
  typ = "microdata",
  path = tempdir(),
  solver = "FREE",
  method = "OPT",
  primSuppRules = primSuppRules,
  responsevar = "num1")

bO_td1 <- createArgusInput(
  obj = obj,
  typ = "tabular",
  path = tempdir(),
  solver = "FREE",
  method = "OPT")

bO_td2 <- createArgusInput(
  obj = obj_dupl,
  typ = "tabular",
  path = tempdir(),
  solver = "FREE",
  method = "OPT")

## in case CPLEX should be used, it is required to specify argument licensefile
bO_md2 <- createArgusInput(
  obj = obj,
  typ = "microdata",
  path = tempdir(),
  solver = "CPLEX",
  method = "OPT",
  primSuppRules = primSuppRules,
  responsevar = "num1",
  licensefile = "/path/to/my/cplexlicense")

## End(Not run)
```

`createJJFormat` *Create input for jj_format*

Description

This function transforms a sdcProblem object into a list that can be used as input for `writeJJFormat()` to write a problem in `"JJ-format"` to disk.

Usage

```
createJJFormat(x)
```

Arguments

x	a sdcProblem object

Value

an input suitable for `writeJJFormat()`

Author(s)

Bernhard Meindl (bernhard.meindl@statistik.gv.at) and Sapphire Yu Han (y.han@cbs.nl)

Examples

```
# setup example problem
# microdata
utils::data("microdata1", package = "sdcTable")

# create hierarchies
dims <- list(
  region = sdcHierarchies::hier_create(root = "Total", nodes = LETTERS[1:4]),
  gender = sdcHierarchies::hier_create(root = "Total", nodes = c("male", "female")))

# create a problem instance
p <- makeProblem(
  data = microdata1,
  dimList = dims,
  numVarInd = "val")

# create suitable input for `writeJJFormat`
inp <- createJJFormat(p); inp

# write files to disk
# frequency table by default
writeJJFormat(
  x = inp,
  path = file.path(tempdir(), "prob_freqs.jj"),
  overwrite = TRUE
)

# or using the numeric variable `val` previously specified
writeJJFormat(
  x = inp,
  tabvar = "val",
  path = file.path(tempdir(), "prob_val.jj"),
  overwrite = TRUE
)
```

---

createRegSDCInput *Create input for RegSDC/other Tools*

Description

This function transforms a sdcProblem object into an object that can be used as input for RegSDC::SuppressDec (among others).

Usage

```
createRegSDCInput(x, chk = FALSE)
```

Arguments

x	a sdcProblem object
chk	a logical value deciding if computed linear relations should be additionally checked for validity

Value

an `list` with the following elements:

- `mat`: linear combinations depending on inner-cells of the given problem instance.
- `y`: a 1-column matrix containing the frequencies of inner cells
- `z`: a 1-column matrix containing the frequencies of all cells
- `z_supp`: a 1-column matrix containing the frequencies of all cells but suppressed cells have a value of `NA`
- `info`: a `data.frame` with the following columns:
  - `cell_id`: internal cell-id used in sdcTable
  - `is_innercell`: a binary indicator if the cell is an internal cell (`TRUE`) or a (sub)total (`FALSE`)

Author(s)

Bernhard Meindl (bernhard.meindl@gmail.com)

Examples

```
## Not run:
utils::data("microdata1", package = "sdcTable")
head(microdata1)

# define the problem
dim_region <- hier_create(root = "total", nodes = sort(unique(microdata1$region)))
dim_gender <- hier_create(root = "total", nodes = sort(unique(microdata1$gender)))

prob <- makeProblem(
  data = microdata1,
  dimList = list(region = dim_region, gender = dim_gender),
  freqVarInd = NULL
)

# suppress some cells
prob <- primarySuppression(prob, type = "freq", maxN = 15)

# compute input for RegSDC-package
inp_regsdc <- createRegSDCInput(x = prob, chk = TRUE)
```

```
# estimate innner cells based on linear dependencies
res_regsdc <- RegSDC::SuppressDec(
  x = as.matrix(inp_regsdc$x),
  z = inp_regsdc$z_supp,
  y = inp_regsdc$y)[, 1]

# check if inner cells are all protected
df <- data.frame(
  freqs_orig = inp_regsdc$z[inp_regsdc$info$is_innercell == TRUE, ],
  freqs_supp = inp_regsdc$z_supp[inp_regsdc$info$is_innercell == TRUE, ],
  regsdc = res_regsdc
)

subset(df, df$regsdc == df$freqs_orig & is.na(freqs_supp))


## End(Not run)
```

`cutList-class` *S4 class describing a cutList-object*

Description

An object of class `cutList` holds constraints that can be extracted and used as for objects of class `linProb-class`. An object of class `cutList` consists of a constraint matrix (slot `con`), a vector of directions (slot `direction`) and a vector specifying the right hand sides of the constraints (slot `rhs`).

Details

slot `con`**:** an object of class `simpleTriplet-class` specifying the constraint matrix of the problem

slot `direction`**:** a character vector holding the directions of the constraints, allowed values are:

- ==: equal
- <: less
- >: greater
- <=: less or equal
- >=: greater or equal

slot `rhs`**:** numeric vector holding right hand side values of the constraints

Note

objects of class `cutList` are dynamically generated (and removed) during the cut and branch algorithm when solving the secondary cell suppression problem

Author(s)

Bernhard Meindl <bernhard.meindl@statistik.gv.at>

dataObj-class *S4 class describing a dataObj-object*

Description

This class models a data object containing the 'raw' data for a given problem as well as information on the position of the dimensional variables, the count variable, additional numerical variables, weights or sampling weights within the raw data. Also slot 'isMicroData' shows if slow 'rawData' consists of microdata (multiple observations for each cell are possible, isMicroData==TRUE) or if data have already been aggregated (isMicroData==FALSE)

Details

slot `rawData`**:** list with each element being a vector of either codes of dimensional variables, counts, weights that should be used for secondary cell suppression problem, numerical variables or sampling weights.

slot `dimVarInd`**:** numeric vector (or NULL) defining the indices of the dimensional variables within slot 'rawData'

slot `freqVarInd`**:** numeric vector (or NULL) defining the indices of the frequency variables within slot 'rawData'

slot `numVarInd`**:** numeric vector (or NULL) defining the indices of the numerical variables within slot 'rawData'

slot `weightVarInd`**:** numeric vector (or NULL) defining the indices of the variables holding weights within slot 'rawData'

slot `sampWeightInd`**:** numeric vector (or NULL) defining the indices of the variables holding sampling weights within slot 'rawData'

slot `isMicroData`**:** logical vector of length 1 (or NULL) that is TRUE if slot 'rawData' are microData and FALSE otherwise

Note

objects of class `dataObj` are input for slot `dataObj` in class `sdcProblem`

Author(s)

Bernhard Meindl <bernhard.meindl@statistik.gv.at>

dimInfo-class *S4 class describing a dimInfo-object*

Description

An object of class `dimInfo` holds all necessary information about the dimensional variables defining a hierarchical table that needs to be protected.

Details

slot `dimInfo`**:** a list (or NULL) with all list elements being objects of class `dimVar`

slot `strID`**:** a character vector (or NULL) defining IDs that identify each table cell. The ID's are based on (default) codes of the dimensional variables defining a cell.

slot `strInfo`**:** a list object (or NULL) with each list element being a numeric vector of length 2 defining the start and end-digit that is allocated by the i-th dimensional variable in ID-codes available in slot `strID`

slot `vNames`**:** a character vector (or NULL) defining the variable names of the dimensional variables defining the table structure

slot `posIndex`**:** a numeric vector (or NULL) holding the position of the dimensional variables within slot `rawData` of class `dataObj`

Note

objects of class `dimInfo` are input for slots in classes `sdcProblem` and `safeObj`

Author(s)

Bernhard Meindl <bernhard.meindl@statistik.gv.at>

dimVar-class *S4 class describing a dimVar-object*

Description

An object of class `dimVar` holds all necessary information about a single dimensional variable such as original and standardized codes, the level-structure, the hierarchical structure, codes that may be (temporarily) removed from building the complete hierarchy (dups) and their corresponding codes that correspond to these duplicated codes.

Details

slot `codesOriginal`**:** a character vector (or NULL) holding original variable codes

slot `codesDefault`**:** a character vector (or NULL) holding standardized codes

slot `codesMinimal`**:** a logical vector (or NULL) defining if a code is required to build the complete hierarchy or not (then the code is a (sub)total)

slot `vName`**:** character vector of length 1 (or NULL) defining the variable name of the dimensional variable

slot `levels`**:** a numeric vector (or NULL) defining the level structure. For each code the corresponding level is listed with the grand-total always having level==1

slot `structure`**:** a numeric vector (or NULL) with length of the total number of levels. Each element shows how many digits the i-th level allocates within the standardized codes (note: level 1 always allocates exactly 1 digit in the standardized codes)

slot `dims`**:** a list (or NULL) defining the hierarchical structure of the dimensional variable. Each list-element is a character vector with elements available in slot `codesDefault` and the first element always being a (sub)total and the remaining elements being the codes that contribute to the (sub)total

slot `dups`**:** character vector (or NULL) having showing original codes that are duplicates in the hierarchy and can temporarily removed when building a table with this dimensional variable

slot `dupsUp`**:** character vector (or NULL) with original codes that are the corresponding upper-levels to the codes that may be removed because they are duplicates and that are listed in slot `dups`

Note

objects of class `dimVar` form the base for elements in slot `dimInfo` of class `dimInfo`.

Author(s)

Bernhard Meindl <bernhard.meindl@statistik.gv.at>

`get.dimInfo` *query* `dimInfo`*-objects depending on argument* `type`

Description

query `dimInfo`-objects depending on argument `type`

Usage

```
get.dimInfo(object, type)

## S4 method for signature 'dimInfo,character'
get.dimInfo(object, type)
```

Arguments

object	an object of class dataObj
type	a character vector of length 1 defining what to calculate|return|modify. Allowed types are:

- strInfo: info on how many digits in the default codes ach dimensional variable allocates
- dimInfo: a list object with each slot containing an object of class dimVar
- varName: variable names
- strID: character vector of ID's defining table cells
- posIndex vector showing the index of the elements of dimInfo in the underlying data

Value

information from objects of class dimInfo depending on argument type

- a list (or NULL) if argument type matches 'strInfo', 'dimInfo'
- numeric vector (or NULL) if argument type matches 'posIndex'
- character vector (or NULL) if argument type matches 'varName' or 'strID'

Note

internal function

Author(s)

Bernhard Meindl <bernhard.meindl@statistik.gv.at>

---

get.problemInstance *query* problemInstance-*objects depending on argument* type

---

Description

query problemInstance-objects depending on argument type

Usage

```
get.problemInstance(object, type)

## S4 method for signature 'problemInstance,character'
get.problemInstance(object, type)
```

Arguments

object	an object of class problemInstance
type	a character vector of length 1 defining what to calculate|return|modify. Allowed types are:

- strID: vector of unique IDs for each table cell
- nrVars: total number of table cells
- freq: vector of frequencies
- w: a vector of weights used in the linear problem (or NULL)
- numVars: a list containing numeric vectors containing values for numerical variables for each table cell (or NULL)
- sdcStatus: a vector containing the suppression state for each cell (possible values are 'u': primary suppression, 'x': secondary suppression, 'z': forced for publication, 's': publishable cell, 'w': dummy cells that are considered only when applying the simple greedy heuristic to protect the table)
- lb: lower bound assumed to be known by attackers for each table cell
- ub: upper bound assumed to be known by attackers for each table cell
- LPL: lower protection level required to protect table cells
- UPL: upper protection level required to protect table cells
- SPL: sliding protection level required to protect table cells
- primSupps: vector of indices of primary sensitive cells
- secondSupps: vector of indices of secondary suppressed cells
- forcedCells: vector of indices of cells that must not be suppressed
- hasPrimSupps: shows if object has primary suppressions or not
- hasSecondSupps: shows if object has secondary suppressions or not
- hasForcedCells: shows if object has cells that must not be suppressed
- weight: gives weight that is used the suppression procedures
- suppPattern: gives the current suppression pattern

Value

information from objects of class dataObj depending on argument type

- a list (or NULL) if argument type matches 'numVars'
- numeric vector if argument type matches 'freq', 'lb', 'ub', 'LPL', 'UPL', 'SPL', 'weight', 'suppPattern'
- numeric vector (or NULL) if argument type matches 'w', 'primSupps', 'secondSupps', 'forcedCells'
- character vector if argument type matches 'strID', 'sdcStatus', ''
- logical vector of length 1 if argument type matches 'hasPrimSupps', 'hasSecondSupps', 'hasForcedCells'
- numerical vector of length 1 if argument type matches 'nrVars'

Note

internal function

Author(s)

Bernhard Meindl <bernhard.meindl@statistik.gv.at>

`get.sdcProblem` *query* `sdcProblem`*-objects depending on argument* `type`

Description

query `sdcProblem`-objects depending on argument `type`

Usage

```
get.sdcProblem(object, type)

## S4 method for signature 'sdcProblem,character'
get.sdcProblem(object, type)
```

Arguments

`object`	an object of class `sdcProblem`
`type`	a character vector of length 1 defining what to calculate|return|modify. Allowed types are:

- dataObj: a list containing the (raw) input data
- problemInstance: return the current problem instance
- partition: a list containing information on the subtables that are required to be protected as well as information on the processing order of the subtables
- dimInfo: information on the variables defining the hierarchical table
- indicesDealtWith: a set of indices that have already been dealt with during the protection algorithmus
- startI: current level at which subtables need to be protected (useful when restarting HITAS|HYPERCUBE)
- startJ: current number of the subtable within a given level that needs to be protected (useful when restarting HITAS|HYPERCUBE)
- innerAndMarginalCellInfo: for a given problem, get indices of inner- and marginal table cells

Value

information from objects of class sdcProblem depending on argument type

- an object of class dataObj (or NULL) if type matches 'dataObj'
- an object of class problemInstance (or NULL) if type matches 'problemInstance'
- a list (or NULL) if argument type matches 'partition' containing the following elements:
  - element 'groups': list with each list-element being a character vector specifying a specific level-group
  - element 'indices': list with each list-element being a numeric vector defining indices of a subtable
  - element 'strIDs': list with each list-element being a character vector defining IDs of a subtable
  - element 'nrGroups': numeric vector of length 1 defining the total number of groups that have to be considered
  - element 'nrTables': numeric vector of length 1 defining the total number of subtables that have to be considered
- a list (or NULL) if argument type matches 'innerAndMarginalCellInfo' containing the following elements:
  - element 'innerCells': character vector specifying ID's of inner cells
  - element 'totCells': character vector specifying ID's of marginal cells
  - element 'indexInnerCells': numeric vector specifying indices of inner cells
  - element 'indexTotCells': numeric vector specifying indices of marginal cells
- an object of class dimInfo (or NULL) if type matches 'dimInfo'
- numeric vector of length 1 if argument type matches 'startI' or 'startJ'

Note

internal function

Author(s)

Bernhard Meindl <bernhard.meindl@statistik.gv.at>

---

getInfo *Retrieve information in* sdcProblem *or* problemInstance *objects*

Description

Function getInfo() is used to extract values from sdcProblem or problemInstance objects

Usage

```
getInfo(object, type)
```

Arguments

object	an object of class sdcProblem or problemInstance
type	a scalar character specifying the information which should be returned. If object inherits class problemInstance, the slots are directly accessed, otherwise the values within slot problemInstance of the sdcProblem object are queried. Valid choices are:

- the object has not yet been protected
  - lb and ub: current possible lower and upper bounds
  - LPL, SPL, UPL: current lower, sliding and upper protection levels
  - sdcStatus: current sdc-status of cells
  - freq: cell frequencies
  - strID: standardized cell ids (chr)
  - numVars: NULL or a list with a slot for each tabulated numerical variable;
  - w: sampling weights or NULL
- the table has already been protected
  - finalData: protected table as a data.table
  - nrNonDuplicatedCells: number of unique (non-bogus) cells in the table
  - nrPrimSupps: number of primary sensitive cells that were protected
  - nrSecondSupps: number of additional secondary suppressions
  - nrPublishableCells: number of cells (status "s or ‘"z") that may be published
  - suppMethod: name of the algorithm used to protect the table

Value

manipulated data depending on arguments object and type

Author(s)

Bernhard Meindl <bernhard.meindl@statistik.gv.at>

Examples

```
# define an example problem with two hierarchies
p <- sdc_testproblem(with_supps = FALSE)

# apply primary suppression
p <- primarySuppression(p, type = "freq", maxN = 3)

# `p` is an `sdcProblem` object
print(class(p))

for (slot in c("lb", "ub", "LPL", "SPL", "UPL", "sdcStatus",
  "freq", "strID", "numVars", "w")) {
  message("slot: ", shQuote(slot))
```

```
  print(getInfo(p, type = slot))
}

# protect the cell and extract results
p_protected <- protectTable(p, method = "SIMPLEHEURISTIC")
for (slot in c("finalData", "nrNonDuplicatedCells", "nrPrimSupps",
  "nrSecondSupps", "nrPublishableCells", "suppMethod")) {
  message("slot: ", shQuote(slot))
  print(getInfo(p_protected, type = slot))
}
```

`linProb-class` *S4 class describing a linProb-object*

Description

An object of class `linProb` defines a linear problem given by the objective coefficients (slot `objective`), a constraint matrix (slot `constraints`), the direction (slot `direction`) and the right hand side (slot `rhs`) of the constraints. Also, allowed lower (slot `boundsLower`) and upper (slot `boundsUpper`) bounds of the variables as well as its types (slot `types`) are specified.

Details

slot `objective`**:** a numeric vector holding coefficients of the objective function

slot `constraints`**:** an object of class `simpleTriplet-class` specifying the constraint matrix of the problem

slot `direction`**:** a character vector holding the directions of the constraints, allowed values are:

- ==: equal
- <: less
- >: greater
- <=: less or equal
- >=: greater or equal

slot `rhs`**:** numeric vector holding right hand side values of the constraints

slot `boundsLower`**:** a numeric vector holding lower bounds of the objective variables

slot `boundsUpper`**:** a numeric vector holding upper bounds of the objective variables

slot `types`**:** a character vector specifying types of the objective variables, allowed types are:

- `C`: binary
- `B`: continuous
- `I`: integer

Note

when solving the problems in the procedure, minimization of the objective is performed.

Author(s)

Bernhard Meindl <bernhard.meindl@statistik.gv.at>

`makeProblem` *Create a problem instance*

Description

Function `makeProblem()` is used to create sdcProblem objects.

Usage

```
makeProblem(
  data,
  dimList,
  dimVarInd = NULL,
  freqVarInd = NULL,
  numVarInd = NULL,
  weightInd = NULL,
  sampWeightInd = NULL
)
```

Arguments

`data`	a data frame featuring at least one column for each desired dimensional variable. Optionally the input data can feature variables that contain information on cell counts, weights that should be used during the cut and branch algorithm, additional numeric variables or variables that hold information on sampling weights.
`dimList`	a (named) list where the names refer to variable names in input `data`. If the list is not named, it is required to specify argument `dimVarInd`. Each list element can be one of: • `tree`: generated with `hier_*()` functions from package `sdcHierarchies` • `data.frame`: a two column `data.frame` containing the full hierarchy of a dimensional variable using a top-to-bottom approach. The format of this `data.frame` is as follows: – **first column:** a character vector specifying levels with each vector element being a string only containing of `@`s from length 1 to n. If a vector element consists of `i`-chars, the corresponding code is of level `i`. The code `@` (one character) equals the grand total (level=1), the code `@@` (two characters) is of level 2 (directly below the overall total). – **second column:** a character vector specifying level codes • `path`: absolute or relative path to a `.csv` file that contains two columns seperated by semicolons (`;`) having the same structure as the `"@;levelname"`-structure described above
`dimVarInd`	if `dimList` is a named list, this argument is ignored (`NULL`). Else either a numeric or character vector defining the column indices or names of dimensional variables (specifying the table) within argument `data` are expected.
`freqVarInd`	if not `NULL`, a scalar numeric or character vector defining the column index or variable name of a variable holding counts in `data`

numVarInd	if not NULL, a numeric or character vector defining the column indices or variable names of additional numeric variables with respect to data
weightInd	if not NULL, a scalar numeric or character vector defining the column index or variable name holding costs within data that should be used as objective coefficients when solving secondary cell suppression problems.
sampWeightInd	if not NULL, a scalar numeric or character vector defining the column index or variable name of a variable holding sampling weights within data. In case a complete table is provided, this parameter is ignored.

Value

a sdcProblem object

Author(s)

Bernhard Meindl

Examples

```
# loading micro data
utils::data("microdata1", package = "sdcTable")

# we can observe that we have a micro data set consisting
# of two spanning variables ('region' and 'gender') and one
# numeric variable ('val')

# specify structure of hierarchical variable 'region'
# levels 'A' to 'D' sum up to a Total
dim.region <- data.frame(
 levels=c('@','@@','@@','@@','@@'),
 codes=c('Total', 'A','B','C','D'),
 stringsAsFactors=FALSE)

# specify structure of hierarchical variable 'gender'
# using create_node() and add_nodes() (see ?manage_hierarchies)
dim.gender <- hier_create(root = "Total", nodes = c("male", "female"))
hier_display(dim.gender)

# create a named list with each element being a data-frame
# containing information on one dimensional variable and
# the names referring to variables in the input data
dimList <- list(region = dim.region, gender = dim.gender)

# third column containts a numeric variable
numVarInd <- 3

# no variables holding counts, numeric values, weights or sampling
# weights are available in the input data
# creating an problem instance using numeric indices
p1 <- makeProblem(
  data = microdata1,
```

```
  dimList = dimList,
  numVarInd = 3 # third variable in `data`
)

# using variable names is also possible
p2 <- makeProblem(
  data = microdata1,
  dimList = dimList,
  numVarInd = "val"
)

# what do we have?
print(class(p1))

# have a look at the data
df1 <- sdcProb2df(p1, addDups = TRUE,
  addNumVars = TRUE, dimCodes = "original")
df2 <- sdcProb2df(p2, addDups=TRUE,
  addNumVars = TRUE, dimCodes = "original")
print(df1)

identical(df1, df2)
```

`microdata1` *Synthetic Microdata (1)*

Description

A ‘data.frame‘ used for examples and problem-generation in various examples.

Usage

```
data(microdata1)
```

Format

a ‘data.frame‘ with ‘100‘ rows and variables ‘region‘, ‘gender‘ and ‘val‘.

Examples

```
utils::data("microdata1", package = "sdcTable")
head(microdata1)
```

microdata2 *Synthetic Microdata (2)*

Description

Example microdata used for example in [protect_linked_tables()].

Usage

```
data(microdata2)
```

Format

a ‘data.frame‘ with ‘100‘ observations containing variables ‘region‘, ‘gender‘, ‘ecoOld‘, ‘ecoNew‘ and ‘numVal‘.

Examples

```
utils::data("microdata2", package = "sdcTable")
head(microdata2)
```

primarySuppression *Apply primary suppression*

Description

Function `primarySuppression()` is used to identify and suppress primary sensitive table cells in sdcProblem objects. Argument `type` allows to select a rule that should be used to identify primary sensitive cells. At the moment it is possible to identify and suppress sensitive table cells using the frequency-rule, the nk-dominance rule and the p-percent rule.

Usage

```
primarySuppression(object, type, ...)
```

Arguments

`object`	a sdcProblem object
`type`	character vector of length 1 defining the primary suppression rule. Allowed types are: • `freq`: apply frequency rule with parameters `maxN` and `allowZeros` • `nk`: apply nk-dominance rule with parameters `n`, `k` • `p`: apply p-percent rule with parameter `p` • `pq`: apply pq-rule with parameters `p` and `q`

... parameters used in the identification of primary sensitive cells. Parameters that can be modified|changed are:

- maxN: numeric vector of length 1 used when applying the frequency rule. All cells having counts <= maxN are set as primary suppressed. The default value of maxN is 3.
- allowZeros: logical value defining if empty cells (with frequency = 0) should be considered sensitive when using the frequency rule. Empty cells are never considered as sensitive when applying dominance rules; The default value of allowZeros is FALSE so that empty cells are not considered primary sensitive by default. Such cells (frequency 0) are then flagged as z which indicates such a cell may be published but should (internally) not be used for (secondary) suppression in the heuristic algorithms.
- p: numeric vector of length 1 specifying parameter p that is used when applying the p-percent rule with default value of 80.
- pq: numeric vector of length 2 specifying parameters p and q that are used when applying the pq-rule with the default being c(25, 50).
- n: numeric vector of length 1 specifying parameter n that is used when applying the nk-dominance rule. Parameter n is set to 2 by default.
- k: scalar numeric specifying parameter k that is used when applying the nk-dominance rule. Parameter n is set to 85 by default.
- numVarName: character scalar specifying the name of the numerical variable that should be used to identify cells that are dominated by dominance rules (p-rule, pq-rule or nk-rule). This setting is mandatory in package versions >= 0.29 If type is either 'nk', 'p' or 'pq', it is mandatory to specify either numVarInd or numVarName.
- numVarInd: same as numVarName but a scalar numeric specifying the index of the variable is expected. If both numVarName and numVarInd are specified, numVarName is used. The index refers to the index of the specified numvars in makeProblem(). This argument is no longer respected in versions >= 0.29 where numVarName must be used.

Details

since versions >= 0.29 it is no longer possible to specify underlying variables for dominance rules ("p", "pq" or "nk") by index; these variables must be set by name using argument numVarName.

Value

a sdcProblem object

Note

the nk-dominance rule, the p-percent rule and the pq-rule can only be applied if micro data have been used as input data to function makeProblem()

Author(s)

Bernhard Meindl <bernhard.meindl@statistik.gv.at>

Examples

```
# load micro data
utils::data("microdata1", package = "sdcTable")

# load problem (as it was created in the example in ?makeProblem
p <- sdc_testproblem(with_supps = FALSE)

# we have a look at the frequency table by gender and region
xtabs(rep(1, nrow(microdata1)) ~ gender + region, data = microdata1)

# 2 units contribute to cell with region=='A' and gender=='female'
# --> this cell is considered sensitive according the the
# freq-rule with 'maxN' equal to 2!
p1 <- primarySuppression(
  object = p,
  type = "freq",
  maxN = 2
)

# we can also apply a p-percent rule with parameter "p" being 30 as below.
# This is only possible if we are dealing with micro data and we also
# have to specify the name of a numeric variable.
p2 <- primarySuppression(
  object = p,
  type = "p",
  p = 30,
  numVarName = "val"
)

# looking at anonymization states we see, that one cell is primary
# suppressed (sdcStatus == "u")
# the remaining cells are possible candidates for secondary cell
# suppression (sdcStatus == "s") given the frequency rule with
# parameter "maxN = 2".
#
# Applying the p-percent rule with parameter 'p = 30' resulted in
# two primary suppressions.
data.frame(
  p1_sdc = getInfo(p1, type = "sdcStatus"),
  p2_sdc = getInfo(p2, type = "sdcStatus")
)
```

`print,dimVar-method` *print* `dimVar-class` *objects*

Description

print `dimVar-class` objects in a resonable way

Usage

```
## S4 method for signature 'dimVar'
print(x, ...)
```

Arguments

x	An object of class dimVar-class
...	currently not used

---

print,sdcProblem-method *print objects of class* sdcProblem-class.

Description

print some useful information instead of just displaying the entire object (which may be large)

Usage

```
## S4 method for signature 'sdcProblem'
print(x, ...)
```

Arguments

x	an objects of class sdcProblem-class
...	currently not used.

---

problemInstance-class *S4 class describing a problemInstance-object*

Description

An object of class problemInstance holds the main information that is required to solve the secondary cell suppression problem.

Details

slot strID**:** a character vector (or NULL) of ID's identifying table cells

slot Freq**:** a numeric vector (or NULL) of counts for each table cell

slot w**:** a numeric vector (or NULL) of weights that should be used when solving the secondary cell suppression problem

slot numVars**:** a list (or NULL) with each element being a numeric vector holding values of specified numerical variables for each table cell

slot `lb`**:** numeric vector (or NULL) holding assumed lower bounds for each table cell

slot `ub`**:** numeric vector (or NULL) holding assumed upper bounds for each table cell

slot `LPL`**:** numeric vector (or NULL) holding required lower protection levels for each table cell

slot `UPL`**:** numeric vector (or NULL) holding required upper protection levels for each table cell

slot `SPL`**:** numeric vector (or NULL) holding required sliding protection levels for each table cell

slot `sdcStatus`**:** character vector (or NULL) holding the current anonymization state for each cell.

- `z`: cell is forced to be published and must not be suppressed
- `u`: cell has been primary suppressed
- `x`: cell is a secondary suppression
- `s`: cell can be published

Note

objects of class `problemInstance` are used as input for slot `problemInstance` in class `sdcProblem`

Author(s)

Bernhard Meindl <bernhard.meindl@statistik.gv.at>

`protectLinkedTables` *Protect two tables with common cells*

Description

`protect_linked_tables()` can be used to protect tables that have common cells. It is of course required that after the anonymization process has finished, all common cells have the same anonymization state in both tables.

Usage

```
protectLinkedTables(
  objectA,
  objectB,
  commonCells,
  method = "SIMPLEHEURISTIC",
  ...
)

protect_linked_tables(x, y, common_cells, method = "SIMPLEHEURISTIC", ...)
```

Arguments

objectA	maps to argument x in protect_linked_tables()
objectB	maps to argument y in protect_linked_tables()
commonCells	maps to argument common_cells in protect_linked_tables()
method	which protection algorithm should be used; choices are "SIMPLEHEURISTIC" and "SIMPLEHEURISTIC_OLD"
...	additional arguments to control the secondary cell suppression algorithm. For details, see protectTable().
x	a sdcProblem object
y	a sdcProblem object
common_cells	a list object defining common cells in x and y. For each variable that has one or more common codes in both tables, a list element needs to be specified.

- List-elements of length 3: Variable has exact same levels and structure in both input tables
  - first element: scalar character vector specifying the variable name in argument x
  - second element: scalar character vector specifying the variable name in argument y
  - third element: scalar character vector being with keyword "ALL"
- List-elements of length 4: Variable has different codes and levels in inputs x and y
  - first element: scalar character vector specifying the variable name in argument x
  - second element: scalar character vector specifying the variable name in argument y
  - third element: character vector defining codes within x
  - fourth element: character vector with length that equals the length of the third list-element. This vector defines codes of the dimensional variable in y that match the codes given in the third list-element for x.

Value

a list elements x and y containing protected sdcProblem objects

Author(s)

Bernhard Meindl <bernhard.meindl@statistik.gv.at>

See Also

protectTable()

Examples

```
## Not run:
# load micro data for further processing
utils::data("microdata2", package = "sdcTable")

# table1: defined by variables 'gender' and 'ecoOld'
md1 <- microdata2[,c(2,3,5)]

# table2: defined by variables 'region', 'gender' and 'ecoNew'
md2 <- microdata2[,c(1,2,4,5)]

# we need to create information on the hierarchies
# variable 'region': exists only in md2
d_region <- hier_create(root = "Tot", nodes = c("R1", "R2"))

# variable 'gender': exists in both datasets
d_gender <- hier_create(root = "Tot", nodes = c("m", "f"))

# variable 'eco1': exists only in md1
d_eco1 <- hier_create(root = "Tot", nodes = c("A", "B"))
d_eco1 <- hier_add(d_eco1, root = "A", nodes = c("Aa", "Ab"))
d_eco1 <- hier_add(d_eco1, root = "B", nodes = c("Ba", "Bb"))

# variable 'ecoNew': exists only in md2
d_eco2 <- hier_create(root = "Tot", nodes = c("C", "D"))
d_eco2 <- hier_add(d_eco2, root = "C", nodes = c("Ca", "Cb", "Cc"))
d_eco2 <- hier_add(d_eco2, root = "D", nodes = c("Da", "Db", "Dc"))

# creating objects holding information on dimensions
dl1 <- list(gender = d_gender, ecoOld = d_eco1)
dl2 <- list(region = d_region, gender = d_gender, ecoNew = d_eco2)

# creating input objects for further processing.
# For details, see ?makeProblem.
p1 <- makeProblem(
  data = md1,
  dimList = dl1,
  dimVarInd = 1:2,
  numVarInd = 3)

p2 <- makeProblem(
  data = md2,
  dimList = dl2,
  dimVarInd = 1:3,
  numVarInd = 4)

# the cell specified by gender == "Tot" and ecoOld == "A"
# is one of the common cells! -> we mark it as primary suppression
p1 <- change_cellstatus(
  object = p1,
  specs = data.frame(gender = "Tot", ecoOld = "A"),
  rule = "u",
```

```
  verbose = FALSE)

# the cell specified by region == "Tot" and gender == "f" and ecoNew == "C"
# is one of the common cells! -> we mark it as primary suppression
p2 <- change_cellstatus(
  object = p2,
  specs = data.frame(region = "Tot", gender = "f", ecoNew = "C"),
  rule = "u",
  verbose = FALSE)

# specifying input to define common cells
common_cells <- list()

# variable "gender"
common_cells$v.gender <- list()
common_cells$v.gender[[1]] <- "gender" # variable name in "p1"
common_cells$v.gender[[2]] <- "gender" # variable name in "p2"

# "gender" has equal characteristics on both datasets -> keyword "ALL"
common_cells$v.gender[[3]] <- "ALL"

# variables: "ecoOld" and "ecoNew"
common_cells$v.eco <- list()
common_cells$v.eco[[1]] <- "ecoOld" # variable name in "p1"
common_cells$v.eco[[2]] <- "ecoNew" # variable name in "p2"

# vector of common characteristics:
# "A" and "B" in variable "ecoOld" in "p1"
common_cells$v.eco[[3]] <- c("A", "B")

# correspond to codes "C" and "D" in variable "ecoNew" in "p2"
common_cells$v.eco[[4]] <- c("C", "D")

# protect the linked data
result <- protect_linked_tables(
  x = p1,
  y = p2,
  common_cells = common_cells,
  verbose = TRUE)

# having a look at the results
result_tab1 <- result$x
result_tab2 <- result$y
summary(result_tab1)
summary(result_tab2)

## End(Not run)
```

---

protectTable *Protecting sdcProblem objects*

Description

Function `protectTable()` is used to protect primary sensitive table cells (that usually have been identified and set using `primarySuppression()`). The function protects primary sensitive table cells according to the method that has been chosen and the parameters that have been set. Additional parameters that are used to control the protection algorithm are set using parameter ....

Usage

```
protectTable(object, method, ...)
```

Arguments

`object` a sdcProblem object that has created using `makeProblem()` and has been modified by `primarySuppression()`

`method` a character vector of length 1 specifying the algorithm that should be used to protect the primary sensitive table cells. Allowed values are:

- `"OPT"`: protect the complete problem at once using a cut and branch algorithm. The optimal algorithm should be used for small problem-instances only.
- `"HITAS"`: split the overall problem in smaller problems. These problems are protected using a top-down approach.
- `"HYPERCUBE"`: protect the complete problem by protecting sub-tables with a fast heuristic that is based on finding and suppressing geometric structures (n-dimensional cubes) that are required to protect primary sensitive table cells.
- `"SIMPLEHEURISTIC"` and `"SIMPLEHEURISTIC_OLD"`: heuristic procedures which might be applied to large(r) problem instances;
  - `"SIMPLEHEURISTIC"` is based on constraints; it also solves attacker problems to make sure each primary sensitive cell cannot be recomputed;
  - `"SIMPLEHEURISTIC_OLD"` was the implementation in `sdcTable` versions prior to `0.32`; this implementation is possibly unsafe but very fast; it is advised to check results using `attack()` afterwards.

`...` parameters used in the protection algorithm that has been selected. Parameters that can be changed are:

- **general parameters**:
  - `verbose`: logical scalar (default is `FALSE`) defining if verbose output should be produced
  - `save`: logical scalar defining if temporary results should be saved in the current working directory (`TRUE`) or not (`FALSE`) which is the default value.
- parameters used for **"HITAS"** and **"OPT"** algorithms:
  - `solver`: character vector of length 1 defining the solver to be used. Currently available choices are limited to `"highs"`.

  - timeLimit: numeric vector of length 1 (or NULL) defining a time limit in minutes after which the cut and branch algorithm should stop and return a possible non-optimal solution. Parameter safe has a default value of NULL
  - maxVars: a integerish number (or NULL) defining the maximum problem size in terms of decision variables for which an optimization should be tried. If the number of decision variables in the current problem are larger than parameter maxVars, only a possible non-optimal, heuristic solution is calculated. Parameter maxVars has a default value of NULL (no restrictions)
  - fastSolution: logical scalar defining (default FALSE) if or if not the cut and branch algorithm will be started or if the possibly non-optimal heuristic solution is returned independent of parameter maxVars.
  - fixVariables: logical scalar (default TRUE) defining whether or not it should be tried to fix some variables to 0 or 1 based on reduced costs early in the cut and branch algorithm.
  - approxPerc: integerish scalar that defines a percentage for which a integer solution of the cut and branch algorithm is accepted as optimal with respect to the upper bound given by the (relaxed) solution of the master problem. Its default value is set to 10
  - useC: logical scalar defining if c++ implementation of the secondary cell suppression problem should be used, defaults to FALSE

- parameters used for **"HYPERCUBE"** procedure:
  - protectionLevel: numeric vector of length 1 specifying the required protectionlevel for the procedure. Its default value is 80
  - suppMethod: character vector of length 1 defining the rule on how to select the 'optimal' cube to protect a single sensitive cells. Possible choices are:
    * minSupps: minimize the number of additional secondary suppressions (this is also the default setting).
    * minSum: minimize the sum of counts of additional suppressed cells
    * minSumLogs: minimize the log of the sum of additional suppressed cells
  - suppAdditionalQuader: logical vector of length 1 specfifying if additional cubes should be suppressed if any secondary suppressions in the 'optimal' cube are 'singletons'. Parameter suppAdditionalQuader has a default value of FALSE

- parameter(s) used for protect_linked_tables():
  - maxIter: integerish number specifying the maximal number of interations that should be make while trying to protect common cells of two different tables. The default value of parameter is 10

- parameters used for the **"SIMPLEHEURISTIC"** and **"SIMPLEHEURISTIC_OLD"** procedure:
  - detectSingletons: logical, should a singleton-detection procedure be run before protecting the data, defaults to FALSE.

  - threshold: if not NULL (the default) an integerish number (> 0). If specified, a procedure similar to the singleton-detection procedure is run that makes sure that for all (simple) rows in the table instance that contains primary sensitive cells the suppressed number of contributors is >= the specified threshold.
- parameters used for the **"GAUSS"** procedure; for details please see ?SSBtools::GaussSuppressio as the default values are the same as in this function:
  - removeDuplicated: should duplicated columns be removed before running the protection algorithm
  - whenEmptySuppressed: a function to be called when primary suppressed input is problematic; NULL (default) does not apply any function
  - whenEmptyUnsuppressed: a function to be called when empty candidate cells aredevto problematic; NULL (default) does not apply any function
  - singletonMethod: parameter singletonMethod in SSBtools::GaussSuppression(); default "anySum"

Details

The implemented methods may have bugs that yield in not-fully protected tables. Especially the usage of "OPT", "HITAS" and "HYPERCUBE" in production is not suggested as these methods may eventually be removed completely. In case you encounter any problems, please report it or use Tau-Argus (https://research.cbs.nl/casc/tau.htm).

Value

an safeObj object

Author(s)

Bernhard Meindl <bernhard.meindl@statistik.gv.at>

Examples

```
## Not run:
# load example-problem with with a single primary suppression
# (same as example from ?primarySuppression)
p <- sdc_testproblem(with_supps = TRUE)

# protect the table using the 'GAUSS' algorithm with verbose output
res1 <- protectTable(p, method = "GAUSS", verbose = TRUE)
res1

# protect the table using the 'HITAS' algorithm with verbose output
res2 <- protectTable(p, method = "HITAS", verbose = TRUE, useC = TRUE)
res2

# protect using the heuristic algorithm
res3 <- protectTable(p, method = "SIMPLEHEURISTIC")
res3
```

```
# protect using the old implmentation of the heuristic algorithm
# used in sdcTable versions <0.32
res4 <- protectTable(p, method = "SIMPLEHEURISTIC_OLD")
res4

# looking at the final table with result suppression pattern
print(getInfo(res1, type = "finalData"))

## End(Not run)
```

---

runArgusBatchFile *runArgusBatchFile*

Description

allows to run batch-files for tau argus given the path to an executable of argus. The provided batch input files can either be created using function createArgusInput or can be arbitrarily created. In the latter case, argument obj should not be specified and not output is returned, the script is just executed in tau-argus.

Usage

```
runArgusBatchFile(
  obj = NULL,
  batchF,
  exe = "C:\\Tau\\TauArgus.exe",
  batchDataDir = NULL,
  verbose = FALSE
)
```

Arguments

obj	NULL or an object of class sdcProblem-class that was used to generate the batchfile for argus. If not NULL, this object is used to create correct variable names. Else, only the output from tau-Argus is read and returned as a data.table. In this case it is possible to run tau-Argus on arbitrarily created batch-files.
batchF	a filepath to an batch-input file created by e.g. createArgusInput.
exe	(character) file-path to tau-argus executable
batchDataDir	if different from NULL, this directory is used to look for input-file and writes output files to. This helps to use relative paths in batch input files.
verbose	(logical) if TRUE, some additional information is printed to the prompt

Value

a data.table containing the protected table or an error in case the batch-file was not solved correctly if the batch-file was created using sdcTable (argument obj) was specified. In case an arbitrarily batch-file has been run, NULL is returned.

Note

in case a custom batch-file is used as input (e.g `obj` is `NULL`), this functions does currently not try to read in any tables to the system.

`safeObj-class` *S4 class describing a safeObj-object*

Description

Objects of class `safeObj` are the final result after protection a tabular structure. After a successful run of `protectTable` an object of this class is generated and returned. Objects of class `safeObj` contain a final, complete data set (slot `finalData`) that has a column showing the anonymization state of each cell and the complete information on the dimensional variables that have defined the table that has been protected (slot `dimInfo`). Also, the number of non-duplicated table cells (slot `nrNonDuplicatedCells`) is returned along with the number of primary (slot `nrPrimSupps`) and secondary (slot `nrSecondSupps`) suppressions. Furthermore, the number of cells that can be published (slot `nrPublishableCells`) and the algorithm that has been used to protect the data (slot `suppMethod`) is returned.

Details

slot `finalData`**:** a data.frame (or NULL) featuring columns for each variable defining the table (with their original codes), the cell counts and values of any numerical variables and the anonymization status for each cell with

- `s`, `z`: cell can be published
- `u`: cell is a primary sensitive cell
- `x`: cell was selected as a secondary suppression

slot `dimInfo`**:** an object of class `dimInfo-class` holding all information on variables defining the table

slot `nrNonDuplicatedCells`**:** numeric vector of length 1 (or NULL) showing the number of non-duplicated table cells. This value is different from 0 if any dimensional variable features duplicated codes. These codes have been re-added to the final dataset.

slot `nrPrimSupps`**:** numeric vector of length 1 (or NULL) showing the number of primary suppressed cells

slot `nrSecondSupps`**:** numeric vector of length 1 (or NULL) showing the number of secondary suppressions

slot `nrPublishableCells`**:** numeric vector of length 1 (or NULL) showing the number of cells that may be published

slot `suppMethod`**:** character vector of length 1 holding information on the protection method

Note

objects of class `safeObj` are returned after the function `protectTable` has finished.

Author(s)

Bernhard Meindl <bernhard.meindl@statistik.gv.at>

---

sdcProb2df *Transform a problem instance*

Description

sdcProb2df() returns a data.table given an sdcProblem input object.

Usage

```
sdcProb2df(obj, addDups = TRUE, addNumVars = FALSE, dimCodes = "both")
```

Arguments

obj	an sdcProblem object
addDups	(logical), if TRUE, duplicated cells are included in the output
addNumVars	(logical), if TRUE, numerical variables (if defined in makeProblem() will be included in the output.
dimCodes	(character) allows to specify in which coding the dimensional variables should be returned. Possible choices are: • "both": both original and internally used, standardized codes are included in the output • "original": only original codes of dimensional variables are included in the output • "default": only internally used, standardized codes are included in the output

Value

a data.table containing information about all cells of the given problem

Examples

```
# loading micro data
utils::data("microdata1", package = "sdcTable")

# we can observe that we have a micro data set consisting
# of two spanning variables ('region' and 'gender') and one
# numeric variable ('val')

# specify structure of hierarchical variable 'region'
# levels 'A' to 'D' sum up to a Total
dim.region <- data.frame(
 levels=c('@','@@','@@','@@','@@'),
```

```
 codes=c('Total', 'A','B','C','D'),
 stringsAsFactors=FALSE)

# specify structure of hierarchical variable 'gender'
# using create_node() and add_nodes() (see ?manage_hierarchies)
dim.gender <- hier_create(root = "Total", nodes = c("male", "female"))
hier_display(dim.gender)

# create a named list with each element being a data-frame
# containing information on one dimensional variable and
# the names referring to variables in the input data
dimList <- list(region = dim.region, gender = dim.gender)

# third column containts a numeric variable
numVarInd <- 3

# no variables holding counts, numeric values, weights or sampling
# weights are available in the input data
# creating an problem instance using numeric indices
p1 <- makeProblem(
  data = microdata1,
  dimList = dimList,
  numVarInd = 3 # third variable in `data`
)

# using variable names is also possible
p2 <- makeProblem(
  data = microdata1,
  dimList = dimList,
  numVarInd = "val"
)

# what do we have?
print(class(p1))

# have a look at the data
df1 <- sdcProb2df(p1, addDups = TRUE,
  addNumVars = TRUE, dimCodes = "original")
df2 <- sdcProb2df(p2, addDups=TRUE,
  addNumVars = TRUE, dimCodes = "original")
print(df1)

identical(df1, df2)
```

`sdcProblem-class` *S4 class describing a sdcProblem-object*

Description

An object of class `sdcProblem` contains the entire information that is required to protect the complete table that is given by the dimensional variables. Such an object holds the data itself (slot

dataObj), the entire information about the dimensional variables (slot dimInfo), information on all table cells (ID's, bounds, values, anonymization state in slot problemInstance), the indices on the sub tables that need to be considered if one wants to protect primary sensitive cells using a heuristic approach (slot partition and the information on which groups or rather subtables have already been protected while performing a heuristic method (slots startI and startJ).

Details

slot dataObj**:** an object of class dataObj (or NULL) holding information on the underlying data

slot dimInfo**:** an object of class dimInfo (or NULL) containing information on all dimensional variables

slot problemInstance**:** an object of class problemInstance holding information on values, bounds, required protection levels as well as the anonymization state for all table cells

slot partition**:** a list object (or NULL) that is typically generated with calc.multiple(type='makePartitions',...) specifying information on the subtables and the necessary order that need to be protected when using a heuristic approach to solve the cell suppression problem

slot startI**:** a numeric vector of length 1 defining the group-level of the subtables in which a heuristic algorithm needs to start. All subtables having a group-index less than startI have already been protected

slot startJ**:** a numeric vector of length 1 defining the number of the table within the group defined by parameter startI at which a heuristic algorithm needs to start. All tables in the group having an index j smaller than startJ have already been protected

slot indicesDealtWith**:** a numeric vector holding indices of table cells that have protected and whose anonymization state must remain fixed

Note

objects of class sdcProblem are typically generated by function makeProblem and are the input of functions primarySuppression and protectTable

Author(s)

Bernhard Meindl <bernhard.meindl@statistik.gv.at>

---

sdc_testproblem *A Problem-Instance used for examples/testing*

Description

sdc_testproblem() returns a sdc-problem instance with 2 hierarchies and optionally with a single suppressed cell that is used in various examples and tests.

Usage

```
sdc_testproblem(with_supps = FALSE)
```

Arguments

with_supps	if TRUE, a single cell (violating minimal-frquency rule with n = 2) is marked as primary sensitive.

Value

a problem instance

Examples

```
p1 <- sdc_testproblem(); p1
sdcProb2df(p1)

# a single protected cell
p2 <- sdc_testproblem(with_supps = TRUE); p2
sdcProb2df(p2)

# cell status differs in one cell
specs <- c(gender = "female", region = c("A"))
cell_info(p1, specs = specs)
cell_info(p2, specs = specs)
```

---

set.dimInfo *modify* dimInfo*-objects depending on argument* type

---

Description

modify dimInfo-objects depending on argument type

Usage

```
set.dimInfo(object, type, input)

## S4 method for signature 'dimInfo,character,character'
set.dimInfo(object, type, input)
```

Arguments

object	an object of class dimInfo
type	a character vector of length 1 defining what to calculate|return|modify. Allowed types are:

- strID: set slot 'strID' of argument object

input	a list depending on argument type.

- type==strID: a character vector containing ID's

Value

an object of class dimInfo

Note

internal function

Author(s)

Bernhard Meindl <bernhard.meindl@statistik.gv.at>

---

set.problemInstance *modify* problemInstance-*objects depending on argument* type

---

Description

modify problemInstance-objects depending on argument type

Usage

```
set.problemInstance(object, type, input)

## S4 method for signature 'problemInstance,character,list'
set.problemInstance(object, type, input)
```

Arguments

object	an object of class problemInstance
type	a character vector of length 1 defining what to calculate|return|modify. Allowed types are:

- lb: set assumed to be known lower bounds
- ub: set assumed to be upper lower bounds
- LPL: set lower protection levels
- UPL: set upper protection levels
- SPL: set sliding protection levels
- sdcStatus: change anonymization status

input	a list with elements 'indices' and 'values'.

- element 'indices': numeric vector defining the indices of the cells that should be modified
- element 'values': numeric vector whose values are going to replace current values for cells defined by 'indices' depending on argument type

Value

an object of class problemInstance

Note

internal function

Author(s)

Bernhard Meindl <bernhard.meindl@statistik.gv.at>

---

set.sdcProblem *modify* sdcProblem*-objects depending on argument* type

---

Description

modify sdcProblem-objects depending on argument type

Usage

```
set.sdcProblem(object, type, input)

## S4 method for signature 'sdcProblem,character,list'
set.sdcProblem(object, type, input)
```

Arguments

object	an object of class sdcProblem
type	a character vector of length 1 defining what to calculate|return|modify. Allowed types are:

- problemInstance: set|modify slot 'problemInstance' of argument object
- partition: set|modify slot 'partition' of argument object
- startI: set|modify slot 'startI' of argument object
- startJ: set|modify slot 'startJ' of argument object
- indicesDealtWith: set|modify slot 'indicesDealtWith' of argument object

input — a list with elements depending on argument type.

- an object of class problemInstance if argument type matches 'problemInstance'
- a list (derived from calc.multiple(type='makePartition', ...) if argument type matches 'partition'
- a numeric vector of length 1 if argument type matches 'startI' or 'startJ'
- a numeric vector if argument type matches 'indicesDealtWith'

Value

an object of class sdcProblem

Note

internal function

Author(s)

Bernhard Meindl <bernhard.meindl@statistik.gv.at>

---

setInfo *Set/Update information in* sdcProblem *or* problemInstance *objects*

Description

Function setInfo() is used to update values in sdcProblem or problemInstance objects

Usage

```
setInfo(object, type, index, input)
```

Arguments

object	an object of class sdcProblem or problemInstance
type	a scalar character specifying the kind of information that should be changed or modified; if object inherits class problemInstance, the slots are directly changed, otherwise the values within slot problemInstance are updated. Valid choices are: • lb: lower possible bounds for the cell • ub: max. upper bound for the given cell • LPL: lower protection level • SPL: sliding protection level • UPL: upper protection level • sdcStatus: cell-status
index	numeric vector defining cell-indices for which which values in a specified slot should be changed|modified
input	numeric or character vector depending on argument type with its length matching the length of argument index • character vector if type matches 'sdcStatus' • a numeric vector if type matches 'lb', 'ub', 'LPL', 'SPL' or 'UPL'

Value

a sdcProblem- or problemInstance object

Author(s)

Bernhard Meindl <bernhard.meindl@statistik.gv.at>

Examples

```
# load example-problem with suppressions
# (same as example from ?primarySuppression)
p <- sdc_testproblem(with_supps = TRUE)

# which is the overall total?
idx <- which.max(getInfo(p, "freq")); idx

# we see that the cell with idx = 1 is the overall total and its
# anonymization state of the total can be extracted as follows:
print(getInfo(p, type = "sdcStatus")[idx])

# we want this cell to never be suppressed
p <- setInfo(p, type = "sdcStatus", index = idx, input = "z")

# we can verify this:
print(getInfo(p, type = "sdcStatus")[idx])

# changing slot 'UPL' for all cells
inp <- data.frame(
  strID = getInfo(p, "strID"),
  UPL_old = getInfo(p, "UPL")
)
inp$UPL_new <- inp$UPL_old + 1
p <- setInfo(p, type = "UPL", index = 1:nrow(inp), input = inp$UPL_new)
```

---

show,sdcProblem-method

show objects of class sdcProblem-class.

Description

just calls the corresponding print-method

Usage

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

Arguments

object	an objects of class sdcProblem-class

simpleTriplet-class *S4 class describing a simpleTriplet-object*

Description

Objects of class simpleTriplet define matrices that are stored in a sparse format. Only the row- and column indices and the corresponding values of non-zero cells are stored. Additionally, the dimension of the matrix given by the total number of rows and columns is stored.

Details

slot i**:** a numeric vector specifying row-indices with each value being geq 1 and leq of the value in nrRows

slot j**:** a numeric vector specifying column-indices with each value being geq 1 and leq of the value in nrCols

slot v**:** a numeric vector specifying the values of the matrix in cells specified by the corresponding row- and column indices

slot nrRows**:** a numeric vector of length 1 holding the total number of rows of the matrix

slot nrCols**:** a numeric vector of length 1 holding the total number of columns of the matrix

Note

objects of class simpleTriplet are input of slot constraints in class linProb-class and slot slot con in class cutList-class

Author(s)

Bernhard Meindl <bernhard.meindl@statistik.gv.at>

summary,sdcProblem-method

summarize object of class sdcProblem-class *or* safeObj-class*.*

Description

extract and show relevant information stored in object ofs class sdcProblem-class or safeObj-class.

Usage

```
## S4 method for signature 'sdcProblem'
summary(object, ...)
```

Arguments

object	Objects of either class sdcProblem-class or safeObj-class.
...	currently not used.

writeJJFormat *Write a problem in jj-format to a file*

Description

This function allows to write a problem instance in JJ-Format to a file.

Usage

```
writeJJFormat(x, tabvar = "freqs", path = "out.jj", overwrite = FALSE)
```

Arguments

x	an input produced by `createJJFormat()`
tabvar	the name of the variable that will be used when producing the problem in JJ format. It is possible to specify `freqs` (the default) or the name of a numeric variable that was available in the sdcProblem object used in `makeProblem()`.
path	a scalar character defining the name of the file that should be written. This can be an absolute or relative URL; however the file must not exist.
overwrite	logical scalar, if `TRUE` the file specified in `path` will be overwritten if it exists

Value

invisibly the path to the file that was created.

Examples

```
# setup example problem
# microdata
utils::data("microdata1", package = "sdcTable")

# create hierarchies
dims <- list(
  region = sdcHierarchies::hier_create(root = "Total", nodes = LETTERS[1:4]),
  gender = sdcHierarchies::hier_create(root = "Total", nodes = c("male", "female")))

# create a problem instance
p <- makeProblem(
  data = microdata1,
  dimList = dims,
  numVarInd = "val")

# create suitable input for `writeJJFormat`
inp <- createJJFormat(p); inp

# write files to disk
# frequency table by default
writeJJFormat(
```

```
  x = inp,
  path = file.path(tempdir(), "prob_freqs.jj"),
  overwrite = TRUE
)

# or using the numeric variable `val` previously specified
writeJJFormat(
  x = inp,
  tabvar = "val",
  path = file.path(tempdir(), "prob_val.jj"),
  overwrite = TRUE
)
```

Index

∗ **datasets**
    microdata1, 30
    microdata2, 31

argusVersion, 3
attack, 3
attack(), *39*

calc.sdcProblem, 5
calc.sdcProblem,sdcProblem,character,list-method
        *(*calc.sdcProblem*)*, 5
cell_info, 8
cellInfo(), *8*
change_cellstatus, 9
change_cellstatus(), *9*
contributing_indices, 11
createArgusInput, 12, *42*
createJJFormat, 15
createJJFormat(), *53*
createRegSDCInput, 16
cutList-class, 18

dataObj-class, 19
dimInfo-class, 20
dimVar-class, 20, *33*

get.dimInfo, 21
get.dimInfo,dimInfo,character-method
        *(*get.dimInfo*)*, 21
get.problemInstance, 22
get.problemInstance,problemInstance,character-method
        *(*get.problemInstance*)*, 22
get.sdcProblem, 24
get.sdcProblem,sdcProblem,character-method
        *(*get.sdcProblem*)*, 24
getInfo, 25
getInfo(), *25*

linProb-class, 27

makeProblem, 28, *46*
makeProblem(), *11*, *28*, *32*, *39*, *44*, *53*
microdata1, 30
microdata2, 31

primarySuppression, 31, *46*
primarySuppression(), *31*, *39*
print,dimVar-method, 33
print,sdcProblem-method, 34
problemInstance-class, 34
protect_linked_tables
        *(*protectLinkedTables*)*, 35
protect_linked_tables(), *35, 36*, *40*
protectLinkedTables, 35
protectTable, 38, *43*, *46*
protectTable(), *8*, *36*, *39*

RegSDC::SuppressDec, *16*
runArgusBatchFile, 42

safeObj, *41*
safeObj-class, 43, *52*
sdc_testproblem, 46
sdc_testproblem(), *46*
sdcProb2df, 44
sdcProb2df(), *11*, *44*
sdcProblem, *8–12*, *15–17*, *28, 29*, *31, 32*, *36*, *38, 39*, *44*, *53*
sdcProblem-class, *34*, 45, *51, 52*
set.dimInfo, 47
set.dimInfo,dimInfo,character,character-method
        *(*set.dimInfo*)*, 47
set.problemInstance, 48
set.problemInstance,problemInstance,character,list-method
        *(*set.problemInstance*)*, 48
set.sdcProblem, 49
set.sdcProblem,sdcProblem,character,list-method
        *(*set.sdcProblem*)*, 49
setInfo, 50
setInfo(), *50*
show,sdcProblem-method, 51

simpleTriplet-class, 52
SSBtools::GaussSuppression(), *41*
summary,sdcProblem-method, 52

writeJJFormat, 53
writeJJFormat(), *15*, *16*