Package 'sdcTable'

June 22, 2018

Version 0.23

Date 2018-06-22

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.

Author Bernhard Meindl

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

URL http://www.statistik.at

BugReports https://github.com/bernhard-da/sdcTable/issues

Depends R (>= 2.10), stringr, methods, Rcpp (>= 0.11.0), Rglpk, slam,
lpSolveAPI

Imports data.table, knitr, data.tree, rlang

Suggests testthat (>= 0.3), rmarkdown, webshot

LazyLoad yes

LinkingTo Rcpp

License GPL (>= 2)

SystemRequirements GLPK library, including -dev or -devel part

RoxygenNote 6.0.1

VignetteBuilder knitr

NeedsCompilation yes

Repository CRAN

Date/Publication 2018-06-22 21:04:46 UTC

R topics documented:

`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 and calculating current lower and upper bounds*

Description

Function attack is used to calculate lower and upper bounds for a given sdcProblem (stored as object of class sdcProblem-class). For all calculations the current suppression pattern is used when calculating solutions of the attacker's problem.

Usage

```
attack(object, verbose = FALSE)
```

Arguments

object	an object of class sdcProblem-class
verbose	a logical vector specifying if output should be verbose (TRUE) or not (FALSE)

Value

a data.frame with column 'index' holding indices of primary suppressed cells and columns 'bounds_min' and 'bounds_max' featuring calculated lower and upper bounds for each cell. Column 'protected' shows if a given cell is accordingly protected (TRUE) or not (FALSE).

Author(s)

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

Examples

```
# load problem (as it was created after performing primary suppression
# in the example of \code{\link{primarySuppression}})
sp <- searchpaths()
fn <- paste(sp[grep("sdcTable", sp)], "/data/problemWithSupps.RData", sep="")
problem <- get(load(fn))

# calculate current lower|upper bounds given current suppression pattern
# (in this case consisting of primary suppressions only)
attack(problem, verbose=FALSE)
```

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

Description

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

Usage

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

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

Arguments

`object`	an object of class `cutList`
`type`	a character vector of length 1 defining what to calculate|return|modify. Allowed types are: • strengthen: strenghten constraints in argument `object` • checkViolation: check if a given solution violates any in argument `object` • bindTogether: combine two `cutList`-objects
`input`	a list depending on argument `type`. • type==strengthen: input is not used (empty list) • type==checkViolation: input is a list of length 2 – first element: numeric vector specifying a solution to a linear problem – second element: numeric vector specifying weights • type==bindTogether: input is a list of length 1 – first element: object of class `cutList`

Value

manipulated data based on argument `type`

- an object of class `cutList` if argument `type` matches 'strengthen' or 'bindTogether'
- a logical vector of length 1 if argument `type` matches 'checkViolation' with TRUE if at least one constraint is violated by the given solution

Note

internal function

Author(s)

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

`calc.dimVar` *modify* `dimVar`*-objects depending on argument* `type`

Description

modify `dimVar`-objects depending on argument `type`

Usage

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

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

Arguments

`object`	an object of class `dimVar`
`type`	a character vector of length 1 defining what to calculate|return|modify. Allowed types are: • hasDefaultCodes: calculates if a vector of codes (specified by argument `input`) corresponds to default codes in `object` • matchCodeOrig: obtain default|standard codes for a vector of original codes specified by argument `input` • matchCodeDefault: obtain original codes for a vector of default|standard codes specified by argument `input` • standardize: perform standardization of level-codes (temporarily removing duplicates,..) • requiredMinimalCodes: calculate a set of minimal codes that are required to calculate a specific (sub)total specified by argument `input`
`input`	a character vector

Value

information from `object` depending on `type`

- a character vector if type matches 'matchCodeOrig', 'matchCodeDefault', 'standardize' or 'requiredMinimalCodes'
- a logical vector of length 1 if type matches 'hasDefaultCodes' being TRUE if argument `input` are default codes and FALSE otherwise

Note

internal function

Author(s)

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

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

Description

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

Usage

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

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

Arguments

`object`	an object of class `linProb`
`type`	a character vector of length 1 defining what to calculate|return|modify. Allowed types are: • solveProblem: solve the linear problem (minimize objective function) • fixVariables: try to fix objective variables to 0|1 based on dual costs depending on input
`input`	a list depending on argument `type`. • type==solveProblem: a list of length 1 – first element: character vector of length 1 specifying the solver to use. • type==fixVariables: a list of length 3 – first element: numeric vector specifying lower bounds for the objective variables – second element: numeric vector specifying upper bounds for the objective variables – third element: numeric vector specifying indices of primary suppressed cells

Value

manipulated data based on argument `type`

- list containing the solution and additional information if argument `type` matches 'solveProblem
- a numeric vector of indices if argument `type` matches 'fixVariables'

Note

internal function

Author(s)

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

`calc.multiple` *perform calculations on multiple objects depending on argument* `type`

Description

perform calculations on multiple objects depending on argument `type`

Usage

```
calc.multiple(type, input)

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

Arguments

`type`	a character vector of length 1 defining what to calculate|return|modify. Allowed types are: • makePartitions: information on subtables required for HITAS and HYPECUBE algorithms • genMatMFull: the constraint matrix used in the master problem • makeAttackerProblem: set up the attackers problem for a given (sub)table • calcFullProblem: calculate a complete problem object containing all information required to solve the secondary cell suppression problem
`input`	a list depending on argument `type`. • if type matches 'makePartitions', 'genMatMFull' or 'makeAttackerProblem': a list of length 2 with elements 'objectA' and 'objectB' – element 'object A': an object of class `problemInstance` – element 'object B': an object of class `dimInfo` • type matches 'calcFullProblem': a list of length 1 – element 'object A': an object of class `dataObj` – element 'object B': an object of class `dimInfo`

Value

manipulated data based on argument `type`

- list with elements 'groups', 'indices', 'strIDs', 'nrGroups' and 'nrTables' if argument `type` matches 'makePartitions'
- object of class `simpleTriplet` if argument `type` matches 'genMatMFull'
- object of class `linProb` if argument `type` matches 'makeAttackerProblem'
- object of class `sdcProblem` if argument `type` matches 'calcFullProblem'

Note

internal function

Author(s)

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

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

Description

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

Usage

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

## S4 method for signature 'problemInstance,character,list'
calc.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: • makeMasterProblem: create the master problem that is the core of the secondary cell suppression problem • isProtectedSolution: check if a solution violates any required (upper|lower|sliding) protection levels
`input`	a list depending on argument `type`. • type==makeMasterProblem: input is not used (empty list) • type==isProtectedSolution: input is a list of length 2 with elements 'input1' and 'input2' – element 'input1': numeric vector of calculated known lower cell bounds (from attacker's problem) – element 'input2': numeric vector of known upper cell bounds (from attacker's problem)

Value

information from objects of class problemInstance depending on argument type

- an object of class linProb if argument type matches 'makeMasterProblem'
- logical vector of length 1 if argument type matches 'isProtectedSolution' with TRUE if all primary suppressed cells are adequately protected, FALSE otherwise

Note

internal function

Author(s)

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

---

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 • rule.nk: modify sdcStatus of object according to nk-dominance rule • rule.p: modify sdcStatus of object according to p-percent rule • rule.pq: modify sdcStatus of object according to pq-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', 'rule.nk' or 'rule.p'
- 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', 'rule.nk', 'rule.p', '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>

`calc.simpleTriplet` *modify* `simpleTriplet`-*objects depending on argument* `type`

Description

modify `simpleTriplet`-objects depending on argument `type`

Usage

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

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

Arguments

`object`	an object of class `simpleTriplet`
`type`	a character vector of length 1 defining what to calculate|return|modify. Allowed types are: • removeRow: remove a row with given index from `object` • removeCol: remove a column with given index from `object` • addRow: add a row to `object` • addCol: add a column to `object` • modifyRow: change specified row of `object` • modifyCol: change specified column of `object` • modifyCell: change specified cell of `object` • bind: bind two objects of class `simpleTriplet` together
`input`	a list depending on argument `type`. • type==removeRow: input is a list of length 1 – first element: numeric vector of length 1 defining the index of the row that should be removed • type==removeCol: input is a list of length 1 – first element: numeric vector of length 1 defining the index of the column that should be removed • type==addRow: input is a list of length 2 – first element: numeric vector of column-indices – second element: numeric vector defining the cell-values of the row that will be added • type==addCol: input is a list of length 2 – first element: numeric vector of row-indices – second element: numeric vector defining the cell-values of the column that will be added • type==modifyRow: input is a list of length 3

  - first element: numeric vector of length 1 specifying the the row-index of the row that will be modified
  - second element: numeric vector specifying the column-indices that should be modified
  - third element: numeric vector defining values that should be set in the given row
- type==modifyCol: input is a list of length 3
  - first element: numeric vector specifying the row-indices that should be modified
  - second element: numeric vector of length 1 specifying the the column-index of the column that will be modified
  - third element: numeric vector defining values that should be set in the given column
- type==modifyCell: input is a list of length 3
  - first element: numeric vector of length 1 defining the column-index
  - second element: numeric vector of length 1 defining the row-index
  - third element: numeric vector of length 1 holding the value that should be set in the given cell
- type==bind: input is a list of length 2
  - first element: an object of class `simpleTriplet`
  - second argument: is a logical vector of length 1 being TRUE if a 'rbind' or 'FALSE' if a 'cbind' should be done

Value

an object of class `simpleTriplet`

Note

internal function

Author(s)

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

`cellInfo` *query information for a specific cell in* `safeObj-class` *objects*

Description

Function `cellInfo` is used to query information for a single table cell for objects of class `safeObj-class`.

Usage

```
cellInfo(object, characteristics, varNames, verbose = FALSE)
```

Arguments

object	an object of class safeObj-class
characteristics	a character vector specifying characteristics of the table cell that should be identified for each dimensional variable defining the table
varNames	a character vector specifying variable names of dimensional variables defining the tables
verbose	logical vector of length 1 defining verbosity, defaults to 'FALSE'

Value

a list containing the following calculated information

- cellID: numeric vector of length 1 specifying the index of the cell within the final result dataset
- data: a data.frame containing a single row with the index of the table cell of interest
- primSupp: logical vector of length 1 that is 'TRUE' if the cell is a primary sensitive cell and 'FALSE' otherwise
- secondSupp: logical vector of length 1 that is 'TRUE' if the cell is a secondary suppressed cell and 'FALSE' otherwise

Note

Important: the i-th element of argument characteristics is uses as the desired characteristic for the dimensional variable specified at the i-th position of argument varNames!

Author(s)

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

Examples

```
# load protected data (as created in the example
# of \code{\link{protectTable}})
sp <- searchpaths()
fn <- paste(sp[grep("sdcTable", sp)], "/data/protectedData.RData", sep="")
protectedData <- get(load(fn))
characteristics <- c('male', 'D')
varNames <- c('gender', 'region')
info <- cellInfo(protectedData, characteristics, varNames, verbose=FALSE)

# show the info about this cell
str(info)
```

changeCellStatus *change anonymization status of a specific cell*

Description

Function changeCellStatus allows to change|modify the anonymization state of single table cells for objects ofs class sdcProblem-class.

Usage

```
changeCellStatus(object, characteristics, varNames, rule, verbose = FALSE)
```

Arguments

object	an object of class sdcProblem-class
characteristics	a character vector specifying characteristics of the table cell that should be identified for each dimensional variable defining the table
varNames	a character vector specifying variable names of dimensional variables defining the tables
rule	character vector of length 1 specifying a valid anonymization code ('u', 'z', 'x', 's') to which the the cell under consideration should be set.
verbose	logical vector of length 1 defining verbosity, defaults to 'FALSE'

Value

a sdcProblem-class object

Note

Important: the i-th element of argument characteristics is uses as the desired characteristic for the dimensional variable specified at the i-th position of argument varNames!

Author(s)

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

Examples

```
# load primary suppressed data (as created in the example
# of \code{\link{primarySuppression}})
sp <- searchpaths()
fn <- paste(sp[grep("sdcTable", sp)], "/data/problemWithSupps.RData", sep="")
problem <- get(load(fn))

# we want to mark the cell region='D' and gender='male' primary sensitive
characteristics <- c('D', 'male')
```

```
varNames <- c('region', 'gender')
verbose <- TRUE
rule <- 'u'

# looking at the distribution of anonymization states before...
print(table(getInfo(problem, 'sdcStatus')))

# setting the specific cell as primary sensitive
problem <- changeCellStatus(problem, characteristics, varNames, rule, verbose)

# having a second look at the anonymization states
print(table(getInfo(problem, 'sdcStatus')))
```

---

createArgusInput *createArgusInput*

Description

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

```
# loading micro data from sdcTable
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 <- data.frame(
  levels=c('@','@@','@@','@@','@@'),
  codes=c('Total', 'A','B','C','D'),
  stringsAsFactors=FALSE)

dim.region_dupl <- data.frame(
  levels=c('@','@@','@@','@@@','@@','@@','@@@'),
  codes=c('Total', 'A','B','b1','C','D','d1'),
  stringsAsFactors=FALSE)

dim.gender <- data.frame(
  levels=c('@','@@','@@'),
  codes=c('Total', 'male','female'),
  stringsAsFactors=FALSE)

dimList <- list(region=dim.region, gender=dim.gender)
dimList_dupl  <- list(region=dim.region_dupl, gender=dim.gender)
dimVarInd <- c(1,2)
numVarInd <- 3:5
sampWeightInd <- 6

freqVarInd <- weightInd <- NULL

## creating an object of class \code{\link{sdcProblem-class}}
obj <- makeProblem(
  data=microData1,
  dimList=dimList,
  dimVarInd=dimVarInd,
  freqVarInd=freqVarInd,
  numVarInd=numVarInd,
  weightInd=weightInd,
  sampWeightInd=sampWeightInd)

## creating an object of class \code{\link{sdcProblem-class}} containing "duplicated" codes
obj_dupl <- makeProblem(
  data=microData1,
  dimList=dimList_dupl,
  dimVarInd=dimVarInd,
  freqVarInd=freqVarInd,
  numVarInd=numVarInd,
  weightInd=weightInd,
  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, typ="microdata", path=getwd(), solver="FREE", method="OPT",
  primSuppRules=primSuppRules, responsevar="num1")
bO_td1 <- createArgusInput(obj, typ="tabular", path=getwd(), solver="FREE", method="OPT")
bO_td2 <- createArgusInput(obj_dupl, typ="tabular", path=getwd(), solver="FREE", method="OPT")
## Not run:
## in case CPLEX should be used, it is required to specify argument licensefile
bO_md2 <- createArgusInput(obj, typ="microdata", path=getwd(), solver="CPLEX", method="OPT",
 primSuppRules=primSuppRules, responsevar="num1", licensefile="/path/to/my/cplexlicense")

## 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.cutList` *query* `cutList`*-objects depending on argument* `type`

Description

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

Usage

```
get.cutList(object, type)

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

Arguments

object	an object of class cutList
type	a character vector of length 1 defining what to calculate|return|modify. Allowed types are: • constraints: constraint matrix of object • direction: directions of the constraints • rhs: right hand side of the constraints • nrConstraints: total number of constraints

Value

information from objects of class cutList depending on argument type

- object of class simpleTriplet if argument type matches 'constraints'
- character vector if argument type matches 'direction'
- numeric vector if argument type matches 'objective' or 'rhs'

Note

internal function

Author(s)

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

---

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

---

Description

query dataObj-objects depending on argument type

Usage

```
get.dataObj(object, type)

## S4 method for signature 'dataObj,character'
get.dataObj(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: • rawData: raw input data • dimVarInd: indices of dimensional variables • freqVarInd: index of frequency variable • numVarInd: indices of numerical variables • weightVarInd index of weight variable • sampWeightInd index of variable holding sampling weights • isMicroData does object consist of microdata? • numVarNames variable names of numerical variables • freqVarName variable name of frequency variable • varName variable names of dimensional variables

Value

information from objects of class dataObj depending on argument type

- a list if argument type matches 'rawData'
- numeric vector if argument type matches 'dimVarInd', 'freqVarInd', 'numVarInd', 'weightVarInd' or 'sampWeightInd'
- character vector if argument type matches 'numVarNames', 'freqVarName' or 'varName'
- logical vector of length 1 if argument type matches 'isMicroData'

Note

internal function

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.dimVar *query* dimVar*-objects depending on argument* type

---

Description

query dimVar-objects depending on argument type

Usage

```
get.dimVar(object, type)

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

Arguments

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

- varName: variable name of the variable from which object was calculated
- codesOriginal: original codes (as specified by the user)
- codesDefault: calculated, default codes
- codesMinimal: all codes required to calculate the complete hierarchy (no sub-totals)
- levels: level-structure of the dimensional variable
- structure: vector showing how many digits in the default codes are required for each level
- dims: list showing the complete hierarchy of the dimensional variable
- dups: vector of duplicated codes
- dupsUp: vector of codes that are the 'upper' levels to which the codes in dups correspond
- hasDuplicates: does the dimensional variable has codes that can be (temporarily) removed
- nrLevels: the total number of levels of a dimensional variable
- minimalCodesDefault: the standardized codes of the minimal set of required level-codes

Value

information from objects of class dataObj depending on argument type

- a list if argument type matches 'dims'
- numeric vector if argument type matches 'levels' or 'nrLevels'
- character vector if argument type matches 'codesOriginal', 'codesDefault', 'vName', 'dups', 'dupsUp' or 'minimalCodesDefault'
- logical vector of length 1 if argument type matches 'hasDuplicates'
- a logical vector if argument type matches 'codesMinimal'

Note

internal function

Author(s)

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

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

Description

query linProb-objects depending on argument type

Usage

```
get.linProb(object, type)

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

Arguments

object	an object of class linProb
type	a character vector of length 1 defining what to calculate|return|modify. Allowed types are: • constraints: constraint matrix of object linProb • direction: directions of the constraints • rhs: right hand side of the constraints • objective: objective function • types: types of the objective variables • bounds: bounds of the objective variables

Value

information from objects of class linProb depending on type

- an object of class simpleTriplet if type matches 'constraints'
- a character vector if type matches 'direction' or 'types'
- a numeric vector if type matches 'objective' or 'rhs'
- a list with elements 'lower' and 'upper' if type matches 'bounds'
  - element 'lower': a list with the first element containing indices and the second element containing corrsponding lower bounds
  - element 'upper': a list with the first element containing indices and the second element containing corrsponding upper bounds

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.safeObj` *query* `safeObj`*-objects depending on argument* `type`

Description

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

Usage

```
get.safeObj(object, type, input)
```

Arguments

`object`	an object of class `safeObj`
`type`	a character vector of length 1 defining what to calculate|return|modify. Allowed types are: • dimInfo: get infos on dimensional variables that formed the base of the protected data • elapsedTime: get elapsed time of the protection procedure • finalData: return final data object • nrNonDuplicatedCells: total number of cells that are duplicates • nrPrimSupps: total number of primary suppressed cells • nrSecondSupps: total number of secondary cell suppressions

- nrPublishableCells: total number of cells that can be published
- suppMethod: suppression method that has been used
- cellInfo: extract information about a specific cell
- cellID: calculate ID of a specific cell defined by level-codes and variable names

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

- type matches 'dimInfo', 'elapsedTime', 'finalData', 'nrNonDuplicatedCells', 'nrPrimSupps', 'nrSecondSupps', 'nrPublishableCells' or 'suppMethod': input is not used (empty list)
- type matches 'cellInfo' or 'cellID': input is a list of length 3
  - 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

Value

information from `object` depending on `type`

- an object of class `dimInfo` (or NULL) if type matches 'dimInfo'
- a numeric vector if type matches 'elapsedTime', 'nrNonDuplicatedCells', 'nrPrimSupps', 'nrSecondSupps', 'nrPublishableCells' or 'cellID'
- a character vector if type matches 'suppMethod'
- a data.frame if type matches 'finalData'
- a list if type matches 'cellInfo' containing the following elements:
  - element 'cellID': numeric vector of length 1 specifying the index of the cell of interest
  - element 'data': row of slot 'finalData' with the row being defined by the calculated `cellID`
  - element 'primSupp': logical vector of length 1 being TRUE if cell is a primary suppressed cell
  - element 'secondSupps': logical vector of length 1 being TRUE if cell is a secondary suppressed cell

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 • elapsedTime: the elapsed time of the protection algorithm so far • 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 if argument `type` matches 'elapsedTime'
- numeric vector of length 1 if argument `type` matches 'startI' or 'startJ'

Note

internal function

Author(s)

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

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

Description

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

Usage

```
get.simpleTriplet(object, type, input)

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

Arguments

`object`	an object of class `simpleTriplet`
`type`	a character vector of length 1 defining what to calculate|return|modify. Allowed types are: • rowInd: extract all row-indices

- colInd: extract all column-indices
- values: extract all values
- nrRows: return the number of rows of the input object
- nrCols: return the number of columns of the input object
- nrCells: return the number of cells (different from 0!)
- duplicatedRows: return a numeric vector showing indices of duplicated rows
- transpose: transpose input `object` and return the transposed matrix
- getRow: return a specific row of input `object`
- getCol: return a specific column of input `object`

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

- type == 'getRow': input is a list of length 1
  - first element: numeric vector of length 1 defining index of row that is to be returned
- type == 'getCol': input is a list of length 1
  - first element: numeric vector of length 1 defining index of column that is to be returned
- else: input is not used at all (empty list)

Value

information from `object` depending on `type`

- a numeric vector if type matches 'rowInd', 'colInd', 'values', 'nrRows', 'nrCols', 'nrCells' or 'duplicatedRows'
- an object of class `simpleTriplet` if type matches 'transpose', 'getRow' or 'getCol'

Note

internal function

Author(s)

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

`getInfo` *query information from objects*

Description

Function `getInfo` is used to query information from objects of class `sdcProblem-class`, `problemInstance-class` or `safeObj-class`

Usage

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

Arguments

`object` a sdcProblem-class object, problemInstance-class object or safeObj-class object.

`type` a character vector of length 1 specifying the information which should be returned.

- if argument `object` is of class `sdcProblem-class` or problemInstance-class, valid choices are:
  - `lb`: slot 'lb' of input `object` if it is of class problemInstance-class or this slot within slot 'problemInstance' if `object` is of class sdcProblem-class
  - `ub`: slot 'ub' of input `object` if it is of class problemInstance-class or this slot within slot 'problemInstance' if `object` is of class sdcProblem-class
  - `LPL`: slot 'LPL' of input `object` if it is of class problemInstance-class or this slot within slot 'problemInstance' if `object` is of class sdcProblem-class
  - `SPL`: slot 'SPL' of input `object` if it is of class problemInstance-class or this slot within slot 'problemInstance' if `object` is of class sdcProblem-class
  - `UPL`: slot 'UPL' of input `object` if it is of class problemInstance-class or this slot within slot 'problemInstance' if `object` is of class sdcProblem-class
  - `sdcStatus`: slot 'sdcStatus' of input `object` if it is of class problemInstance-class or this slot within slot 'problemInstance' if `object` is of class sdcProblem-class
  - `freq`: slot 'freq' of input `object` if it is of class problemInstance-class or this slot within slot 'problemInstance' if `object` is of class sdcProblem-class
  - `strID`: slot 'strID' of input `object` if it is of class problemInstance-class or this slot within slot 'problemInstance' if `object` is of class sdcProblem-class
  - `numVars`: slot 'numVars' of input `object` if it is of class problemInstance-class or this slot within slot 'problemInstance' if `object` is of class sdcProblem-class
  - `w`: slot 'w' of input `object` if it is of class problemInstance-class or this slot within slot 'problemInstance' if `object` is of class sdcProblem-class
- if argument `object` is of class safeObj-class, valid choices are:
  - `finalData`: slot 'finalData' of input `object` of class safeObj-class
  - `nrNonDuplicatedCells`: slot 'nrNonDuplicatedCells' of input `object` of class safeObj-class
  - `nrPrimSupps`: slot 'nrPrimSupps' of input `object` of class safeObj-class
  - `nrSecondSupps`: slot 'nrSecondSupps' of input `object` of class safeObj-class
  - `nrPublishableCells`: slot 'nrPublishableCells' of input `object` of class safeObj-class
  - `suppMethod`: slot 'suppMethod' of input `object` of class safeObj-class

Value

manipulated data dependend on arguments `object` and `type`

Author(s)

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

Examples

```
# load problem (as it was created in the example
# of \code{\link{makeProblem}})
sp <- searchpaths()
fn <- paste(sp[grep("sdcTable", sp)], "/data/problem.RData", sep="")
problem <- get(load(fn))

# problem is an object of class \code{\link{sdcProblem-class}}
print(class(problem))

for (slot in c('lb','ub','LPL','SPL','UPL','sdcStatus',
  'freq', 'strID', 'numVars', 'w')) {
  cat('slot', slot,':\n')
  print(getInfo(problem, type=slot))
}

# extracting information for objects of class \code{\link{safeObj-class}}
fn <- paste(sp[grep("sdcTable", sp)], "/data/protectedData.RData", sep="")
protectedData <- get(load(fn))
for (slot in c('finalData', 'nrNonDuplicatedCells', 'nrPrimSupps',
  'nrSecondSupps', 'nrPublishableCells', 'suppMethod')) {
  cat('slot', slot,':\n')
  print(getInfo(protectedData, type=slot))
}
```

`init.cutList` *initialize* `cutList`*-objects depending on argument* `type`

Description

initialize `cutList`-objects depending on argument `type`

Usage

```
init.cutList(type, input)

## S4 method for signature 'character,list'
init.cutList(type, input)
```

Arguments

`type`	a character vector of length 1 defining what|how to initialize. Allowed types are: • empty: create an empty `cutList`-object • singleCut: create a `cutList`-object with exactly one constraint • multipleCuts: create a `cutList`-object with more than one constraint
`input`	a list depending on argument `type`. • type==empty: input is not used (empty list)

- type==singleCut: input is a list of length 3
  - first element: numeric vector specifying a values for the row of the constraint matrix that must be created
  - second element: character vector of length 1 specifying the direction
  - third element: numeric vector of length 1 specifying the right hand side of the constraint
- type==multipleCuts: input is a list of length 3
  - first element: object of class `matrix`
  - second element: character vector specifying the direction of the constraints
  - third element: numeric vector specifying the right hand side of the constraints

Value

an object of class `cutList`

Note

internal function

Author(s)

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

`init.dataObj` *initialize* `dataObj`*-objects*

Description

initialize `dataObj`-objects

Usage

```
init.dataObj(input)

## S4 method for signature 'list'
init.dataObj(input)
```

Arguments

`input` a list with element described below:

- element 'inputData': a list object holding data
- element 'dimVarInd': index (within `inputData`) of variables that define the table to protect

- element 'freqVarInd': index (within `inputData`) of variable holding frequencies
- element 'numVarInd' index (within `inputData`) of numerical variables (or NULL)
- element 'weightInd': index (within `inputData`) of variable holding weights (or NULL)
- element 'sampWeightInd': index (within `inputData`) of variable holding sampling weights (or NULL)
- element 'isMicroData': logical vector of length 1

Value

an object of class `dataObj`

Note

internal function

Author(s)

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

`init.dimVar` *initialize* `dimVar`-*object*

Description

initialize `dimVar`-object

Usage

```
init.dimVar(input)

## S4 method for signature 'list'
init.dimVar(input)
```

Arguments

`input` a list with 2 elements

- first element: either an object of class 'matrix' or a data.frame or a link to a file. The input data need to be in a specific format (2 columns) with the first column defining the level-structure and the second column defining the level-codes.
- second element: a character vector of length 1 specifying a variable name

Note

internal function

Author(s)

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

`init.simpleTriplet` *initialize* `simpleTriplet`*-objects depending on argument* `type`

Description

init.simpleTriplet should be used to create objects of class `simpleTriplet`. It is possible to create an object from class `simpleTriplet` from an existing matrix (using type=='simpleTriplet'). A positive (or negative) identity matrix stored as an object of class `simpleTriplet` can be created by specifying type=='simpleTripletDiag'.

Usage

```
init.simpleTriplet(type, input)

## S4 method for signature 'character,list'
init.simpleTriplet(type, input)
```

Arguments

`type`	a character vector of length 1 defining whatlhow to initialize. Allowed types are: • simpleTriplet: a simple triplet matrix • simpleTripletDiag: identity matrix
`input`	a list depending on argument `type`. • type == 'simpleTriplet': input is a list of length 1 – first element: object of class 'matrix' • type == 'simpleTripletDiag': input is a list of length 2 – first element: numeric vector of length 1 defining the desired number of rows of the identiy matrix – second element: logical vector of length 1 being TRUE if a positive and FALSE if a negative identity matrix should be returned

Value

an object of class `simpleTriplet`

Note

internal function

Author(s)

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

`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* sdcProblem-class-*objects*

Description

Function makeProblem is used to create sdcProblem-class-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 with each list element being either a data-frame or a link to a .csv-file containing the complete level-hierarchy of a dimensional variable using a top-to-bottom approach. The list names correspond to variable names that must exist in argument data. The level-hierarchy must be specified as follows: • list-element is a data-frame that must contain exactly 2 columns with the first column specifying levels and the second column holding variable-codes. – 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). – second column: a character vector specifying level codes • list-element is full path to a .csv-file with two columns seperated by semicolons (;) having the same structure as the data.frame described above
dimVarInd	numeric vector (or NULL) defining the column-indices of dimensional variables (defining the table) within argument data. If NULL, the names of argument dimList are used to calculate the indices of the dimensional variables within data internally.
freqVarInd	numeric vector (or NULL) defining the column-indices of a variable holding counts within argument data
numVarInd	numeric vector (or NULL) defining the column-indices of additional numeric variables available in argument data
weightInd	numeric vector of length 1 (or NULL) defining the column-index of a variable holding weights that should be used during as objective coefficients during the cut and branch algorithm to protect primary sensitive cells within argument data
sampWeightInd	numeric vector of length 1 (or NULL) defining the column-index of a variable holding sampling weights within argument data

Value

a sdcProblem-class-object

Author(s)

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

Examples

```
# loading micro data
data("microData1", package="sdcTable")
microData <- microData1; rm(microData1)
# having a look at the data structure
str(microData)

# 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 <- create_node(total_lab="Total")
dim.gender <- add_nodes(dim.gender,
  node_labs=c("male","female"), reference_node="Total")
print(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
freqVarInd <- weightInd <- sampWeightInd <- NULL

# creating an object of class \code{\link{sdcProblem-class}}
problem <- makeProblem(
 data=microData,
 dimList=dimList,
 freqVarInd=freqVarInd,
 numVarInd=numVarInd,
 weightInd=weightInd,
```

```
 sampWeightInd=sampWeightInd)

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

# have a look at the data
sdcProb2df(problem, addDups=TRUE,
  addNumVars=TRUE, dimCodes="original")
```

---

manage_hierarchies *Create and modify the structure of hierarchies*

Description

Functions `create_node()`, `add_nodes()` and `delete_nodes()` allow to define and modify hierarchical structures represented as trees. These objects can be used in `makeProblem` to define the (hierarchical) structure of tables.

Usage

```
create_node(total_lab = "Total")

add_nodes(node, node_labs, reference_node)

delete_nodes(node, node_labs, reference_node)
```

Arguments

`total_lab`	the name of the overall total (summation over all contributing)
`node`	a object as created in `create_node()` or returned from `add_nodes()` or `delete_nodes()`.
`node_labs`	character name(s) of new elements that should be inserted to or deleted from a hierarchical structure
`reference_node`	character name of an existing node in the hierarchical structure. When using `add_nodes()`, the new elements are created as children of the reference node. In `delete_nodes()`, all children of the reference node that match the names with argument `node_labs` are deleted from the hierarchy.

Value

a Node object that can be used to specify a hierarchy used to define table inputs in `protectTable`.

Examples

```
dim <- create_node(total_lab="Total")
dim <- add_nodes(dim, reference_node="Total", node_labs=LETTERS[1:4])
print(dim)

## add some levels below "A" and "C"
```

```
dim <- add_nodes(dim, reference_node="A", node_labs=paste0("a", 1:5))
dim <- add_nodes(dim, reference_node="C", node_labs=paste0("c", 1:5))
print(dim)

## delete some specific levels
dim <- delete_nodes(dim, reference_node="A", node_labs=c("a1", "a4"))
print(dim)

## delete entire subtree
dim <- delete_nodes(dim, reference_node="Total", node_labs=c("C"))
print(dim)
# plot(dim)
```

`microData1` *synthetic microdata*

Description

example microdata used for example in `protectLinkedTables`.

Format

A dataframe with 100 observations on 5 variables (`region`,`gender`,`ecoOld`,`ecoNew` and `numVal`)

`microData2` *synthetic microdata*

Description

example microdata used for various examples.

Format

A dataframe with 100 observations on 2 variables (`region` and `gender`)

primarySuppression *perform primary suppression in* sdcProblem-class*-objects*

Description

Function primarySuppression is used to identify and suppress primary sensitive table cells in sdcProblem-class 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-class 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 and numVarInd • p: apply p-percent rule with parameters p and numVarInd • 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 vector of length 1 specifying if empty cells (count==0) should be considered sensitive when using the frequency rule. The default value of allowZeros is 'FALSE' so that empty cells are not considered primary sensitive by default. • 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: numeric vector of length 1 specifying parameter k that is used when applying the nk-dominance rule. Parameter n is set to 85 by default. • numVarInd: numeric vector of length 1 specifying the index of the numerical variable that should be used to identify cells that are dominated by 2 (p-percent rule) or n (nk-dominance)-rule. If type is either 'nk', 'p' or 'pq', it is mandatory to specify numVarInd.

Value

a sdcProblem-class 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
sp <- searchpaths()
fn <- paste(sp[grep("sdcTable", sp)], "/data/microData1.RData", sep="")
microData <- get(load(fn))

# load problem (as it was created in the example in \code{\link{makeProblem}})
fn <- paste(sp[grep("sdcTable", sp)], "/data/problem.RData", sep="")
problem <- get(load(fn))

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

# cell with region=='A' and gender=='female' has 2 units contributing to it
# this cell should be considered sensitive according the the freq-rule with 'maxN' equal to 2!
p1 <- primarySuppression(problem, 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 index of
# a numeric variable.
p2 <- primarySuppression(problem, type='p', p=30, numVarInd=1)

# looking at anonymization states we see, that one cell is primary suppressed (sdcStatus=='u')
# and the remaining cells are possible candidates for secondary 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.

---

problem *data of class* sdcProblem-class

Description

example data of class sdcProblem-class as created in the example of makeProblem

Format

an object of class sdcProblem-class

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>

problemWithSupps *data of class* sdcProblem-class

Description

example data of class sdcProblem-class as created in the example of primarySuppression

Format

an object of class sdcProblem-class featuring primary suppressed table cells

protectedData *data of class safeObj-class*

Description

example data of class safeObj-class as created in the example of protectTable

Format

an object of class safeObj-class being a protected dataset

protectLinkedTables *protect two sdcProblem-class objects that have common cells*

Description

protectLinkedTables 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, ...)
```

Arguments

objectA	a sdcProblem-class object
objectB	a sdcProblem-class object
commonCells	a list object defining common cells in objectA and objectB. 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 tables – first element: character vector of length 1 specifying the variable name in argument objectA – second element: character vector of length 1 specifying the variable name in argument objectB – third element: character vector of length 1 being with keyword ALL • List-elements of length 4: Variable has different codes and levels in tables objectA and objectB – first element: character vector of length 1 specifying the variable name in argument objectA – second element: character vector of length 1 specifying the variable name in argument objectB

- third element: character vector defining codes within objectA
- fourth element: character vector with length that equals the length of the third list-element. The vector defines codes of the variable in objectB that match the codes given in the third list-element for objectA.

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

- HITAS:
- SIMPLEHEURISTIC:
- OPT:

... — additional arguments to control the secondary cell suppression algorithm. For details, see protectTable.

Value

a list of length 2 with each list-element being an safeObj-class object

Author(s)

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

See Also

protectTable

Examples

```
## Not run:
# load micro data for further processing
sp <- searchpaths()
fn <- paste(sp[grep("sdcTable", sp)], "/data/microData2.RData", sep="")
microData <- get(load(fn))

# table1: defined by variables 'gender' and 'ecoOld'
microData1 <- microData[,c(2,3,5)]

# table2: defined by variables 'region', 'gender' and 'ecoNew'
microData2 <- microData[,c(1,2,4,5)]

# we need to create information on the hierarchies
# variable 'region': exists only in microDat2
dim.region <- data.frame(h=c('@','@@','@@'), l=c('Tot', 'R1','R2'))

# variable 'gender': exists in both datasets
dim.gender <- data.frame(h=c('@','@@','@@'), l=c('Tot', 'm','f'))

# variable 'ecoOld': exists only in microDat1
dim.ecoOld <- data.frame(
 h=c('@','@@','@@@','@@@','@@','@@@','@@@'),
 l=c('Tot','A','Aa','Ab','B','Ba','Bb'))
```

```
# variable 'ecoNew': exists only in microDat2
dim.ecoNew <- data.frame(
 h=c('@','@@','@@@','@@@','@@@','@@','@@@','@@@','@@@'),
 l=c('Tot','C','Ca','Cb','Cc','D','Da','Db','Dc'))

# creating objects holding information on dimensions
dimList1 <- list(gender=dim.gender, ecoOld=dim.ecoOld)
dimList2 <- list(region=dim.region, gender=dim.gender, ecoNew=dim.ecoNew)

# creating input objects for further processing. For details have a look at
# \code{\link{makeProblem}}.
problem1 <- makeProblem(data=microData1, dimList=dimList1, dimVarInd=c(1,2),
         numVarInd=3)
problem2 <- makeProblem(data=microData2, dimList=dimList2, dimVarInd=c(1,2,3),
         numVarInd=4)

# the cell specified by gender=='Tot' and ecoOld=='A'
# is one of the common cells! -> we mark it as primary suppression
problem1 <- changeCellStatus(problem1, characteristics=c('Tot', 'A'),
     varNames=c('gender','ecoOld'), 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
problem2 <- changeCellStatus(problem2, characteristics=c('Tot', 'f', 'C'),
 varNames=c('region','gender', 'ecoNew'), rule='u', verbose=FALSE)

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

# variable "gender"
commonCells$v.gender <- list()
commonCells$v.gender[[1]] <- 'gender' # variable name in 'problem1'
commonCells$v.gender[[2]] <- 'gender' # variable name in 'problem2'
# 'gender' has equal characteristics on both datasets -> keyword 'ALL'
commonCells$v.gender[[3]] <- 'ALL'

# variable: ecoOld and ecoNew
commonCells$v.eco <- list()
commonCells$v.eco[[1]] <- 'ecoOld'   # variable name in 'problem1'
commonCells$v.eco[[2]] <- 'ecoNew'   # variable name in 'problem2'

# vector of common characteristics: A and B in variable 'ecoOld' in 'problem1'
commonCells$v.eco[[3]] <- c("A","B")
# correspond to characteristics 'C' and 'D' in variable 'ecoNew' in 'problem2'
commonCells$v.eco[[4]] <- c("C","D")

# protect the linked data
result <- protectLinkedTables(problem1, problem2, commonCells, method='HITAS', verbose=TRUE)

# having a look at the results
result.tab1 <- result[[1]]
result.tab2 <- result[[2]]
summary(result.tab1)
```

```
summary(result.tab2)

## End(Not run)
```

---

protectTable *protecting* sdcProblem-class *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-class 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: heuristic, quick procedure which might be applied to very large problem instances
...	parameters used in the protection algorithm that has been selected. Parameters that can be changed are: • general parameters include: – verbose: logical vector of length 1 defining if verbose output should be produced. Parameter verbose defaults to 'FALSE' – save: logical vector of length 1 defining if temporary results should be saved in the current working directory (TRUE) or not (FALSE). Parameter save defaults to 'FALSE' • parameters used for HITAS|OPT procedures:

  - `solver`: character vector of length 1 defining the solver to be used. Currently available choices are limited to 'glpk'.
  - `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 numeric vector of length 1 (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 `safe` has a default value of 'NULL'
  - `fastSolution`: logical vector of length 1 defining 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`. Parameter `fastSolution` has a default value of 'FALSE'
  - `fixVariables`: logical vector of length 1 defining whether or not it should be tried to fix some variables to zero or one based on reduced costs early in the cut and branch algorithm. Parameter `fixVariables` has a default value of 'TRUE'
  - `approxPerc`: numeric vector of length 1 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`: boolean vector of length 1 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 protection level for the HYPERCUBE-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 used for protectLinkedTables():
  - `maxIter`: numeric vector of length 1 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 `maxIter` is 10

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

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 (http://neon.vb.cbs.nl/casc/tau.htm).

Value

an safeObj-class object

Author(s)

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

Examples

```
# load problem (as it was created after performing primary suppression
# in the example of \code{\link{primarySuppression}})
sp <- searchpaths()
fn <- paste(sp[grep("sdcTable", sp)], "/data/problemWithSupps.RData", sep="")
problem <- get(load(fn))

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

# showing a summary
summary(protectedData)

# looking at the final table with result suppression pattern
print(getInfo(protectedData, type='finalData'))
```

---

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), the algorithm that has been used to protect the data (slot suppMethod) and the time that was needed to protect the data structure (slot elapsedTime) 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

slot elapsedTime**:** numeric vector of length 1 holding the time that was required to protect the table

Note

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

Author(s)

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

---

sdcProb2df *sdcProb2df*

Description

returns a data.frame or data.table from an sdcProblem-class-object which contains the current state of the problem instance.

Usage

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

Arguments

obj	an object of class sdcProblem-class from sdcTable
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 sdc problem instance is returned.

Examples

```
## have a look at ?makeProblem
```

`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 subtables that need to be considered if one wants to protect primary sensitive cells using a heuristic approach (slot `partition`, information on which groups or rather subtables have already been protected while performing a heuristic method (slots `startI` and `startJ`) and the time that has been elapsed (slot `elapsedTime`).

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

slot `elapsedTime`**:** a numeric vector of length 1 holding the time that has already been elapsed during the anonymization process

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>

`set.cutList` *modify* `cutList`*-objects depending on argument* `type`

Description

modify `cutList`-objects depending on argument `type`

Usage

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

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

Arguments

`object`	an object of class `cutList`
`type`	a character vector of length 1 defining what to calculate|return|modify. Allowed types are: • addCompleteConstraint: add a constraint to argument `object` • removeCompleteConstraint: remove a constraint from argument `object`
`input`	a list depending on argument `type`. • type==addCompleteConstraint: a list of length 1 – first element: an object of class `cutList` with exactly one constraint • type==removeCompleteConstraint: a list of length 1 – first element: numeric vector of length 1 specifying the index of the constraint that should be removed

Value

an object of class `cutList`

Note

internal function

Author(s)

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

`set.dataObj` *modify* `dataObj`*-objects depending on argument* `type`

Description

modify `dataObj`-objects depending on argument `type`

Usage

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

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

Arguments

`object`	an object of class `dataObj`
`type`	a character vector of length 1 defining what to calculate|return|modify. Allowed types are: • rawData: set slot 'rawData' of argument `object`
`input`	a list depending on argument `type`. • type==rawData: a list containing raw data

Value

an object of class `dataObj`

Note

internal function

Author(s)

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

`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.linProb` *change* `linProb`*-objects depending on argument* `type`

Description

change `linProb`-objects depending on argument `type`

Usage

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

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

Arguments

`object`	an object of class `linProb`
`type`	a character vector of length 1 defining what to calculate|return|modify. Allowed types are: • objective: change coefficients of the objective • direction: change vector of direction of the constraints • rhs: change vector of right hand side of the constraints • types: change vector of bounds of the objective variables • bounds: change bounds of the objective variables • constraints: change constraint matrix • removeCompleteConstraint: remove a specific constraint from the object • addCompleteConstraint: add a constraint to the object
`input`	a list depending on argument `type`. • type==objective: a list of length 1 – first element: numeric vector defining coefficients of the objective • type==direction: a list of length 1 – first element: character vector defining direction of the constraints • type==rhs: a list of length 1 – first element: numeric vector defining right hand side of the constraints • type==types: a list of length 1 – first element: character vector defining types of objective variables • type==bounds: a list of length 2 – element 'lower': a list with the first element containing indices and the second element containing corrsponding lower bounds – element 'upper': a list with the first element containing indices and the second element containing corrsponding upper bounds • type==constraints: a list of length 1 – first element: an object of class `simpleTriplet`

- type==removeCompleteConstraint: a list of length 1
  - first element: numeric vector of length 1 defining the index of the constraint that should be removed
- type==addCompleteConstraint: a list of length 1
  - first element: an object of class `cutList` defining the constraint that should be added

Value

an object of class `linProb`

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 • elapsedTime: set|modify slot 'elapsedTime' 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', 'startJ' or 'elapsedTime'
- 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 information of* sdcProblem-class- *or* problemInstance-class *objects*

Description

Function getInfo is used to query information from sdcProblem-class- or problemInstance-class objects

Usage

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

Arguments

object	an object of class sdcProblem-class or problemInstance-class
type	a character vector of length 1 specifying the the information that should be changed or modified, valid choices are: • lb: slot 'lb' of input object if it is of class problemInstance-class or slot 'lb' within slot 'problemInstance' if object is of class sdcProblem-class • ub: slot 'ub' of input object if it is of class problemInstance-class or slot 'ub' within slot 'problemInstance' if object is of class sdcProblem-class • LPL: slot 'LPL' of input object if it is of class problemInstance-class or slot 'LPL' within slot 'problemInstance' if object is of class sdcProblem-class • SPL: slot 'SPL' of input object if it is of class problemInstance-class or slot 'SPL' within slot 'problemInstance' if object is of class sdcProblem-class • UPL: slot 'UPL' of input object if it is of class problemInstance-class or slot 'UPL' within slot 'problemInstance' if object is of class sdcProblem-class • sdcStatus: slot 'sdcStatus' of input object if it is of class problemInstance-class or slot 'sdcStatus' within slot 'problemInstance' if object is of class sdcProblem-class

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-class- or problemInstance-class object

Author(s)

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

Examples

```
# load primary suppressed data (created in the example of \code{\link{primarySuppression}})
sp <- searchpaths()
fn <- paste(sp[grep("sdcTable", sp)], "/data/problemWithSupps.RData", sep="")
problem <- get(load(fn))

# which is the overall total?
index.tot <- which.max(getInfo(problem, 'freq'))
index.tot

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

# we want this cell to never be suppressed
problem <- setInfo(problem, type='sdcStatus', index=index.tot, input='z')

# we can verify this:
print(getInfo(problem, type='sdcStatus')[index.tot])

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

---

show,safeObj-method *show* safeObj-class *objects*

Description

extract and show information stored in safeObj-class objects

Usage

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

Arguments

object	an object of class safeObj-class

---

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,safeObj-method

summarize safeObj-class *objects*

Description

extract and show information stored in safeObj-class objects

Usage

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

Arguments

object	an object of class safeObj-class
...	additional arguments, currently ignored

---

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.

Index