

Package ‘sdcTable’

January 2, 2012

Version 0.9.9

Date 2011-11-20

Title Methods for SDC (statistical disclosure control) in tabular data

Description Implementation of methods for SDC (statistical disclosure control) in tabular data

Author Bernhard Meindl

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

Depends methods, Rcpp, Rglpk, lpSolveAPI

LazyLoad yes

LinkingTo Rcpp

License GPL-2

URL <http://www.statistik.at>

Collate

‘classes.r’ ‘data.r’ ‘export_functions.r’ ‘generics_cutList.r’ ‘generics_dataObj.r’ ‘generics_dimInfo.r’ ‘generics_dimVar.r’ ‘g

Repository CRAN

Date/Publication 2011-11-21 10:57:33

R topics documented:

calc.cutList	3
calc.dimVar	4
calc.linProb	5
calc.multiple	6
calc.problemInstance	7
calc.sdcProblem	8
calc.simpleTriplet	11
cellInfo	13
changeCellStatus	14

cutList-class	15
dataObj-class	16
dimInfo-class	17
dimVar-class	18
get.cutList	19
get.dataObj	19
get.dimInfo	20
get.dimVar	21
get.linProb	22
get.problemInstance	23
get.safeObj	25
get.sdcProblem	26
get.simpleTriplet	28
getInfo	29
init.cutList	31
init.dataObj	32
init.dimVar	33
init.simpleTriplet	33
linProb-class	34
makeProblem	35
microData1	37
microData2	38
primarySuppression	38
print	40
problem	40
problemInstance-class	40
problemWithSupps	41
protectedData	41
protectLinkedTables	41
protectTable	44
safeObj-class	47
sdcProblem-class	48
set.cutList	49
set.dataObj	49
set.dimInfo	50
set.linProb	51
set.problemInstance	52
set.sdcProblem	53
setInfo	54
show	55
simpleTriplet-class	55
summary	56

calc.cutList	<i>perform calculations on cutList-objects depending on argument type</i>
--------------	---

Description

perform calculations on cutList-objects depending on argument type

Arguments

object	an object of class cutList
type	a character vector of length 1 defining what to calculate/return/modify. Allowed types are: <ul style="list-style-type: none">• strengthen: strengthen 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. <ul style="list-style-type: none">• type==strengthen: input is not used (empty list)• type==checkViolation: input is a list of length 2<ul style="list-style-type: none">– 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<ul style="list-style-type: none">– 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

Arguments

object	an object of class dimVar
type	<p>a character vector of length 1 defining what to calculate return modify. Allowed types are:</p> <ul style="list-style-type: none"> • 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	<i>perform calculations on linProb-objects depending on argument type</i>
--------------	---

Description

perform calculations on linProb-objects depending on argument type

Arguments

object	an object of class linProb
type	a character vector of length 1 defining what to calculate/return/modify. Allowed types are: <ul style="list-style-type: none"> • 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. <ul style="list-style-type: none"> • type==solveProblem: a list of length 1 <ul style="list-style-type: none"> – first element: character vector of length 1 specifying the solver to use. • type==fixVariables: a list of length 3 <ul style="list-style-type: none"> – 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

Arguments

type	<p>a character vector of length 1 defining what to calculate/return/modify. Allowed types are:</p> <ul style="list-style-type: none"> • 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	<p>a list depending on argument type.</p> <ul style="list-style-type: none"> • if type matches 'makePartitions', 'genMatMFull' or 'makeAttackerProblem': a list of length 2 with elements 'objectA' and 'objectB' <ul style="list-style-type: none"> – element 'object A': an object of class problemInstance – element 'object B': an object of class dimInfo • type matches 'calcFullProblem': a list of length 1 <ul style="list-style-type: none"> – 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

Arguments

object	an object of class problemInstance
type	a character vector of length 1 defining what to calculate return modify. Allowed types are: <ul style="list-style-type: none"> • 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. <ul style="list-style-type: none"> • type==makeMasterProblem: input is not used (empty list) • type==isProtectedSolution: input is a list of length 2 with elements 'input1' and 'input2' <ul style="list-style-type: none"> – 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	<i>perform calculations on sdcProblem-objects depending on argument type</i>
-----------------	--

Description

perform calculations on sdcProblem-objects depending on argument type

Arguments

object	an object of class sdcProblem
type	<p>a character vector of length 1 defining what to calculate/return/modify. Allowed types are:</p> <ul style="list-style-type: none"> • 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 • 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 'cellIID'
- 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

Arguments

- | | |
|--------|--|
| object | an object of class simpleTriplet |
| type | a character vector of length 1 defining what to calculate/return/modify. Allowed types are: <ul style="list-style-type: none"> • 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. <ul style="list-style-type: none"> • type==removeRow: input is a list of length 1 <ul style="list-style-type: none"> – 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	<i>query information for a specific cell in safeObj-class objects</i>
----------	---

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

<code>object</code>	an object of class <code>safeObj-class</code>
<code>characteristics</code>	a character vector specifying characteristics of the table cell that should be identified for each dimensional variable defining the table
<code>varNames</code>	a character vector specifying variable names of dimensional variables defining the tables
<code>verbose</code>	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 used 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

```
## Not run:
# 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')
verbose <- TRUE
info <- cellInfo(protectedData, characteristics, varNames, verbose)

# show the info about this cell
str(info)

## End(Not run)
```

changeCellStatus	<i>change anonymization status of a specific cell</i>
------------------	---

Description

Function `changeCellStatus` allows to changemodify the anonymization state of single table cells for objects of class `sdcProblem-class`.

Usage

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

Arguments

object	an object of class <code>sdcProblem-class</code>
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 used 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

```
## Not run:
# 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'))))

## 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	<i>S4 class describing a dataObj-object</i>
---------------	---

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 slot '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 micro-Data 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	<i>query cutList-objects depending on argument type</i>
-------------	---

Description

query cutList-objects depending on argument type

Arguments

object	an object of class cutList
type	a character vector of length 1 defining what to calculate/return/modify. Allowed types are: <ul style="list-style-type: none">• 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	<i>query dataObj-objects depending on argument type</i>
-------------	---

Description

query dataObj-objects depending on argument type

Arguments

object	an object of class dataObj
type	a character vector of length 1 defining what to calculate/return/modify. Allowed types are: <ul style="list-style-type: none">• 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

Arguments

object	an object of class dataObj
type	a character vector of length 1 defining what to calculate return modify. Allowed types are: <ul style="list-style-type: none"> • strInfo: info on how many digits in the default codes each 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	<i>query dimVar-objects depending on argument type</i>
------------	--

Description

query dimVar-objects depending on argument type

Arguments

object	an object of class dimVar
type	a character vector of length 1 defining what to calculate return modify. Allowed types are: <ul style="list-style-type: none"> • 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

Arguments

object	an object of class linProb
type	a character vector of length 1 defining what to calculate return modify. Allowed types are: <ul style="list-style-type: none">• 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 corresponding lower bounds
 - element 'upper': a list with the first element containing indices and the second element containing corresponding 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

Arguments

object	an object of class <code>problemInstance</code>
type	a character vector of length 1 defining what to calculate/return/modify. Allowed types are: <ul style="list-style-type: none"> • <code>strID</code>: vector of unique IDs for each table cell • <code>nrVars</code>: total number of table cells • <code>freq</code>: vector of frequencies • <code>w</code>: a vector of weights used in the linear problem (or <code>NULL</code>) • <code>numVars</code>: a list containing numeric vectors containing values for numerical variables for each table cell (or <code>NULL</code>) • <code>sdcStatus</code>: a vector containing the suppression state for each cell (possible values are <code>'u'</code>: primary suppression, <code>'x'</code>: secondary suppression, <code>'z'</code>: forced for publication, <code>'s'</code>: publishable cell) • <code>lb</code>: lower bound assumed to be known by attackers for each table cell • <code>ub</code>: upper bound assumed to be known by attackers for each table cell • <code>LPL</code>: lower protection level required to protect table cells • <code>UPL</code>: upper protection level required to protect table cells • <code>SPL</code>: sliding protection level required to protect table cells • <code>primSupps</code>: vector of indices of primary sensitive cells • <code>secondSupps</code>: vector of indices of secondary suppressed cells • <code>forcedCells</code>: vector of indices of cells that must not be suppressed • <code>hasPrimSupps</code>: shows if object has primary suppressions or not • <code>hasSecondSupps</code>: shows if object has secondary suppressions or not • <code>hasForcedCells</code>: shows if object has cells that must not be suppressed • <code>weight</code>: gives weight that is used the suppression procedures • <code>suppPattern</code>: 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

Arguments

object	an object of class safeObj
type	<p>a character vector of length 1 defining what to calculate return modify. Allowed types are:</p> <ul style="list-style-type: none"> • 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	<p>a list depending on argument type.</p> <ul style="list-style-type: none"> • 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 <ul style="list-style-type: none"> – 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

Arguments

- | | |
|--------|---|
| object | an object of class sdcProblem |
| type | a character vector of length 1 defining what to calculate/return/modify. Allowed types are: <ul style="list-style-type: none"> • 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 HITASIHYPERCUBE)
- startJ: current number of the subtable within a given level that needs to be protected (useful when restarting HITASIHYPERCUBE)
- 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

Arguments

object	an object of class simpleTriplet
type	a character vector of length 1 defining what to calculate/return/modify. Allowed types are: <ul style="list-style-type: none"> • 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. <ul style="list-style-type: none"> • type == 'getRow': input is a list of length 1 <ul style="list-style-type: none"> – 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 <ul style="list-style-type: none"> – 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	<i>query information from objects</i>
---------	---------------------------------------

Description

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

Usage

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

Arguments

- | | |
|--------|---|
| object | a <code>sdProblem-class</code> object, <code>problemInstance-class</code> object or <code>safeObj-class</code> object. |
| type | a character vector of length 1 specifying the information which should be returned. <ul style="list-style-type: none"> • if argument object is of class <code>sdProblem-class</code> or <code>problemInstance-class</code>, valid choices are: <ul style="list-style-type: none"> – lb: slot 'lb' of input object if it is of class <code>problemInstance-class</code> or this slot within slot 'problemInstance' if object is of class <code>sdProblem-class</code> – ub: slot 'ub' of input object if it is of class <code>problemInstance-class</code> or this slot within slot 'problemInstance' if object is of class <code>sdProblem-class</code> – LPL: slot 'LPL' of input object if it is of class <code>problemInstance-class</code> or this slot within slot 'problemInstance' if object is of class <code>sdProblem-class</code> – SPL: slot 'SPL' of input object if it is of class <code>problemInstance-class</code> or this slot within slot 'problemInstance' if object is of class <code>sdProblem-class</code> – UPL: slot 'UPL' of input object if it is of class <code>problemInstance-class</code> or this slot within slot 'problemInstance' if object is of class <code>sdProblem-class</code> – sdcStatus: slot 'sdcStatus' of input object if it is of class <code>problemInstance-class</code> or this slot within slot 'problemInstance' if object is of class <code>sdProblem-class</code> – freq: slot 'freq' of input object if it is of class <code>problemInstance-class</code> or this slot within slot 'problemInstance' if object is of class <code>sdProblem-class</code> |

- strID: slot 'strID' of input object if it is of class `problemInstance-class` or this slot within slot 'problemInstance' if object is of class `sdProblem-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 `sdProblem-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 `sdProblem-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 dependent on arguments object and type

Author(s)

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

Examples

```
## Not run:
# load problem (as it was created in the example
# of \code{\link{makeProblem}})
sp <- searchpaths()
fn <- paste(sp[grep("sdTable", sp)], "/data/problem.RData", sep="")
problem <- get(load(fn))

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

for ( slot in c('lb', 'ub', 'LPL', 'SPL', 'UPL', 'sdStatus',
'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))
}

## End(Not run)

```

init.cutList

initialize cutList-objects depending on argument type

Description

initialize cutList-objects depending on argument type

Arguments

- | | |
|-------|---|
| type | <p>a character vector of length 1 defining what/how to initialize. Allowed types are:</p> <ul style="list-style-type: none"> • 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 | <p>a list depending on argument type.</p> <ul style="list-style-type: none"> • type==empty: input is not used (empty list) • type==singleCut: input is a list of length 3 <ul style="list-style-type: none"> – 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 <ul style="list-style-type: none"> – first element: object of class <code>matrix</code> – 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

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	<i>initialize dimVar-object</i>
-------------	---------------------------------

Description

initialize dimVar-object

Arguments

input	a list with 2 elements <ul style="list-style-type: none"> • 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	<i>initialize simpleTriplet-objects depending on argument type</i>
--------------------	--

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'.

Arguments

type	a character vector of length 1 defining what/how to initialize. Allowed types are: <ul style="list-style-type: none"> • simpleTriplet: a simple triplet matrix • simpleTripletDiag: identity matrix
input	a list depending on argument type. <ul style="list-style-type: none"> • type == 'simpleTriplet': input is a list of length 1 <ul style="list-style-type: none"> – 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 identity 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	<i>create <code>sdcProblem-class</code>-objects</i>
-------------	---

Description

Function `makeProblem` is used to create `sdcProblem-class`-objects.

Usage

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

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: <ul style="list-style-type: none"> 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. <ul style="list-style-type: none"> 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 separated by semi-colons (;) 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
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
isMicroData	logical vector of length 1 that is 'TRUE' if argument data are micro data and 'FALSE' otherwise

Value

a `sdProblem-class`-object

Author(s)

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

Examples

```
## Not run:
# loading micro data
sp <- searchpaths()
fn <- paste(sp[grep("sdTable", sp)], "/data/microData1.RData", sep="")
microData <- get(load(fn))

having a look at the data structure
str(microData)

# 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'
# levels 'male' and 'female' sum up to a Total
dim.gender <- data.frame(
  levels=c('@', '@@', '@@'),
  codes=c('Total', 'male', 'female'),
  stringsAsFactors=FALSE)

# create a list with each element being a data-frame containing information
# on a dimensional variables
dimList <- list(dim.region, dim.gender)

# name the list:
# - first list-element: corresponds to variable 'region'
# - second list-element: corresponds to variable 'gender'
names(dimList) <- c('region', 'gender')
```

```
# specify the indices where dimensional variables are located
# within the input data

# - variable 'region': first column
# - variable 'gender': second column
dimVarInd <- c(1,2)

# no variables holding counts, numeric values, weights or sampling
# weights are available in the input data
freqVarInd <- numVarInd <- weightInd <- sampWeightInd <- NULL

# but we are dealing with micro data!
isMicroData <- TRUE

# creating an object of class \link{sdProblem-class}
problem <- makeProblem(
  data=microData,
  dimList=dimList,
  dimVarInd=dimVarInd,
  freqVarInd=freqVarInd,
  numVarInd=numVarInd,
  weightInd=weightInd,
  sampWeightInd=sampWeightInd,
  isMicroData=isMicroData)

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

## End(Not run)
```

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	<i>synthetic microdata</i>
------------	----------------------------

Description

example microdata used for various examples.

Format

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

primarySuppression	<i>perform primary suppression in <code>sdcProblem-class</code>-objects</i>
--------------------	---

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

<code>object</code>	a <code>sdcProblem-class</code> object
<code>type</code>	character vector of length 1 defining the primary suppression rule. Allowed types are: <ul style="list-style-type: none"> • <code>freq</code>: apply frequency rule with parameters <code>maxN</code> and <code>allowZeros</code> • <code>nk</code>: apply nk-dominance rule with parameters <code>n</code>, <code>k</code> and <code>numVarInd</code> • <code>p</code>: apply p-percent rule with parameters <code>p</code> and <code>numVarInd</code>
<code>...</code>	parameters used in the identification of primary sensitive cells. Parameters that can be modified/changed are: <ul style="list-style-type: none"> • <code>maxN</code>: numeric vector of length 1 used when applying the frequency rule. All cells having counts \leq <code>maxN</code> are set as primary suppressed. The default value of <code>maxN</code> is 3. • <code>allowZeros</code>: logical vector of length 1 specifying if empty cells (<code>count==0</code>) should be considered sensitive when using the frequency rule. The default value of <code>allowZeros</code> is 'FALSE' so that empty cells are not considered primary sensitive by default. • <code>p</code>: numeric vector of length 1 specifying parameter <code>p</code> that is used when applying the p-percent rule with default value of 80.

- `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'` or `'p'` it is mandatory to specify `p`.

Value

a `sdcProblem-class` object

Note

the `nk`-dominance rule and the `p`-percent 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

```
## Not run:
# 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 table
print(table(microData))

# cell with region=='A' and gender=='female' has 2 units contributing to it
# this cell should be considered sensitive!
problem <- primarySuppression(problem, type='freq', maxN=3)

# looking at anonymization states
print(table(getInfo(problem, type='sdcStatus'))))

# we see that exactly one cell is primary suppressed (sdcStatus=='u') and
# the remaining cells are possible candidates for secondary suppression ('s')

## End(Not run)
```

print *print dimVar-class objects*

Description

print [dimVar-class](#) objects in a reasonable way

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	<p>a list object defining common cells in codeobjectA and objectB. For each variable that has one or more common codes in both tables, a list element needs to be specified.</p> <ul style="list-style-type: none"> • List-elements of length 3: Variable has exact same levels and structure in both tables <ul style="list-style-type: none"> – 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 <ul style="list-style-type: none"> – 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	<p>a character vector of length 1 specifying the algorithm that should be used to protect the primary sensitive table cells. Allowed values are:</p> <ul style="list-style-type: none"> • HITAS: • HYPERCUBE: • OPT:
...	<p>additional arguments to control the secondary cell suppression algorithm. For details, see protectTable.</p>

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, isMicroData=TRUE)
problem2 <- makeProblem(data=microData2, dimList=dimList2, dimVarInd=c(1,2,3),
  numVarInd=4, isMicroData=TRUE)

# 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 <code>sdcProblem-class</code> object that has created using <code>makeProblem</code> and has been modified by <code>primarySuppression</code>
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.

... 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'
- 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 specifying 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 iterations that should be make while trying to protect common cells of two different tables. The default value of parameter `maxIter` is 10

Value

an `safeObj-class` object

Author(s)

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

Examples

```
## Not run:
# load problem (as it was created after performing primary suppression
# in the example of \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)

# showing a summary
summary(protectedData)

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

## End(Not run)
```

`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>

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	<i>modify cutList-objects depending on argument type</i>
-------------	--

Description

modify cutList-objects depending on argument type

Arguments

object	an object of class cutList
type	a character vector of length 1 defining what to calculate/return/modify. Allowed types are: <ul style="list-style-type: none"> • addCompleteConstraint: add a constraint to argument object • removeCompleteConstraint: remove a constraint from argument object
input	a list depending on argument type. <ul style="list-style-type: none"> • type==addCompleteConstraint: a list of length 1 <ul style="list-style-type: none"> – first element: an object of class cutList with exactly one constraint • type==removeCompleteConstraint: a list of length 1 <ul style="list-style-type: none"> – 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	<i>modify dataObj-objects depending on argument type</i>
-------------	--

Description

modify dataObj-objects depending on argument type

Arguments

object	an object of class dataObj
type	a character vector of length 1 defining what to calculate return modify. Allowed types are: <ul style="list-style-type: none"> • rawData: set slot 'rawData' of argument object
input	a list depending on argument type. <ul style="list-style-type: none"> • 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	<i>modify dimInfo-objects depending on argument type</i>
-------------	--

Description

modify dimInfo-objects depending on argument type

Arguments

object	an object of class dimInfo
type	a character vector of length 1 defining what to calculate return modify. Allowed types are: <ul style="list-style-type: none"> • strID: set slot 'strID' of argument object
input	a list depending on argument type. <ul style="list-style-type: none"> • 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	<i>change linProb-objects depending on argument type</i>
-------------	--

Description

change linProb-objects depending on argument type

Arguments

object	an object of class linProb
type	<p>a character vector of length 1 defining what to calculate/return/modify. Allowed types are:</p> <ul style="list-style-type: none"> • 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	<p>a list depending on argument type.</p> <ul style="list-style-type: none"> • type==objective: a list of length 1 <ul style="list-style-type: none"> – first element: numeric vector defining coefficients of the objective • type==direction: a list of length 1 <ul style="list-style-type: none"> – first element: character vector defining direction of the constraints • type==rhs: a list of length 1 <ul style="list-style-type: none"> – first element: numeric vector defining right hand side of the constraints • type==types: a list of length 1 <ul style="list-style-type: none"> – first element: character vector defining types of objective variables • type==bounds: a list of length 2 <ul style="list-style-type: none"> – element 'lower': a list with the first element containing indices and the second element containing corresponding lower bounds – element 'upper': a list with the first element containing indices and the second element containing corresponding upper bounds • type==constraints: a list of length 1 <ul style="list-style-type: none"> – first element: an object of class simpleTriplet • type==removeCompleteConstraint: a list of length 1 <ul style="list-style-type: none"> – first element: numeric vector of length 1 defining the index of the constraint that should be removed • type==addCompleteConstraint: a list of length 1 <ul style="list-style-type: none"> – 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

Arguments

object	an object of class problemInstance
type	a character vector of length 1 defining what to calculate return modify. Allowed types are: <ul style="list-style-type: none"> • 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'. <ul style="list-style-type: none"> • 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	<i>modify sdcProblem-objects depending on argument type</i>
----------------	---

Description

modify sdcProblem-objects depending on argument type

Arguments

object	an object of class sdcProblem
type	a character vector of length 1 defining what to calculate/return/modify. Allowed types are: <ul style="list-style-type: none"> • 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. <ul style="list-style-type: none"> • 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	<i>set information of <code>sdcProblem-class</code>- or <code>problemInstance-class</code> objects</i>
---------	--

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 <code>sdcProblem-class</code> or <code>problemInstance-class</code>
type	<p>a character vector of length 1 specifying the the information that should be changed or modified, valid choices are:</p> <ul style="list-style-type: none"> • lb: slot 'lb' of input object if it is of class <code>problemInstance-class</code> or slot 'lb' within slot 'problemInstance' if object is of class <code>sdcProblem-class</code> • ub: slot 'ub' of input object if it is of class <code>problemInstance-class</code> or slot 'ub' within slot 'problemInstance' if object is of class <code>sdcProblem-class</code> • LPL: slot 'LPL' of input object if it is of class <code>problemInstance-class</code> or slot 'LPL' within slot 'problemInstance' if object is of class <code>sdcProblem-class</code> • SPL: slot 'SPL' of input object if it is of class <code>problemInstance-class</code> or slot 'SPL' within slot 'problemInstance' if object is of class <code>sdcProblem-class</code> • UPL: slot 'UPL' of input object if it is of class <code>problemInstance-class</code> or slot 'UPL' within slot 'problemInstance' if object is of class <code>sdcProblem-class</code> • sdcStatus: slot 'sdcStatus' of input object if it is of class <code>problemInstance-class</code> or slot 'sdcStatus' within slot 'problemInstance' if object is of class <code>sdcProblem-class</code>
index	numeric vector defining cell-indices for which which values in a specified slot should be changed/modified
input	<p>numeric or character vector depending on argument type with its length matching the length of argument index</p> <ul style="list-style-type: none"> • 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

```
## Not run:
# 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))

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

# the anonymization state of the total is
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')

print(getInfo(problem, type='sdcStatus')[index.tot])

## End(Not run)
```

show *show `safeObj-class` objects*

Description

extract and show information stored in `safeObj-class` objects

`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 `con` in class `cutList-class`

Author(s)

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

summary

summarize `safeObj-class` objects

Description

extract and show information stored in `safeObj-class` objects

Index

*Topic **datasets**
microData1, 37
microData2, 38
problem, 40
problemWithSupps, 41
protectedData, 41

calc.cutList, 3
calc.cutList, cutList, character, list-method
 (calc.cutList), 3
calc.dimVar, 4
calc.dimVar, dimVar, character, character-method
 (calc.dimVar), 4
calc.linProb, 5
calc.linProb, linProb, character, list-method
 (calc.linProb), 5
calc.multiple, 6
calc.multiple, character, list-method
 (calc.multiple), 6
calc.problemInstance, 7
calc.problemInstance, problemInstance, character, list-method
 (calc.problemInstance), 7
calc.sdcProblem, 8
calc.sdcProblem, sdcProblem, character, list-method
 (calc.sdcProblem), 8
calc.simpleTriplet, 11
calc.simpleTriplet, simpleTriplet, character, list-method
 (calc.simpleTriplet), 11

cellInfo, 13, 13
changeCellStatus, 14, 14
cutList-class, 56
cutList-class, 15

dataObj-class, 16
dimInfo-class, 47
dimInfo-class, 17
dimVar-class, 40
dimVar-class, 18, 40

get.cutList, 19
get.cutList, cutList, character-method
 (get.cutList), 19
get.dataObj, 19
get.dataObj, dataObj, character-method
 (get.dataObj), 19
get.dimInfo, 20
get.dimInfo, dimInfo, character-method
 (get.dimInfo), 20
get.dimVar, 21
get.dimVar, dimVar, character-method
 (get.dimVar), 21
get.linProb, 22
get.linProb, linProb, character-method
 (get.linProb), 22
get.problemInstance, 23
get.problemInstance, problemInstance, character-method
 (get.problemInstance), 23
get.safeObj, 25
get.safeObj, safeObj, character, list-method
 (get.safeObj), 25
get.sdcProblem, 26
get.sdcProblem, sdcProblem, character-method
 (get.sdcProblem), 26
get.simpleTriplet, 28
get.simpleTriplet, simpleTriplet, character, list-method
 (get.simpleTriplet), 28
get.info, 29, 29, 54

init.cutList, 31
init.cutList, character, list-method
 (init.cutList), 31
init.dataObj, 32
init.dataObj, list-method
 (init.dataObj), 32
init.dimVar, 33
init.dimVar, list-method (init.dimVar),
 33
init.simpleTriplet, 33
init.simpleTriplet, character, list-method
 (init.simpleTriplet), 33

linProb-class, [15](#), [56](#)
linProb-class, [34](#)

makeProblem, [35](#), [35](#), [39](#), [40](#), [44](#), [48](#)
microData1, [37](#)
microData2, [38](#)

primarySuppression, [38](#), [38](#), [41](#), [44](#), [48](#)
print, [40](#)
print, dimVar-method (print), [40](#)
problem, [40](#)
problemInstance-class, [29](#), [30](#), [54](#), [55](#)
problemInstance-class, [40](#), [54](#)
problemWithSupps, [41](#)
protectedData, [41](#)
protectLinkedTables, [37](#), [41](#), [41](#)
protectTable, [41](#), [42](#), [44](#), [44](#), [47](#), [48](#)

safeObj-class, [13](#), [29](#), [30](#), [41](#), [42](#), [46](#), [55](#), [56](#)
safeObj-class, [13](#), [41](#), [47](#), [55](#), [56](#)
sdcProblem-class, [14](#), [29](#), [30](#), [35](#), [36](#), [38–42](#),
[44](#), [54](#), [55](#)
sdcProblem-class, [35](#), [38](#), [40](#), [41](#), [44](#), [48](#), [54](#)
set.cutList, [49](#)
set.cutList, cutList, character, list-method
(set.cutList), [49](#)
set.dataObj, [49](#)
set.dataObj, dataObj, character, listOrNULL-method
(set.dataObj), [49](#)
set.dimInfo, [50](#)
set.dimInfo, dimInfo, character, character-method
(set.dimInfo), [50](#)
set.linProb, [51](#)
set.linProb, linProb, character, list-method
(set.linProb), [51](#)
set.problemInstance, [52](#)
set.problemInstance, problemInstance, character, list-method
(set.problemInstance), [52](#)
set.sdcProblem, [53](#)
set.sdcProblem, sdcProblem, character, list-method
(set.sdcProblem), [53](#)
setInfo, [54](#)
show, [55](#)
show, safeObj-method (show), [55](#)
simpleTriplet-class, [15](#), [34](#)
simpleTriplet-class, [55](#)
summary, [56](#)
summary, safeObj-method (summary), [56](#)