Package 'rorutadis'

January 17, 2017

Type Package

Title Robust Ordinal Regression UTADIS

Version 0.4.2

Date 2017-01-18

Author Krzysztof Ciomek

Maintainer Krzysztof Ciomek <k.ciomek@gmail.com>

URL https://github.com/kciomek/rorutadis

Depends Rglpk (>= 0.5-1), ggplot2 (>= 0.9.3.1), gridExtra (>= 0.9.1), hitandrun (>= 0.5-2)

Description Implementation of Robust Ordinal Regression for multiple criteria value-based sorting with preference information provided in form of possibly imprecise assignment examples, assignment-based pairwise comparisons, and desired class cardinalities [Kadzinski et al. 2015, <doi:10.1016/j.ejor.2014.09.050>].

License GPL-3

Suggests testthat (>= 0.7.1)

NeedsCompilation no

Repository CRAN

Date/Publication 2017-01-17 23:07:43

R topics documented:

`rorutadis-package` *Robust Ordinal Regression UTADIS*

Description

Implementation of Robust Ordinal Regression for multiple criteria value-based sorting with some extensions and additional tools.

Details

Package:	rorutadis
Type:	Package
Version:	0.4.2
Date:	2017-01-18
License:	GPL-3

Author(s)

Krzysztof Ciomek

Maintainer: Krzysztof Ciomek <k.ciomek at gmail.com>

`addAssignmentPairwiseAtLeastComparisons`
Add assignment pairwise at least *comparisons*

Description

The comparison of a pair of alternatives may indicate that *a_i* should be assigned to a class at least as good as class of *a_j* or at least better by *k* classes. The function `assignmentPairwiseAtLeastComparisons` allows to define such pairwise comparisons.

Usage

```
addAssignmentPairwiseAtLeastComparisons(problem, ...)
```

Arguments

`problem`	Problem to which preference information will be added.
`...`	Comparisons as three-element vectors. Each vector `c(i, j, k)` represents a single assignment comparison: alternative *a_i* has to be assigned to class at least better by *k* classes then class of *a_j*.

Value

Problem with added comparisons.

See Also

`buildProblem` `removeAssignmentPairwiseAtLeastComparisons`

Examples

```
# 4 alternatives, 2 gain criteria, 3 classes, monotonously increasing
# and general marginal value functions
perf <- matrix(c(5, 2, 1, 7, 0.5, 0.9, 0.4, 0.4), ncol = 2)
problem <- buildProblem(perf, 3, FALSE, c('g', 'g'), c(0, 0))

# add comparisons:
# alternative 2 to class at least as good as class of alternative 1
# alternative 4 to class at least better by 1 class then class
# of alternative 3
problem <- addAssignmentPairwiseAtLeastComparisons(problem,
   c(4, 3, 1), c(2, 1, 0))
```

addAssignmentPairwiseAtMostComparisons

Add assignment pairwise at most *comparisons*

Description

The comparison of a pair of alternatives may indicate that alternative a_i should be assigned to a class at most better by k classes then class of a_j. The function `assignmentPairwiseAtMostComparisons` allows to define such pairwise comparisons.

Usage

```
addAssignmentPairwiseAtMostComparisons(problem, ...)
```

Arguments

`problem`	Problem to which preference information will be added.
`...`	Comparisons as three-element vectors. Each vector `c(i, j, k)` represents a single assignment comparison: alternative a_i has to be assigned to class at most better by k classes then class of a_j.

Value

Problem with added comparisons.

See Also

`buildProblem` `removeAssignmentPairwiseAtMostComparisons`

Examples

```
# 4 alternatives, 2 gain criteria, 3 classes, monotonously increasing
# and general marginal value functions
perf <- matrix(c(5, 2, 1, 7, 0.5, 0.9, 0.4, 0.4), ncol = 2)
problem <- buildProblem(perf, 3, FALSE, c('g', 'g'), c(0, 0))

# add comparison:
# alternative 4 to class at most better by 1 class then class
# of alternative 3
problem <- addAssignmentPairwiseAtMostComparisons(problem, c(4, 3, 1))
```

addAssignmentsLB — *Add lower bound of alternative possible assignments*

Description

This function adds lower bounds of possible assignments to a problem.

Usage

```
addAssignmentsLB(problem, ...)
```

Arguments

`problem`	Problem to which preference information will be added.
`...`	Assignments as two-element vectors. Each vector `c(i, j)` represents assignment of an alternative *a_i* to class at least as good as class *C_j*.

Value

Problem with added assignment examples.

See Also

`buildProblem` `removeAssignmentsLB`

Examples

```
# 4 alternatives, 2 gain criteria, 3 classes, monotonously increasing
# and general marginal value functions
perf <- matrix(c(5, 2, 1, 7, 0.5, 0.9, 0.4, 0.4), ncol = 2)
problem <- buildProblem(perf, 3, FALSE, c('g', 'g'), c(0, 0))

# add assignment examples: alternative 1 to class at least as good as class 2
# and alternative 2 to class at least as good as class 3
problem <- addAssignmentsLB(problem, c(1, 2), c(2, 3))
```

addAssignmentsUB — *Add upper bound of alternative possible assignments*

Description

This function adds upper bounds of possible assignments to a problem.

Usage

```
addAssignmentsUB(problem, ...)
```

Arguments

problem Problem to which preference information will be added.

... Assignments as two-element vectors. Each vector c(i, j) represents assignment of an alternative *a_i* to at most class as good as *C_j*.

Value

Problem with added assignment examples.

See Also

buildProblem removeAssignmentsUB

Examples

```
# 4 alternatives, 2 gain criteria, 3 classes, monotonously increasing
# and general marginal value functions
perf <- matrix(c(5, 2, 1, 7, 0.5, 0.9, 0.4, 0.4), ncol = 2)
problem <- buildProblem(perf, 3, FALSE, c('g', 'g'), c(0, 0))

# add assignment examples: alternative 3 at most to class as good as class 1
# and alternative 4 to class at most as good as class 2
problem <- addAssignmentsUB(problem, c(3, 1), c(4, 2))
```

---

addMaximalClassCardinalities

Add maximal class cardinality restrictions

Description

This function allows to define maximal cardinality of particular classes.

Usage

```
addMaximalClassCardinalities(problem, ...)
```

Arguments

problem Problem to which preference information will be added.

... Minimal cardinalities as two-element vectors c(i, j), where *j* is a maximal cardinality of class *C_i*.

Value

Problem with added preference information.

See Also

buildProblem removeMaximalClassCardinalities

Examples

```
# 4 alternatives, 2 gain criteria, 3 classes, monotonously increasing
# and general marginal value functions
perf <- matrix(c(5, 2, 1, 7, 0.5, 0.9, 0.4, 0.4), ncol = 2)
problem <- buildProblem(perf, 3, FALSE, c('g', 'g'), c(0, 0))

# set maximal class cardinalities:
# at most two alternatives could be assigned to class 2
# and at most one alternative could be assigned to class 3
problem <- addMaximalClassCardinalities(problem, c(2, 2), c(3, 1))
```

---

addMinimalClassCardinalities

Add minimal class cardinality restrictions

Description

This function allows to define minimal cardinality of particular classes.

Usage

```
addMinimalClassCardinalities(problem, ...)
```

Arguments

`problem`	Problem to which preference information will be added.
`...`	Minimal cardinalities as two-element vectors `c(i, j)`, where *j* is a minimal cardinality of class *C_i*.

Value

Problem with added preference information.

See Also

buildProblem removeMinimalClassCardinalities

Examples

```
# 4 alternatives, 2 gain criteria, 3 classes, monotonously increasing
# and general marginal value functions
perf <- matrix(c(5, 2, 1, 7, 0.5, 0.9, 0.4, 0.4), ncol = 2)
problem <- buildProblem(perf, 3, FALSE, c('g', 'g'), c(0, 0))

# set minimal class cardinalities:
# at least one alternative has to be assigned to class 2
# and at least one alternative has to be assigned to class 3
problem <- addMinimalClassCardinalities(problem, c(2, 1), c(3, 1))
```

---

buildProblem — *Build a representation of a problem*

Description

This function creates representation of a given problem for usage in farther computations.

Usage

```
buildProblem(perf, nrClasses, strictVF, criteria, characteristicPoints)
```

Arguments

`perf`	A *n* x *m* performance matrix of *n* alternatives evaluated on *m* criteria.
`nrClasses`	Number of classes.
`strictVF`	`TRUE` for strictly monotonic marginal value functions, `FALSE` for weakly monotonic.
`criteria`	A vector containing type of each criterion (`'g'` - gain, `'c'` - cost).
`characteristicPoints`	A vector of integers that for each criterion contains number of characteristic points or *0* for general marginal value function.

Value

Representation of a problem as a list with named members.

See Also

`addAssignmentsLB` `removeAssignmentsLB` `addAssignmentsUB` `removeAssignmentsUB` `addAssignmentPairwiseAtLeast`
`removeAssignmentPairwiseAtLeastComparisons` `addAssignmentPairwiseAtMostComparisons`
`removeAssignmentPairwiseAtMostComparisons` `addMinimalClassCardinalities` `removeMinimalClassCardinaliti`
`addMaximalClassCardinalities` `removeMaximalClassCardinalities`

Examples

```
# 4 alternatives, 2 gain criteria, 3 classes, monotonously increasing
# and general marginal value functions
perf <- matrix(c(5, 2, 1, 7, 0.5, 0.9, 0.4, 0.4), ncol = 2)
problem <- buildProblem(perf, 3, FALSE, c('g', 'g'), c(0, 0))
```

`calculateAssignments` *Calculate assignments*

Description

This function calculates possible and necessary assignments.

Usage

```
calculateAssignments(problem, necessary)
```

Arguments

`problem`	Problem for which assignments will be calculated.
`necessary`	Whether necessary or possible assignments.

Value

n x p logical matrix, where each row represents one of n alternatives and each column represents one of p classes. Element `[i, h]` is `TRUE` if:

- for necessary assignments: alternative `a_i` is always assigned to class `C_h`,
- for possible assignments: alternative `a_i` can be assigned to class `C_h`.

Examples

```
perf <- matrix(c(5, 2, 1, 7, 0.5, 0.9, 0.4, 0.4), ncol = 2)
problem <- buildProblem(perf, 3, FALSE, c('g', 'g'), c(0, 0))
problem <- addAssignmentsLB(problem, c(1, 2), c(2, 3))

possibleAssignments <- calculateAssignments(problem, FALSE)
necessaryAssignments <- calculateAssignments(problem, TRUE)
```

calculateExtremeClassCardinalities

Calculate extreme class cardinalities

Description

This function calculates minimal and maximal possible cardinality of each class.

Usage

```
calculateExtremeClassCardinalities(problem)
```

Arguments

`problem`	Problem for which extreme class cardinalities will be calculated.

Value

p x *2* matrix, where *p* is the number of classes. Value at `[h, 1]` is a minimal possible cardinality of class `C_h`, and value at `[h, 2]` is a maximal possible cardinality of class `C_h`.

See Also

`addMinimalClassCardinalities` `addMaximalClassCardinalities`

Examples

```
perf <- matrix(c(5, 2, 1, 7, 0.5, 0.9, 0.4, 0.4), ncol = 2)
problem <- buildProblem(perf, 3, FALSE, c('g', 'g'), c(0, 0))
problem <- addAssignmentsLB(problem, c(1, 2), c(2, 3))

extremeClassCardinalities <- calculateExtremeClassCardinalities(problem)
```

calculateStochasticResults

Stochastic results

Description

The function calculates stochastic results for alternative assignments, assignment-based preference relation and class cardinalities. The results are computed by sampling the space of compatible models.

Usage

```
calculateStochasticResults(problem, nrSamples = 100)
```

Arguments

`problem`	A problem to consider.
`nrSamples`	Number of samples. Use more for better quality of results.

Value

List with the following named elements:

- *assignments* - *n* x *p* matrix, where *n* is the number of alternatives and *p* is number of classes; each element `[i, j]` contains the rate of samples, for which alternative *a_i* was assigned to class *C_j*. The exact result can be calculated with function calculateAssignments.
- *preferenceRelation* - *n* x *n* matrix, where *n* is the number of alternatives; each element `[i, j]` contains the rate of samples, for which alternative *a_i* was assigned to class at least as good as class of *a_j*. The exact result can be calculated with function compareAssignments.
- *classCardinalities* - *p* x *(n + 1)* matrix, where *n* is the number of alternatives and *p* is number of classes; each element `[i, j]` contains the rate of samples, for which *j-1* alternatives were assigned to class *C_i*. **Note!** first column corresponds to **0** elements. The exact result can be calculated with function calculateExtremeClassCardinalities.

See Also

`buildProblem` `calculateAssignments` `compareAssignments` `calculateExtremeClassCardinalities`

Examples

```
perf <- matrix(c(2,1,1,2), 2)
problem <- buildProblem(perf, 2, FALSE, c('g', 'g'), c(0, 0))

calculateStochasticResults(problem, 1000)
```

`checkConsistency` *Check problem consistency*

Description

This function allows to check if preference information is consistent.

Usage

```
checkConsistency(problem)
```

Arguments

`problem`	Problem to check.

Value

`TRUE` if a model of a problem is feasible and `FALSE` otherwise.

Examples

```
perf <- matrix(c(5, 2, 1, 7, 0.5, 0.9, 0.4, 0.4), ncol = 2)
problem <- buildProblem(perf, 3, FALSE, c('g', 'g'), c(0, 0))
problem <- addAssignmentsLB(problem, c(1, 2), c(2, 3))

isConsistent <- checkConsistency(problem)
```

`compareAssignments` *Compare assignments*

Description

This function compares assignments.

Usage

```
compareAssignments(problem, necessary = TRUE)
```

Arguments

`problem`	Problem for which assignments will be compared.
`necessary`	Whether necessary or possible assignments.

Value

n x n logical matrix, where `n` is a number of alternatives. Cell `[i, j]` is `TRUE` if *a_i* is assigned to class at least as good as class of *a_j* for all compatible value functions.

Examples

```
perf <- matrix(c(5, 2, 1, 7, 0.5, 0.9, 0.4, 0.4), ncol = 2)
problem <- buildProblem(perf, 3, FALSE, c('g', 'g'), c(0, 0))
problem <- addAssignmentsLB(problem, c(1, 2), c(2, 3))

resultOfComparison <- compareAssignments(problem)
```

deteriorateAssignment *Post factum analysis: deteriorate assignment*

Description

This function checks how much an alternative evaluations can be deteriorated so that that alternative would stay possibly (or necessarily) in at least some specific class. Deterioration is based on minimization value of `rho` in multiplication of an alternative evaluations on selected criteria by value `rho` (where `0 < rho <= 1`). **Note!** This function works for problems with only non-negative alternative evaluations.

Usage

```
deteriorateAssignment(alternative, atLeastToClass, criteriaManipulability,
  necessary, problem)
```

Arguments

`alternative`	An alternative for assignment deterioration.
`atLeastToClass`	An assignment to investigate.
`criteriaManipulability`	Vector containing a logical value for each criterion. Each value denotes whether multiplying by `rho` on corresponding criterion is allowed or not. At least one criterion has to be available for that manipulation.
`necessary`	Whether necessary or possible assignment is considered.
`problem`	Problem for which deterioration will be performed.

Value

Value of `rho` or `NULL` if given assignment is not possible in any scenario.

See Also

`improveAssignment`

Examples

```
perf <- matrix(c(5, 2, 1, 7, 0.5, 0.9, 0.4, 0.5), ncol = 2)
problem <- buildProblem(perf, 3, FALSE, c('g', 'g'), c(0, 0))
problem <- addAssignmentsLB(problem, c(1, 2), c(2, 3))

rho <- deteriorateAssignment(4, 1, c(TRUE, TRUE), FALSE, problem)
```

`drawUtilityPlots` *Draw marginal value functions and chart of alternative utilities*

Description

This function draws marginal value functions and alternative utilities chart.

Usage

```
drawUtilityPlots(problem, solution, printLabels = TRUE, criteria = NULL,
  plotsPerRow = 2, descending = NULL)
```

Arguments

`problem`	Problem.
`solution`	Solution.
`printLabels`	Whether to print labels.
`criteria`	Vector containing *0* for utility chart and/or indices of criteria for which marginal value functions should be plotted. If this parameter was `NULL` functions for all criteria and utility chart will be plotted (default `NULL`).
`plotsPerRow`	Number of plots per row (default `2`).
`descending`	Mode of sorting alternatives on utility chart: • `NULL` - unsorted, preserved `problem$perf` order, • `TRUE` - sorted descending by value of utility, • `FALSE` - sorted ascending by value of utility.

Details

This function is deprecated. Use `plotVF` and `plotComprehensiveValue`.

See Also

`plotVF` `plotComprehensiveValue`

explainAssignment *Explain assignment*

Description

This function allows to obtain explanation of an alternative assignment to a specific class interval or one class in case if assignment is necessary. The function returns all preferential reducts for an assignment relation.

Usage

```
explainAssignment(alternative, classInterval, problem)
```

Arguments

`alternative`	Index of an alternative.
`classInterval`	Two-element vector `c(l, u)` that represents an assignment of `alternative` to class interval `[C_l, C_u]` (`l <= u`).
`problem`	Problem for which computations will be performed.

Value

List of all preferential reducts for an assignment relation. If the assignment is not influenced by restrictions then empty list will be returned. Each element of the list is a preferential reduct represented as a vector of restriction indices. To identify preferential core use `getPreferentialCore`. To find out about restrictions by their indices use `getRestrictions`.

See Also

`getPreferentialCore` `getRestrictions` `calculateAssignments`

Examples

```
perf <- matrix(c(5, 2, 1, 7, 0.5, 0.9, 0.4, 0.5), ncol = 2)
problem <- buildProblem(perf, 3, FALSE, c('g', 'g'), c(0, 0))
problem <- addAssignmentsLB(problem, c(1, 2), c(2, 3))

possibleAssignments <- calculateAssignments(problem, FALSE)
alternative <- 4
assignment <- c(min(which(possibleAssignments[alternative, ])),
               max(which(possibleAssignments[alternative, ])))

preferentialReducts <- explainAssignment(alternative,
   assignment, problem)
preferentialCore <- getPreferentialCore(preferentialReducts)
coreRestrictions <- getRestrictions(problem, preferentialCore)
```

findInconsistencies *Find inconsistencies in preference information*

Description

This function finds sets of pieces of preference information that make problem inconsistent.

Usage

```
findInconsistencies(problem)
```

Arguments

problem	Problem to investigate.

Value

List of ordered by cardinality sets of indices of preference information that makes problem inconsistent. Use getRestrictions on sets to find out related preference information.

Examples

```
perf <- matrix(c(1, 2, 2, 1), ncol = 2)
problem <- buildProblem(perf, 3, TRUE, c('g', 'g'), c(0, 0))
problem <- addAssignmentsUB(problem, c(1, 1))
problem <- addAssignmentsLB(problem, c(2, 2))

checkConsistency(problem) # TRUE

problem <- addAssignmentsLB(problem, c(1, 3)) # added inconsistency

checkConsistency(problem) # FALSE

inconsistencies <- findInconsistencies(problem)

setsOfprefInfo <- lapply(inconsistencies,
                         function(x) { getRestrictions(problem, x) })
```

findRepresentativeFunction
Find representative utility function

Description

This function finds a representative utility function for a problem.

Usage

```
findRepresentativeFunction(problem, mode, relation = NULL)
```

Arguments

problem	Problem to investigate.
mode	An integer that represents a method of a computing representative utility function: • 0 - iterative mode, • 1 - compromise mode.
relation	A matrix of assignment pairwise comparisons (see compareAssignments). If the parameter is NULL, the relation will be computed.

Value

List with named elements:

- vf - list of 2-column matrices with marginal value functions (characteristic point in rows),
- thresholds,
- assignments,
- alternativeValues,
- epsilon.

NULL is returned if representative function cannot be found.

See Also

plotVF plotComprehensiveValue findSimpleFunction

Examples

```
perf <- matrix(c(5, 2, 1, 7, 0.5, 0.9, 0.4, 0.4), ncol = 2)
problem <- buildProblem(perf, 3, FALSE, c('g', 'g'), c(0, 0))
problem <- addAssignmentsLB(problem, c(1, 2), c(2, 3))

representativeFunction <- findRepresentativeFunction(problem, 0)
assignments <- representativeFunction$assignments
```

findSimpleFunction *Find one value function*

Description

This function finds single value function that is consistent with provided preferece information. Search is done by epsilon maximization.

Usage

```
findSimpleFunction(problem)
```

Arguments

problem	Problem

Value

List with named elements:

- vf - list of 2-column matrices with marginal value functions (characteristic point in rows),
- thresholds,
- assignments,
- alternativeValues,
- epsilon.

See Also

plotVF plotComprehensiveValue findRepresentativeFunction

Examples

```
perf <- matrix(c(5, 2, 1, 7, 0.5, 0.9, 0.4, 0.4), ncol = 2)
problem <- buildProblem(perf, 3, FALSE, c('g', 'g'), c(0, 0))
problem <- addAssignmentsLB(problem, c(1, 2), c(2, 3))

simpleFunction <- findSimpleFunction(problem)
```

findSolutionWithIncomplete
Find single value function from incomplete preference information

Description

This function finds a single value function from incomplete preference information for a problem.

Usage

```
findSolutionWithIncomplete(problem, stochasticResults, method, reg = 1e-20,
  accuracy = 1e-10)
```

Arguments

`problem`	Problem to investigate.
`stochasticResults`	Stochastic results (see `calculateStochasticResults`).
`method`	`cai-product`, `apoi-product`, or `combined-product`.
`reg`	Reg
`accuracy`	Accuracy

Value

List with named elements:

- `vf` - list of 2-column matrices with marginal value functions (characteristic point in rows),
- `thresholds`,
- `assignments`,
- `alternativeValues`,
- `epsilon`.

See Also

`calculateStochasticResults` `findRepresentativeFunction` `plotComprehensiveValue` `findSimpleFunction`

Examples

```
perf <- matrix(c(5, 2, 1, 7, 0.5, 0.9, 0.4, 0.4), ncol = 2)
problem <- buildProblem(perf, 3, FALSE, c('g', 'g'), c(0, 0))
problem <- addAssignmentsLB(problem, c(1, 2), c(2, 3))

stochasticResults <- calculateStochasticResults(problem, 100)
representativeFunction <- findSolutionWithIncomplete(problem, stochasticResults, "cai-product")
assignments <- representativeFunction$assignments
```

`getAssignments` *Get assignments*

Description

This function returns assignments for given model solution.

Usage

```
getAssignments(problem, solution)
```

Arguments

`problem`	Problem whose model was solved.
`solution`	Result of model solving (e.g. result of `findRepresentativeFunction` or `investigateUtility`).

Details

Function is deprecated. Solution already contains assignments.

Value

Vector of alternative assignments. Each element contains an index of a class that corresponding alternative was assigned to.

`getCharacteristicPoints` *Get characteristic points*

Description

This function extracts values of characteristic points from model solution.

Usage

```
getCharacteristicPoints(problem, solution)
```

Arguments

`problem`	Problem whose model was solved.
`solution`	Result of model solving (e.g. result of `findRepresentativeFunction` or `investigateUtility`).

Details

Function is deprecated. Solution already contains characteristic points.

Value

List of m matrices for each of m criteria. Each row c(g, u) of each matrix contains coordinates of a single characteristic point, where g - evaluation on corresponding criterion, u - marginal utility.

---

getMarginalUtilities *Get marginal utilities*

Description

This function extracts alternatives marginal values from model solution.

Usage

```
getMarginalUtilities(problem, solution)
```

Arguments

problem	Problem whose model was solved.
solution	Result of model solving (e.g. result of findRepresentativeFunction or investigateUtility).

Details

Function is deprecated. Solution already contains marginal utilities.

Value

A *n* x *m* matrix containing marginal values of n alternatives on m criteria.

---

getPreferentialCore *Identify preferential core*

Description

This function identifies preferential core.

Usage

```
getPreferentialCore(preferentialReducts)
```

Arguments

preferentialReducts	List of all preferential reducts (a result of explainAssignment).

Value

Preferential core as a vector of restriction indices. To find out about restrictions by their indices use `getRestrictions`.

See Also

`explainAssignment` `getRestrictions`

Examples

```
perf <- matrix(c(5, 2, 1, 7, 0.5, 0.9, 0.4, 0.5), ncol = 2)
problem <- buildProblem(perf, 3, FALSE, c('g', 'g'), c(0, 0))
problem <- addAssignmentsLB(problem, c(1, 2), c(2, 3))

possibleAssignments <- calculateAssignments(problem, FALSE)
alternative <- 4
assignment <- c(min(which(possibleAssignments[alternative, ])),
               max(which(possibleAssignments[alternative, ])))

preferentialReducts <- explainAssignment(alternative,
   assignment, problem)
preferentialCore <- getPreferentialCore(preferentialReducts)
coreRestrictions <- getRestrictions(problem, preferentialCore)
```

---

getRestrictions — *Get restrictions by indices*

Description

This function gets restrictions by indices.

Usage

```
getRestrictions(problem, indices)
```

Arguments

`problem`	Problem whose restrictions will be searched.
`indices`	A vector of restriction indices (eg. a result of calling `getPreferentialCore`.) Incorrect indices are skipped.

Value

List with named elements. Each element is a matrix which contains set of restrictions of same type.

See Also

`getPreferentialCore` `explainAssignment`

Examples

```
perf <- matrix(c(5, 2, 1, 7, 0.5, 0.9, 0.4, 0.5), ncol = 2)
problem <- buildProblem(perf, 3, FALSE, c('g', 'g'), c(0, 0))
problem <- addAssignmentsLB(problem, c(1, 2), c(2, 3))

possibleAssignments <- calculateAssignments(problem, FALSE)
alternative <- 4
assignment <- c(min(which(possibleAssignments[alternative, ])),
               max(which(possibleAssignments[alternative, ])))

preferentialReducts <- explainAssignment(alternative,
   assignment, problem)
preferentialCore <- getPreferentialCore(preferentialReducts)
coreRestrictions <- getRestrictions(problem, preferentialCore)
```

---

getThresholds *Get thresholds*

Description

This function extracts values of thresholds from solution.

Usage

```
getThresholds(problem, solution)
```

Arguments

`problem`	Problem whose model was solved.
`solution`	Result of model solving (e.g. result of `findRepresentativeFunction` or `investigateUtility`).

Details

Function is deprecated. Solution already contains thresholds.

Value

Vector containing `h-1` thresholds from `t_1` to `t_h-1` where `t_p-1` is lower threshold of class `C_p` and `h` is number of classes.

improveAssignment *Post factum analysis: improve assignment*

Description

This function calculates minimal `rho` by which alternative evaluations on selected criteria have to be multiplied for that alternative to be possibly (or necessarily) assigned to at least some specific class (`rho >= 1`). **Note!** This function works for problems with only non-negative alternative evaluations.

Usage

```
improveAssignment(alternative, atLeastToClass, criteriaManipulability,
  necessary, problem)
```

Arguments

`alternative`	An alternative for assignment improvement.
`atLeastToClass`	Desired assignment.
`criteriaManipulability`	Vector containing a logical value for each criterion. Each value denotes whether multiplying by `rho` on corresponding criterion is allowed or not. At least one criterion has to be available for that manipulation.
`necessary`	Whether necessary or possible assignment is considered.
`problem`	Problem for which improvement will be performed.

Value

Value of `rho` or `NULL` if given assignment is not possible in any scenario.

See Also

`deteriorateAssignment`

Examples

```
perf <- matrix(c(8, 2, 1, 7, 0.5, 0.9, 0.4, 0.5), ncol = 2)
problem <- buildProblem(perf, 3, FALSE, c('g', 'g'), c(0, 0))
problem <- addAssignmentsUB(problem, c(1, 2), c(2, 3))

# a_1 dominates a_4 and a_1 is assigned at most to class C_2
# How many times evaluations of a_4 should be improved
# that a_4 will be assigned possibly to class C_3?
rho <- improveAssignment(4, 3, c(TRUE, TRUE), FALSE, problem)
```

investigateUtility *Post factum analysis: check how much utility is missing*

Description

This function calculates missing value of an alternative utility for that alternative to be possibly (or necessarily) assigned to at least some specific class.

Usage

```
investigateUtility(alternative, atLeastToClass, necessary, problem)
```

Arguments

alternative	An alternative index.
atLeastToClass	An assignment to investigate.
necessary	Whether necessary or possible assignment is considered.
problem	Problem for investigation.

Value

List with named elements:

- ux - value of missing utility,
- solution - result of solving model. It can be used for further computations (getAssignments, getThresholds, getMarginalUtilities, getCharacteristicPoints).

NULL is returned if given assignment is not possible.

See Also

getMarginalUtilities getCharacteristicPoints getThresholds improveAssignment

Examples

```
perf <- matrix(c(8, 2, 1, 7, 0.5, 0.9, 0.4, 0.5), ncol = 2)
problem <- buildProblem(perf, 3, FALSE, c('g', 'g'), c(0, 0))
problem <- addAssignmentsUB(problem, c(1, 2), c(2, 3))

result <- investigateUtility(4, 3, FALSE, problem)
```

mergeAssignments *Merge different assignments*

Description

This function allows to merge different assignments, e.g. from various decision makers (group result, group assignment). There are four types of group assignments:

- **P**ossible **P**ossible - alternative a_i is **possibly** in class C_h **for at least one** decision maker,
- **P**ossible **N**ecessary - alternative a_i is **possibly** in class C_h **for all** decision makers,
- **N**ecessary **P**ossible - alternative a_i is **necessarily** in class C_h **for at least one** decision maker,
- **N**ecessary **N**ecessary - alternative a_i is **necessarily** in class C_h **for all** decision makers.

The first possible-necessary parameter depends on decision makers assignments computed earlier, and the second is define as function parameter.

Usage

```
mergeAssignments(assignmentList, necessary)
```

Arguments

`assignmentList`	List of assignment matrices (results of calling `calculateAssignments` function).
`necessary`	Whether necessary or possible merging.

Value

n x p logical matrix, where each row represents one of n alternatives and each column represents one of p classes. Element `[i, h]` is `TRUE` if alternative `a_i` can be assigned to class `C_h`.

See Also

`calculateAssignments`

Examples

```
perf <- matrix(c(5, 2, 1, 7, 0.5, 0.9, 0.4, 0.4), ncol = 2)
problem <- buildProblem(perf, 3, FALSE, c('g', 'g'), c(0, 0))
DM1Problem <- addAssignmentsLB(problem, c(1, 2), c(2, 3))
DM2Problem <- addAssignmentsLB(problem, c(2, 2), c(4, 2))

necessary <- FALSE
assignmentList <- list()
assignmentList[[1]] <- calculateAssignments(DM1Problem, necessary)
assignmentList[[2]] <- calculateAssignments(DM2Problem, necessary)
```

```
# generate possible - necessary assignments
PNAssignments <- mergeAssignments(assignmentList, TRUE)
```

---

plotComprehensiveValue *Plot comprehensive values of altarnatives*

Description

This function draws bar chart of comprehensive values of altarnatives.

Usage

```
plotComprehensiveValue(solution, order = "alternatives",
  showThresholds = FALSE, title = FALSE)
```

Arguments

solution	Solution to plot (e.g. result of findRepresentativeFunction, findSimpleFunction or investigateUtility).
order	Order of alternatives ("alternatives", "asc", "desc").
showThresholds	Whether to print threholds (dashed lines).
title	Title for chart or boolean value whether default title should be used.

Value

Plot.

See Also

findRepresentativeFunction findSimpleFunction investigateUtility plotVF

Examples

```
perf <- matrix(c(5, 2, 1, 7, 0.5, 0.9, 0.4, 0.4), ncol = 2)
problem <- buildProblem(perf, 3, FALSE, c('c', 'g'), c(3, 3))
problem <- addAssignmentsLB(problem, c(1, 2), c(2, 3))

representativeFunction <- findRepresentativeFunction(problem, 0)
plotComprehensiveValue(representativeFunction)
```

plotVF — *Plot value function*

Description

This function draws value function for selected criteria.

Usage

```
plotVF(solution, criteria = NULL, yAxis = "max", showAlternatives = FALSE,
  titles = TRUE, plotsPerRow = 2)
```

Arguments

- `solution` — Solution to plot (e.g. result of `findRepresentativeFunction`, `findSimpleFunction` or `investigateUtility`).
- `criteria` — Indices of criteria to plot. If NULL all criteria will be plotted.
- `yAxis` — Y axis limit (`"adjusted"` - maximal value on single plot, `"max"` - maximal value on all criteria, `"unit"` - one).
- `showAlternatives` — Whether to mark values of alternatives.
- `titles` — Vector of titles for charts or boolean value(s) whether default title should be used.
- `plotsPerRow` — Maximal plots per row.

See Also

`findRepresentativeFunction` `findSimpleFunction` `investigateUtility` `plotComprehensiveValue`

Examples

```
perf <- matrix(c(5, 2, 1, 7, 0.5, 0.9, 0.4, 0.4), ncol = 2)
problem <- buildProblem(perf, 3, FALSE, c('c', 'g'), c(3, 3))
problem <- addAssignmentsLB(problem, c(1, 2), c(2, 3))

representativeFunction <- findRepresentativeFunction(problem, 0)
plotVF(representativeFunction)
```

removeAssignmentPairwiseAtLeastComparisons

Remove assignment pairwise at least *comparisons*

Description

This function removes pairwise *at least* comparisons. For more information see `addPairwiseAtLeastComparisons`.

Usage

```
removeAssignmentPairwiseAtLeastComparisons(problem, ...)
```

Arguments

`problem`	Problem from which preference information will be removed
`...`	Comparisons as three-element vectors and/or two-element vectors. Each argument represents comparison to remove. If `c(i, j, k)` vector was provided a corresponding comparison will be removed. In case where two-element vector `c(i,j)` was given a comparison of an alternative *a_i* with *a_j* will be removed regardless of value of *k*. If a specific comparison was not found nothing will happen.

Value

Problem with removed comparisons.

Examples

```
# 4 alternatives, 2 gain criteria, 3 classes, monotonously increasing
# and general marginal value functions
perf <- matrix(c(5, 2, 1, 7, 0.5, 0.9, 0.4, 0.4), ncol = 2)
problem <- buildProblem(perf, 3, FALSE, c('g', 'g'), c(0, 0))

# add comparisons:
# alternative 2 to class at least as good as class of alternative 1
# alternative 4 to class at least better by 1 class then class
# of alternative 3
problem <- addAssignmentPairwiseAtLeastComparisons(problem,
   c(4, 3, 1), c(2, 1, 0))
# remove comparison between alternative 4 and 3
problem <- removeAssignmentPairwiseAtLeastComparisons(problem, c(4, 3))
```

`removeAssignmentPairwiseAtMostComparisons`
Remove assignment pairwise at most *comparisons*

Description

This function removes pairwise *at most* comparisons. For more information see `addPairwiseAtMostComparisons`.

Usage

```
removeAssignmentPairwiseAtMostComparisons(problem, ...)
```

Arguments

`problem`	Problem from which preference information will be removed
`...`	Comparisons as three-element vectors and/or two-element vectors. Each argument represents comparison to remove. If `c(i, j, k)` vector was provided a corresponding comparison will be removed. In case where two-element vector `c(i,j)` was given a comparison of an alternative *a_i* with *a_j* will be removed regardless of value of *k*. If a specific comparison was not found nothing will happen.

Value

Problem with removed comparisons.

Examples

```
# 4 alternatives, 2 gain criteria, 3 classes, monotonously increasing
# and general marginal value functions
perf <- matrix(c(5, 2, 1, 7, 0.5, 0.9, 0.4, 0.4), ncol = 2)
problem <- buildProblem(perf, 3, FALSE, c('g', 'g'), c(0, 0))

# add comparison:
# alternative 4 to class at most better by 1 class then class
# of alternative 3
problem <- addAssignmentPairwiseAtMostComparisons(problem, c(4, 3, 1))
# remove comparison between alternative 4 and 3
problem <- removeAssignmentPairwiseAtMostComparisons(problem, c(4, 3))
```

removeAssignmentsLB *Remove lower bound of alternative possible assignments*

Description

This function removes lower bounds of possible assignments from a problem.

Usage

```
removeAssignmentsLB(problem, ...)
```

Arguments

problem	Problem from which preference information will be removed.
...	Assignments as two-element vectors and/or integers. Each argument represents assignment to remove. If c(i, j) vector was provided an assignment of an alternative *a_i* to at least class *C_j* will be removed. In case where single value i was given an assignment of an alternative *a_i* will be removed regardless of class. If a specific assignment was not found nothing will happen.

Value

Problem with removed assignment examples.

Examples

```
# 4 alternatives, 2 gain criteria, 3 classes, monotonously increasing
# and general marginal value functions
perf <- matrix(c(5, 2, 1, 7, 0.5, 0.9, 0.4, 0.4), ncol = 2)
problem <- buildProblem(perf, 3, FALSE, c('g', 'g'), c(0, 0))

# add assignment examples: alternative 1 at least to class 2
# alternative 2 at least to class 3
problem <- addAssignmentsLB(problem, c(1, 2), c(2, 3))

# and remove the assignments
problem <- removeAssignmentsLB(problem, c(1, 2), 2)
```

removeAssignmentsUB *Remove upper bound of alternative possible assignments*

Description

This function removes upper bounds of possible assignments from a problem.

Usage

```
removeAssignmentsUB(problem, ...)
```

Arguments

`problem`	Problem from which preference information will be removed.
`...`	Assignments as two-element vectors and/or integers. Each argument represents assignment to remove. If `c(i, j)` vector was provided an assignment of an alternative *a_i* to at most class *C_j* will be removed. In case where single value `i` was given an assignment of an alternative *a_i* will be removed regardless of class. If a specific assignment was not found nothing will happen.

Value

Problem with removed assignment examples.

Examples

```
# 4 alternatives, 2 gain criteria, 3 classes, monotonously increasing
# and general marginal value functions
perf <- matrix(c(5, 2, 1, 7, 0.5, 0.9, 0.4, 0.4), ncol = 2)
problem <- buildProblem(perf, 3, FALSE, c('g', 'g'), c(0, 0))

# add assignment examples: alternative 1 at least to class 2
# alternative 2 at least to class 3
problem <- addAssignmentsLB(problem, c(1, 2), c(2, 3))

# and remove the assignments
problem <- removeAssignmentsLB(problem, c(1, 2), 2)
```

---

removeMaximalClassCardinalities

Remove maximal class cardinality restrictions

Description

This function allows to remove defined maximal cardinality of particular classes.

Usage

```
removeMaximalClassCardinalities(problem, ...)
```

Arguments

`problem`	Problem from which preference information will be removed.
`...`	Two-element vectors and/or integers. Each argument represents restriction to remove. If `c(i, j)` vector was provided then defined maximal cardinality *j* for class *C_i* will be removed. In case where single value `i` was given, a restriction for class *a_i* will be removed regardless of maximal cardinality value. If a specific restriction was not found nothing will happen.

Value

Problem with removed preference information.

Examples

```
# 4 alternatives, 2 gain criteria, 3 classes, monotonously increasing
# and general marginal value functions
perf <- matrix(c(5, 2, 1, 7, 0.5, 0.9, 0.4, 0.4), ncol = 2)
problem <- buildProblem(perf, 3, FALSE, c('g', 'g'), c(0, 0))

# set maximal class cardinalities:
# at most two alternatives could be assigned to class 2
# and at most one alternative could be assigned to class 3
problem <- addMaximalClassCardinalities(problem, c(2, 2), c(3, 1))
# remove defined restriction for class 2
problem <- removeMaximalClassCardinalities(problem, 2)
```

`removeMinimalClassCardinalities`

Remove minimal class cardinality restrictions

Description

This function allows to remove defined minimal cardinality of particular classes.

Usage

```
removeMinimalClassCardinalities(problem, ...)
```

Arguments

`problem`	Problem from which preference information will be removed.
`...`	Two-element vectors and/or integers. Each argument represents restriction to remove. If `c(i, j)` vector was provided then defined minimal cardinality *j* for class *C_i* will be removed. In case where single value `i` was given a restriction for class *a_i* will be removed regardless of minimal cardinality value. If a specific restriction was not found nothing will happen.

Value

Problem with removed preference information.

Examples

```
# 4 alternatives, 2 gain criteria, 3 classes, monotonously increasing
# and general marginal value functions
perf <- matrix(c(5, 2, 1, 7, 0.5, 0.9, 0.4, 0.4), ncol = 2)
problem <- buildProblem(perf, 3, FALSE, c('g', 'g'), c(0, 0))

# set minimal class cardinalities:
# at least one alternative has to be assigned to class 2
# and at least one alternative has to be assigned to class 3
problem <- addMinimalClassCardinalities(problem, c(2, 1), c(3, 1))
# remove defined restriction for class 2
problem <- removeMinimalClassCardinalities(problem, 2)
```

Index