

# Package ‘cnaOpt’

December 9, 2019

**Title** Optimizing Consistency and Coverage in Configurational Causal Modeling

**Version** 0.1.1

**Date** 2019-12-09

**Description** This is an add-on to the 'cna' package <<https://CRAN.R-project.org/package=cna>> comprising various functions for optimizing consistency and coverage scores of models of configurational comparative methods as Coincidence Analysis (CNA) and Qualitative Comparative Analysis (QCA). The function `conCovOpt()` calculates con-cov optima, `selectMax()` selects con-cov maxima among the con-cov optima, `DNFbuild()` can be used to build models actually reaching those optima, and `findOutcomes()` identifies those factor values in analyzed data that can be modeled as outcomes. For a theoretical introduction to these functions see <<https://people.uib.no/mba110/docs/ConCovOpt.pdf>>.

**Depends** R (>= 3.4.0), cna (>= 2.2.2)

**Imports** Rcpp, matrixStats, ggplot2, dplyr, stats, utils

**LinkingTo** Rcpp

**License** GPL (>= 2)

**Encoding** UTF-8

**LazyData** true

**NeedsCompilation** yes

**Author** Mathias Ambuehl [aut, cre, cph],  
Michael Baumgartner [aut, cph]

**Maintainer** Mathias Ambuehl <[mathias.ambuehl@consultag.ch](mailto:mathias.ambuehl@consultag.ch)>

**Repository** CRAN

**Date/Publication** 2019-12-09 09:50:02 UTC

## R topics documented:

<code>cnaOpt</code> . . . . .	2
<code>conCovOpt</code> . . . . .	3
<code>findOutcomes</code> . . . . .	6
<code>reprodAssign</code> . . . . .	7
<code>rreduce_ereduce</code> . . . . .	9
<code>selectMax</code> . . . . .	11

---

cnaOpt	<i>Find atomic solution formulas with optimal consistency and coverage</i>
--------	--

---

### Description

cnaOpt attempts to find atomic solution formulas (asfs) for a given outcome (inferred from crisp-set, "cs", or multi-value, "mv", data) that are optimal with respect to consistency and coverage.

### Usage

```
cnaOpt(x, outcome, ..., crit = quote(con * cov), cond = quote(TRUE))
```

### Arguments

x	A <a href="#">data.frame</a> or <a href="#">truthTab</a> of type "cs" or "mv"; if x is of type "mv", x must be a <a href="#">truthTab</a> .
outcome	A character string specifying one outcome, i.e. one factor in x.
...	Additional arguments passed to <a href="#">truthTab</a> , for instance type, <code>rm.dup.factors</code> , <code>rm.dup.factors</code> , or <code>case.cutoff</code> .
crit, cond	A quoted expression specifying additional criteria for selecting optimal solutions, passed to <a href="#">selectMax</a> .

### Details

cnaOpt infers causal models (atomic solution formulas, asf) for the outcome from data x that are optimal with respect to the optimality criterion `crit` and complying with conditions `cond`. Data x may be crisp-set ("cs") or multi-value ("mv"), but not fuzzy-set ("fs"). The function first calculates consistency and coverage optima (con-cov optima) for x, then selects the optimum that is best according to `crit` and `cond`, builds the canonical disjunctive normal form (DNF) realizing the best optimum and, finally, generates all minimal forms of that canonical DNF.

Roughly speaking, running cnaOpt amounts to sequentially executing [truthTab](#), [conCovOpt](#), [selectMax](#), [DNFbuild](#) and [condTbl](#).

### Value

cnaOpt returns a [data.frame](#) with additional class "condTbl". See the "Value" section in [?condTbl](#) for details.

### See Also

[cna](#), [conCovOpt](#)

## Examples

```
# Example 1: Real-life crisp-set data, d.educate.
(res_opt1 <- cnaOpt(d.educate, "E"))

# Using the pipe operator (%>%), the steps processed by cnaOpt in the
# call above can be reproduced as follows:
library(dplyr)
conCovOpt(d.educate, "E") %>% selectMax %>% DNFbuild("E", reduce = "erreduce") %>%
  paste("<-> E") %>% condTbl(d.educate)

# Example 2: Simulated crisp-set data.
dat1 <- data.frame(
  A = c(1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0),
  B = c(0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0),
  C = c(0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0),
  D = c(1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1),
  E = c(1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1),
  F = c(0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1)
)

(res_opt2 <- cnaOpt(dat1, "E"))

# Change the optimality criterion.
cnaOpt(dat1, "E", crit = quote(pmin(con, cov)))
# Impose an additional condition.
cnaOpt(dat1, "E", cond = quote(con >= 0.9))

# Example 3: All logically possible configurations.
(res_opt3 <- cnaOpt(full.tt(4), "D")) # All combinations are equally bad.

# Example 4: Real-life multi-value data, d.pban.
cnaOpt(mvtt(d.pban), outcome = "PB=1")
cnaOpt(mvtt(d.pban), outcome = "PB=1", crit = quote(pmin(con, cov)))
cnaOpt(mvtt(d.pban), outcome = "PB=1", cond = quote(con > 0.93))
cnaOpt(mvtt(d.pban), outcome = "PB=0")
cnaOpt(mvtt(d.pban), outcome = "PB=0", cond = quote(con > 0.93))
cnaOpt(d.pban, type = "mv", outcome = "F=2")
cnaOpt(d.pban, type = "mv", outcome = "F=2", cond = quote(con > 0.75))
```

---

conCovOpt

*Find consistency and coverage optima for configurational data*


---

## Description

conCovOpt issues pairs of optimal consistency and coverage scores that atomic solution formulas (asf) of an outcome inferred from configurational data can possibly reach.

**Usage**

```

conCovOpt(x, outcome = NULL,
          type = if (inherits(x, "truthTab")) attr(x, "type") else "cs",
          maxCombs = 1e+07, approx = FALSE, allConCov = FALSE)
## S3 method for class 'conCovOpt'
print(x, ...)
## S3 method for class 'conCovOpt'
plot(x, con = 1, cov = 1, ...)

```

**Arguments**

x	A <a href="#">data.frame</a> or <a href="#">truthTab</a> . In the print- and plot-method: The output of conCovOpt.
outcome	A character vector of one or several factors in x.
type	A character string specifying the data type; one of "cs", "mv", "fs". Must be specified if the data type is not "cs".
maxCombs	Maximal number of combinations that will be tested for optimality.
approx	Logical; if TRUE, an exhaustive search is only approximated; if FALSE, an exhaustive search is conducted.
allConCov	Logical; if TRUE, all possible con-cov scores are stored as attribute "allConCov"; if FALSE, only the optimal con-cov scores are stored.
con, cov	Numeric scalars between 0 and 1 indicating consistency and coverage thresholds marking the area of "good" models in a square drawn in the plot. Points within the square correspond to models reaching these thresholds.
...	Currently not used.

**Details**

Prior to actual CNA or QCA analyses, conCovOpt calculates consistency and coverage optima for models (i.e. atomic solution formulas, asf) of an outcome inferred from data x.

An ordered pair (con, cov) of consistency and coverage scores is a **con-cov optimum** for outcome Y=k in data x iff it is not excluded (on principled grounds) for an asf of Y=k inferred from x to reach (con, cov) but excluded to score better on one element of the pair and at least as well on the other.

conCovOpt calculates con-cov optima by executing the following steps:

1. if x is a data frame, aggregate x in a truthTab,
2. build exo-groups with constant values in all factors other than the outcome,
3. assign output values to each exo-group that reproduce the behavior of outcome as closely as possible,
4. calculate con-cov scores for each assignment resulting in step 3,
5. eliminate all non-optimal scores.

In case of "cs" and "mv" data, at least one actual model (asf) inferrable from x and reaching an optimum's consistency and coverage scores is guaranteed to exist for every con-cov optimum. The function `DNFbuild` can be used to build these optimal models. The same does not hold for "fs" data. In case of "fs" data, it merely holds that the existence of a model reaching an optimum's consistency and coverage scores cannot be excluded prior to an actual application of `cna`.

## Value

An object of class 'conCovOpt'. The exo-groups resulting from step 2 are stored as attribute "exoGroups", the lists of output values resulting from step 3 are stored as attribute "reprodList" (reproduction list), and all possible con-cov scores are stored as attribute "allConCov".

## See Also

`truthTab`, `selectMax`, `DNFbuild`

## Examples

```
(cco.irrigate <- conCovOpt(d.irrigate))
conCovOpt(d.irrigate, outcome = c("R","W"))
# Plot method.
plot(cco.irrigate)
plot(cco.irrigate, con = .8, cov = .8)

dat1 <- d.autonomy[15:30, c("EM","SP","CO","AU")]
(cco1 <- conCovOpt(dat1, type = "fs", outcome = "AU"))

print(cco1, digits = 3, row.names = TRUE)
plot(cco1)

# Exo-groups (configurations with constant values in all factors other than the outcome).
attr(cco1$A, "exoGroups")

# Rep-list (list of values optimally reproducing the outcome).
attr(cco1$A, "reprodList")

# allConCov (add all possible con-cov scores, not just optimal ones).
cco1_acc <- conCovOpt(dat1, type = "fs", outcome="AU", allConCov = TRUE)
attr(cco1_acc$A, "allConCov")
# If the allConCov table has been built, it is passed to the output of selectMax().
sm1 <- selectMax(cco1_acc)
attr(sm1$A, "allConCov")

dat2 <- d.pacts
# Maximal number of combinations exceeds maxCombs.
(cco2 <- conCovOpt(dat2, type = "fs", outcome = "PACT")) # Generates a warning
# Increase maxCombs.
(cco2_full <- try(conCovOpt(dat2, type = "fs", outcome = "PACT",
  maxCombs=1e+08))) # Takes long or fails due to memory shortage
# Approximate an exhaustive search.
```

```
(cco2_approx <- conCovOpt(dat2, type = "fs", outcome = "PACT", approx = TRUE))
```

---

findOutcomes	<i>Identify the factors that can possibly be modeled as outcomes prior to running CNA</i>
--------------	---

---

### Description

Prior to running CNA (or any other configurational comparative method), `findOutcomes` identifies those factors in data `x` that can be modeled as outcomes relative to specified consistency and coverage thresholds `con` and `cov`.

### Usage

```
findOutcomes(x, con = 1, cov = 1, ...)
```

### Arguments

<code>x</code>	A <code>data.frame</code> or <code>truthTab</code> ; if <code>x</code> is of type multi-value ("mv") or fuzzy-set ("fs"), <code>x</code> must be a <code>truthTab</code> .
<code>con</code> , <code>cov</code>	Numeric scalars between 0 and 1 specifying consistency and coverage thresholds.
<code>...</code>	Additional arguments passed to <code>truthTab</code> , for instance <code>type</code> , <code>rm.dup.factors</code> , <code>rm.dup.factors</code> , or <code>case.cutoff</code> .

### Details

`findOutcomes` first runs `conCovOpt` to find the con-cov optima for all factors in `x` and then applies `selectMax` to select those factors with con-cov optima meeting the consistency and coverage thresholds specified in `con` and `cov`.

In case of "cs" and "mv" data, an actual model (asf) meeting the specified `con` and `cov` thresholds is guaranteed to exist for every factor value with an entry `TRUE` in the outcome column. The function `DNFbuild` can be used to build these models. The same does not hold for "fs" data. In case of "fs" data, an entry `TRUE` in the outcome column simply means that the existence of a model reaching the specified `con` and `cov` thresholds cannot be excluded prior to an actual application of `cna`.

### Value

A `data.frame`.

### See Also

`conCovOpt`, `selectMax`, `selectCases`, `DNFbuild`, `full.tt`

**Examples**

```

# CS data
findOutcomes(d.educate)
findOutcomes(d.educate, con = 0.75, cov = 0.75)

# A causal chain.
target1 <- "(A + B <-> C)*(C + D <-> E)"
dat1 <- selectCases(target1, full.tt(target1))
findOutcomes(dat1)

# A causal cycle.
target2 <- "(A + Y1 <-> B)*(B + Y2 <-> A)*(A + Y3 <-> C)"
dat2 <- selectCases(target2, full.tt(target2))
findOutcomes(dat2)

# MV data
findOutcomes(mvtt(d.pban)) # no possible outcomes at con = cov = 1
findOutcomes(mvtt(d.pban), con = 0.8)
findOutcomes(d.pban, type = "mv", con = 0.8, cov= 0.8)

# FS data
findOutcomes(fsst(d.jobsecurity)) # no possible outcomes at con = cov = 1
findOutcomes(d.jobsecurity, type = "fs", con = 0.86)

```

reprodAssign

*Build disjunctive normal forms realizing con-cov optima***Description**

reprodAssign generates the output values of a disjunctive normal form (DNF) reaching a con-cov optimum. DNFbuild builds a DNF realizing a targeted con-cov optimum; it only works for crisp-set and multi-value data.

**Usage**

```

reprodAssign(x, outcome, id = xi$id)
DNFbuild(x, outcome, reduce = c("rreduce", "ereduce", "none"), id = xi$id)

```

**Arguments**

x	An object produced by <a href="#">selectMax</a> .
outcome	A character string specifying <i>one</i> outcome in <code>attr(x, "truthTab")</code> .
id	An integer referring to the identifier of the targeted con-cov optimum.

reduce            A character string: if "rreduce" or "ereduce", the canonical DNF realizing the con-cov optimum is freed of redundancies using [rreduce](#) or [ereduce](#), respectively; if "none", the unreduced canonical DNF is returned. reduce=TRUE is interpreted as "rreduce", reduce=FALSE and reduce=NULL as "none".

### Details

An atomic CNA model (asf) accounts for the behavior of the outcome in terms of a redundancy-free DNF. `reprodAssign` generates the output values such a DNF has to return in order to reach a con-cov optimum stored in an object of class 'selectMax'. If the data stored in `attr(x, "truthTab")` is of type "cs" or "mv", `DNFbuild` builds a concrete DNF realizing the targeted con-cov optimum. (For data of type "fs" an error is returned.) If `reduce = "rreduce"` (default), *one* (randomly selected) redundancy-free DNF is built using [rreduce](#); if `reduce = "ereduce"`, *all* redundancy-free DNFs are built using [ereduce](#); if `reduce = "none"`, the non-reduced canonical DNF is returned. The argument `id` allows for selecting a targeted con-cov optimum via its identifier (see examples below).

### Value

`reprodAssign`: A vector of scores. `DNFbuild`: A Boolean formula in disjunctive normal form (DNF).

### See Also

[conCovOpt](#), [selectMax](#), [condTbl](#), [reprodAssign](#)

### Examples

```
# CS data, d.educate
cco1 <- conCovOpt(d.educate)
best1 <- selectMax(cco1)
reprodAssign(best1, outcome = "E")
DNFbuild(best1, outcome = "E")
DNFbuild(best1, outcome = "E", reduce = FALSE) # canonical DNF
DNFbuild(best1, outcome = "E", reduce = "ereduce") # all redundancy-free DNFs
DNFbuild(best1, outcome = "E", reduce = "rreduce") # one redundancy-free DNF
DNFbuild(best1, outcome = "E", reduce = "none") # canonical DNF

# Simulated mv data
datMV <- data.frame(
  A = c(3,2,1,1,2,3,2,2,2,1,1,2,3,2,2,2,1,2,3,3,3,1,1,1,3,1,2,1,2,3,3,2,2,2,1,2,2,3,2,1,2,1,3,3),
  B = c(1,2,3,2,1,1,2,1,2,2,3,1,1,1,2,3,1,3,3,3,1,1,3,2,2,1,1,3,3,2,3,1,2,1,2,2,1,1,2,2,3,3,3,3),
  C = c(1,3,3,3,1,1,1,2,2,3,3,1,1,2,2,2,3,1,1,2,1,2,2,3,3,1,2,2,2,3,2,1,1,2,2,2,1,1,1,2,2,1,1,2),
  D = c(3,1,2,2,1,1,1,1,1,1,1,2,2,2,2,2,2,3,3,3,1,1,1,1,1,2,2,2,2,2,3,1,1,1,1,1,2,2,2,2,3,3,3),
  E = c(3,2,2,3,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,2,2,2,2,2,2,2,2,2,2,2,3,3,3,3,3,3,3,3,3,3,3,3)
)

# Apply conCovOpt and selectMax.
cco2 <- conCovOpt(mvtt(datMV))
best2 <- selectMax(cco2)
```



```

# Apply DNFbuild to build one redundancy-free DNF reaching best2.
(formula1 <- DNFbuild(best2, outcome = "D=3"))
# formula1 reaches the con-cov score stored in best2 for outcome "D=3".
condTbl(paste0(formula1, "<-> D=3"), mvtt(datMV))
# Build all redundancy-free DNFs reaching best2.
DNFbuild(best2, outcome = "D=3", reduce = "ereduce")
# Any factor value in datMV can be treated as outcome.
(formula2 <- DNFbuild(best2, outcome = "E=3"))
condTbl(paste0(formula2, "<-> E=3"), mvtt(datMV))
# Any con-cov optimum in cco2 can be targeted via its identifier.
(formula3 <- DNFbuild(best2, outcome = "E=3", id = 508))
condTbl(paste0(formula3, "<-> E=3"), mvtt(datMV))

# Simulated fs data
datFS <- data.frame(
  A = c(.73, .85, .94, .36, .73, .79, .39, .82, .15, .12, .67, .27, .3),
  B = c(.21, .03, .91, .64, .39, .12, .06, .7, .73, .15, .88, .73, .36),
  C = c(.61, 0, .61, 1, .94, .15, .88, .27, .12, .12, .27, .15, .15),
  D = c(.64, .67, .3, .06, .33, .03, .76, .94, .67, .76, .18, .27, .36),
  E = c(.91, .94, .67, .85, .73, .79, .24, .09, .03, .21, .33, .36, .27)
)

# Apply conCovOpt and selectMax.
cco3 <- conCovOpt(datFS, outcome = "E", type = "fs", allConCov = TRUE)
best3 <- selectMax(cco3)

# Apply reprodAssign.
reprodAssign(best3, outcome = "E")
# Select a con-cov optimum in cco3 via its identifier.
reprodAssign(best3, outcome = "E", id = 252)

# DNFbuild does not work for fs data; it generates an error.
try(DNFbuild(best3, outcome = "E"))

```

---

rreduce\_ereduce

*Eliminate redundancies from disjunctive normal forms (DNF)*


---

## Description

ereduce and rreduce implement different algorithmic approaches to eliminate redundancies from disjunctive normal forms (DNF), i.e. disjunctions of conjunctions of literals. If there are several minimal DNF, ereducer will return them all, while rreduce selects one at random.

## Usage

```

ereduce(cond, x = full.tt(cond), full = !missing(x), simplify2constant = TRUE)
rreduce(cond, x = full.tt(cond), full = !missing(x), verbose = FALSE, maxiter = 1000,
simplify2constant = TRUE)

```

**Arguments**

cond	A character string specifying a disjunctive normal form; can be either crisp-set or multi-value.
x	A truthTab or data.frame; can be either crisp-set or multi-value.
full	Logical; if TRUE (the default), redundancies are eliminated relative to full.tt(x), otherwise relative to x.
simplify2constant	Logical; if TRUE (the default), a tautologous or contradictory cond is reduced to a constant "1" or "0", respectively. If FALSE, a minimal tautology or contradiction, i.e. "A+a" or "A*a", will result.
verbose	Logical; if TRUE, the reduction process will be traced in the console.
maxiter	Maximal number of iterations.

**Details**

ereduce and rreduce eliminate conjuncts and disjuncts from a DNF cond as long as the consistency and coverage of cond in data x does not change, that is, as long as the result of condition(cond, x) remains the same. The only required argument is cond. If x is not provided, redundancies are eliminated relative to full.tt(cond). If x is provided and full = TRUE, redundancies are eliminated relative to full.tt(x). If x is provided and full = FALSE, redundancies are eliminated relative to x.

While ereducer generates all redundancy-free forms of cond, rreduce only returns one randomly chosen one. rreduce is faster than ereducer, but often incomplete. ereducer, in a nutshell, searches for minimal hitting sets in cond preventing cond from being false in data x.

**Value**

Redundancy-free disjunctive normal form (DNF).

**See Also**

[full.tt](#), [conCovOpt](#), [DNFbuild](#).

**Examples**

```
# Logical redundancies.
cond1 <- "A*b + a*B + A*C + B*C"
ereduce(cond1)
rreduce(cond1)
cond2 <- "A*b + a*B + A*B + a*b"
ereduce(cond2)
ereduce(cond2, simplify2constant = FALSE)

# Redundancy elimination relative to simulated cs data.
dat1 <- data.frame(
  A = c(0, 0, 0, 0, 1, 1, 0, 1),
  B = c(0, 1, 0, 1, 1, 0, 0, 0),
```

```

    C = c(1, 1, 0, 1, 1, 0, 1, 1),
    D = c(0, 0, 0, 0, 0, 1, 1, 1))
cco1 <- conCovOpt(dat1, "D")
best1 <- selectMax(cco1)
formula1 <- DNFbuild(best1, outcome = "D", reduce = FALSE)
# ereducer
ereducer(formula1, dat1, full = FALSE)
# rreducer
rreducer(formula1, dat1, full = FALSE)
rreducer(formula1, dat1, full = FALSE, verbose = TRUE)

# Redundancy elimination relative to simulated mv data.
dat2 <- data.frame(
  A = c(3,2,1,1,2,3,2,2,2,1,1,2,3,2,2,1,2,3,3,3,1,1,1,3,1,2,1,2,3,3,2,2,2,1,2,2,3,2,1,2,1,3,3),
  B = c(1,2,3,2,1,1,2,1,2,2,3,1,1,1,2,3,1,3,3,3,1,1,3,2,2,1,1,3,3,2,3,1,2,1,2,2,1,1,2,2,3,3,3,3),
  C = c(1,3,3,3,1,1,1,2,2,3,3,1,1,2,2,2,3,1,1,2,1,2,2,3,3,1,2,2,2,3,2,1,1,2,2,2,1,1,1,2,2,1,1,2),
  D = c(3,1,2,2,1,1,1,1,1,1,1,2,2,2,2,2,2,3,3,3,1,1,1,1,1,2,2,2,2,2,3,1,1,1,1,1,2,2,2,2,2,3,3,3),
  E = c(3,2,2,3,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,2,2,2,2,2,2,2,2,2,2,2,2,3,3,3,3,3,3,3,3,3,3,3,3,3,3)
)
cco2 <- conCovOpt(dat2, "D=3", type="mv")
best2 <- selectMax(cco2)
formula2 <- DNFbuild(best2, outcome = "D=3", reduce = FALSE)
# ereducer
ereducer(formula2, mvtt(dat2), full = FALSE)
# rreducer
rreducer(formula2, mvtt(dat2), full = FALSE)

# Any Boolean expressions.
cond <- "!(A*B*C)*(a*b*c)" # or "A + B*(D + e) <-> C"
x <- selectCases(cond)
cond <- cna::getCond(x) # Returns a DNF equivalent to cond, but with many redundancies.
ereducer(cond)
rreducer(cond)

```

---

selectMax

*Select the con-cov optimum from a 'conCovOpt' object that is best according to a specified optimality criterion*


---

## Description

selectMax selects a con-cov optimum from a 'conCovOpt' object that is best according to some specified optimality criterion. multipleMax checks a 'selectMax' object for multiple solutions with identical values in the optimality criterion.

## Usage

```

selectMax(x, crit = quote(con * cov), cond = quote(TRUE))
multipleMax(x, outcome)

```

## Arguments

x	In selectMax: an object output by <code>conCovOpt</code> . In multipleMax: an object output by selectMax.
crit	A quoted expression specifying the optimality criterion (see examples).
cond	A quoted expression specifying additional constraints imposed on the optimality criterion (see examples).
outcome	A character string specifying a single outcome in the original data.

## Details

While `conCovOpt` identifies *all* con-cov optima in an analyzed data set, `selectMax` selects *one* con-cov optimum from a 'conCovOpt' object that is best according to the optimality criterion specified in the argument `crit`. The default is to select a con-cov maximum: An ordered pair (con, cov) of consistency and coverage scores is a **con-cov maximum** for outcome  $Y=k$  in data  $\delta$  iff (con, cov) is a con-cov optimum for  $Y=k$  in  $\delta$  with highest product of consistency and coverage (con-cov product). However, the argument `crit` allows for specifying any other optimality criterion, e.g. `pmin(con, cov)`, `pmax(con, cov)`, etc. If the 'conCovOpt' object contains multiple outcomes, the selection of a best con-cov optimum is done separately for each outcome.

Whereas `selectMax` selects only one con-cov optimum satisfying `crit`, `multipleMax` selects all elements in an `allConCov` list contained in the 'conCovOpt' object reaching identical scores on the optimality criterion. It is executed for *one* outcome only (see the examples below).

Via the column `id` in the output of `selectMax` it is possible to select one among many equally good maxima, for instance, by means of `reprodAssign` (see the examples below).

## Value

`selectMax` returns an object of class 'selectMax'.

`multipleMax` returns a `data.frame`.

## See Also

[conCovOpt](#), [reprodAssign](#)

See also examples in [conCovOpt](#).

## Examples

```
dat1 <- d.autonomy[15:30, c("EM", "SP", "CO", "AU")]
(cco1 <- conCovOpt(dat1, type = "fs", outcome = "AU"))
selectMax(cco1)
selectMax(cco1, cond = quote(con > 0.95))
selectMax(cco1, cond = quote(cov > 0.98))
selectMax(cco1, crit = quote(pmin(con, cov)))
selectMax(cco1, crit = quote(pmax(con, cov)), cond = quote(cov > 0.9))

# Multiple equally good maxima.
(cco2 <- conCovOpt(dat1, type = "fs", outcome = "AU", allConCov = TRUE))
(sm2 <- selectMax(cco2, cond = quote(con > 0.93)))
```

```
multipleMax(sm2, "AU")  
# Each maximum corresponds to a different rep-assignment, which can be selected  
# using the id argument.  
reprodAssign(sm2, "AU", id = 10)  
reprodAssign(sm2, "AU", id = 11)  
reprodAssign(sm2, "AU", id = 13)
```

# Index

`cna`, 2, 5, 6  
`cnaOpt`, 2  
`conCovOpt`, 2, 3, 6, 8, 10, 12  
`condTbl`, 2, 8  
  
`data.frame`, 2, 4, 6  
`DNFbuild`, 2, 5, 6, 10  
`DNFbuild (reprodAssign)`, 7  
  
`erreduce`, 8  
`erreduce (rreduce_erreduce)`, 9  
  
`findOutcomes`, 6  
`full.tt`, 6, 10  
  
`multipleMax (selectMax)`, 11  
  
`plot.conCovOpt (conCovOpt)`, 3  
`print.conCovOpt (conCovOpt)`, 3  
  
`reprodAssign`, 7, 8, 12  
`rreduce`, 8  
`rreduce (rreduce_erreduce)`, 9  
`rreduce_erreduce`, 9  
  
`selectCases`, 6  
`selectMax`, 2, 5–8, 11  
  
`truthTab`, 2, 4–6