# Package 'cnaOpt'

July 8, 2022

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

**Version** 0.5.2

**Date** 2022-07-06

**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 Baumgartner and Ambuehl (2021) <doi:10.1177/0049124121995554>.

**Depends** R (>= 3.5.0), cna (>= 3.2.0)

**Imports** Rcpp (>= 1.0.7), matrixStats, ggplot2, dplyr, stats, utils

**LinkingTo** Rcpp

**License** GPL (>= 2)

**Encoding** UTF-8

**NeedsCompilation** yes

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

**Maintainer** Mathias Ambuehl <mathias.ambuehl@consultag.ch>

**Repository** CRAN

**Date/Publication** 2022-07-08 14:00:11 UTC

## R topics documented:

---

`cnaOpt` *Find atomic solution formulas with optimal consistency and coverage*

---

## 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 the model fit parameters consistency and coverage (cf. Baumgartner and Ambuehl 2021).

## Usage

```
cnaOpt(x, outcome, ..., reduce = c("ereduce", "rreduce", "none"),
       niter = 1, crit = quote(con * cov), cond = quote(TRUE),
 approx = FALSE, maxCombs = 1e7)
```

## Arguments

| | |
|---|---|
| `x` | A `data.frame` or `configTable` of type `"cs"` or `"mv"`. |
| `outcome` | A character string specifying one outcome, i.e. one factor value in `x`. |
| `...` | Additional arguments passed to `configTable`, for instance `rm.dup.factors`, `rm.dup.factors`, or `case.cutoff`. |
| `reduce` | A character string: if `"ereduce"` or `"rreduce"`, the canonical DNF realizing the con-cov optimum is freed of redundancies using `ereduce` or `rreduce` (possibly repeatedly, see `niter`), respectively; if `"none"`, the unreduced canonical DNF is returned. `reduce = TRUE` is interpreted as `"rreduce"`, `reduce = FALSE` and `reduce = NULL` as `"none"`. |
| `niter` | An integer value indicating the number of repetitive applications of `rreduce`. `niter` will be ignored (with a warning) if `reduce` is not equal to `"rreduce"`. Note that repeated applications may yield identical solutions and that duplicate solutions are eliminiated, so that the number of resulting solutions can be smaller than `niter`. |
| `crit` | Quoted expression specifying a numeric criterion to be maximized when selecting the best solutions among the ones that meet criterion `cond`, for example, `quote(min(con,cov))` or `quote(0.8*con + 0.2*cov)`, etc. |
| `cond` | Quoted expression specifying a logical criterion to be imposed on the solutions inferred from `x` before selecting the best solutions on the basis of `crit`, for example, `quote(con > 0.85)` or `quote(con > cov)`, etc. |
| `approx` | As in `conCovOpt`. |
| `maxCombs` | Maximal number of combinations that will be tested for optimality. If the number of necessary iterations exceeds `maxCombs`, `cnaOpt` will stop executing and return an error message stating the necessary number of iterations. Early termination can then be avoided by increasing `maxCombs` accordingly. This argument is passed to `conCovOpt` and `ereduce`. |

## Details

cnaOpt implements a procedure introduced in Baumgartner and Ambuehl (2021). It infers causal models (atomic solution formulas, asf) for the outcome from data x that comply with the logical condition cond and maximize the numeric criterion crit. Data x may be crisp-set ("cs") or multi-value ("mv"), but not fuzzy-set ("fs"). The function proceeds as follows:

1. it calculates consistency and coverage optima (con-cov optima) for x;
2. it selects the optima that meet cond;
3. among those optima, it selects those that maximize crit;
4. it builds the canonical disjunctive normal forms (DNF) of the selected optima
5. it generates all minimal forms of those canonical DNFs

Roughly speaking, running cnaOpt amounts to sequentially executing configTable, conCovOpt, selectMax, DNFbuild and condTbl.

In the default setting, cnaOpt attempts to build all optimal solutions using ereduce. But that may be too computationally demanding because the space of optimal solutions can be very large. If the argument reduce is set to "rreduce", cnaOpt builds one arbitrarily selected optimal solution, which typically terminates quickly. By giving the argument niter a non-default value, say, 20, the process of selecting one optimal solution under reduce = "rreduce" is repeated 20 times. As the same solutions will be generated on some iterations and duplicates are not returned, the output may contain less models than the value given to niter. If reduce is not set to "rreduce", niter is ignored with a warning.

## Value

cnaOpt returns a data.frame with additional classes "cnaOpt" and "condTbl". See the "Value" section in ?condTbl for details.

## References

Baumgartner, Michael and Mathias Ambuehl. 2021. "Optimizing Consistency and Coverage in Configurational Causal Modeling." *Sociological Methods & Research*. doi:10.1177/0049124121995554.

## 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(reduce = "ereduce") %>%
  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 maximality criterion.
cnaOpt(dat1, "E", crit = quote(min(con, cov)))
# Change the selection condition.
cnaOpt(dat1, "E", cond = quote(con >= 0.9))
# Build all con-cov optima with coverage above 0.9 that maximize min(con, cov).
cnaOpt(dat1, "E", crit = quote(min(con, cov)), cond = quote(cov > 0.9))
# Different values of the reduce argument.
cnaOpt(dat1, "E", reduce = "none") # canonical DNF
cnaOpt(dat1, "E", reduce = "rreduce") # one randomly drawn optimal solution
# Iterate random solution generation 10 times.
cnaOpt(dat1, "E", reduce = "rreduce", niter = 10)

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

# Example 4: Real-life multi-value data, d.pban.
cnaOpt(d.pban, outcome = "PB=1")
cnaOpt(d.pban, outcome = "PB=1", crit = quote(0.8*con + 0.2*cov))
cnaOpt(d.pban, outcome = "PB=1", cond = quote(con > 0.9))
cnaOpt(d.pban, outcome = "PB=0")
cnaOpt(d.pban, outcome = "PB=0", cond = quote(con > 0.9))
cnaOpt(d.pban, outcome = "F=2")
cnaOpt(d.pban, outcome = "F=2", crit = quote(0.8*con + 0.2*cov))

# Example 5: High computational demand.
dat2 <- configTable(d.performance[,1:8], frequency = d.performance$frequency)
try(cnaOpt(dat2, outcome = "SP"))   # error because too computationally demanding
# The following call does not terminate because of reduce = "ereduce".
try(cnaOpt(dat2, outcome = "SP", approx = TRUE))
# We could increase maxCombs, as in the line below
## Not run: cnaOpt(dat2, outcome = "SP", approx = TRUE, maxCombs = 1.08e+09)
# but this takes very long to terminate.
# Alternative approach: Produce one (randomly selected) optimal solution using reduce = "rreduce".
cnaOpt(dat2, outcome = "SP",  approx = TRUE, reduce = "rreduce")
# Iterate the previous call 10 times.
cnaOpt(dat2, outcome = "SP", approx = TRUE, reduce = "rreduce", niter = 10)
# Another alternative: Use ereduce for minimization but introduce a case.cutoff.
cnaOpt(dat2, outcome = "SP", case.cutoff = 10)
```

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 (cf. Baumgartner and Ambuehl 2021).

## Usage

```
conCovOpt(x, outcome = NULL, ..., rm.dup.factors = FALSE, rm.const.factors = FALSE,
          maxCombs = 1e+07, approx = FALSE, allConCov)
## S3 method for class 'conCovOpt'
print(x, ...)
## S3 method for class 'conCovOpt'
plot(x, con = 1, cov = 1, ...)
```

## Arguments

| | |
|---|---|
| x | In conCovOpt: a data.frame or configTable. In the print- and plot-method: an output of conCovOpt. |
| outcome | A character vector of one or several factor values in x. |
| ... | In conCovOpt: arguments passed to configTable, e.g. case.cutoff. The '...' are currently not used in plot.conCovOpt. |
| rm.dup.factors | Logical; defaults to FALSE (which is different from configTable). If TRUE, all but the first of a set of factors with identical values in x are removed. |
| rm.const.factors | Logical; defaults to FALSE (which is different from configTable). If TRUE, factors with constant values in x are removed. |
| maxCombs | Maximal number of combinations that will be tested for optimality. If the number of necessary iterations exceeds maxCombs, conCovOpt will stop executing and return an error message stating the necessary number of iterations. Early termination can then be avoided by increasing maxCombs accordingly. |
| approx | Logical; if TRUE, an exhaustive search is only approximated; if FALSE, an exhaustive search is conducted. |
| allConCov | Defunct argument (as of package version 0.5.0). See the remark in ?multipleMax. |
| 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. |

## Details

`conCovOpt` implements a procedure introduced in Baumgartner and Ambuehl (2021). It calculates consistency and coverage optima for models (i.e. atomic solution formulas, asf) of an `outcome` inferred from data `x` prior to actual CNA or QCA analyses.

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 (based e.g. on the data structure) 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 `configTable`,
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.

The implementation of step 4 calculates con-cov scores of about 10 million output value assignments in reasonable time, but step 3 may result in considerably more assignments. In such cases, the argument `approx` may be set to its non-default value `"TRUE"`, which determines that step 4 is only executed for those assignments closest to the `outcome`'s median value. This is an efficient approach for finding many, but possibly not all, con-cov optima.

In case of crisp-set and multi-value 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 fuzzy-set data. In fuzzy-set 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).

## References

Baumgartner, Michael and Mathias Ambuehl. 2021. "Optimizing Consistency and Coverage in Configurational Causal Modeling." *Sociological Methods & Research*. doi:10.1177/0049124121995554.

## See Also

`configTable`, `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, 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")

dat2 <- d.pacts
# Maximal number of combinations exceeds maxCombs.
(cco2 <- conCovOpt(dat2, outcome = "PACT")) # Generates a warning
# Increase maxCombs.
(cco2_full <- try(conCovOpt(dat2, outcome = "PACT",
  maxCombs=1e+08))) # Takes a long time to terminate
# Approximate an exhaustive search.
(cco2_approx1 <- conCovOpt(dat2, outcome = "PACT", approx = TRUE))
selectMax(cco2_approx1)
# The search space can also be reduced by means of a case cutoff.
(cco2_approx2 <- conCovOpt(dat2, outcome = "PACT", case.cutoff=2))
selectMax(cco2_approx2)
```

---

# conCovOpt_utils *Build disjunctive normal forms realizing con-cov optima*

---

## Description

`reprodAssign` generates the output values of disjunctive normal forms (DNFs) reaching con-cov optima. `DNFbuild` builds a DNF realizing a targeted con-cov optimum; it only works for crisp-set and multi-value data (cf. Baumgartner and Ambuehl 2021).

## Usage

```
reprodAssign(x, outcome = names(x), id = xi$id)
DNFbuild(x, outcome = names(x), reduce = c("ereduce", "rreduce", "none"),
         id = xi$id, maxCombs = 1e7)
```

## Arguments

| | |
|---|---|
| x | An object produced by selectMax. |
| outcome | A character string specifying *one* outcome value in attr(x, "configTable"). |
| id | An integer vector referring to the identifier of the targeted con-cov optimum or optima. |
| reduce | A character string: if "ereduce" or "rreduce", the canonical DNF realizing the con-cov optimum is freed of redundancies using ereduce or rreduce, respectively; if "none", the unreduced canonical DNF is returned. reduce=TRUE is interpreted as "rreduce", reduce=FALSE and reduce=NULL as "none". |
| maxCombs | Passed to ereduce if reduce = "ereduce"; ignored otherwise. (See ereduce for details.) |

## 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, "configTable") are crisp-set or multi-value, DNFbuild builds the DNFs realizing the targeted con-cov optimum. (For fuzzy-set data an error is returned.) If reduce = "ereduce" (default), *all* redundancy-free DNFs are built using ereduce; if reduce = "rreduce" (more computationally efficient), *one* (randomly selected) redundancy-free DNF is built using rreduce; 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 matrix of scores. DNFbuild: A Boolean formula in disjunctive normal form (DNF).

## References

Baumgartner, Michael and Mathias Ambuehl. 2021. "Optimizing Consistency and Coverage in Configurational Causal Modeling." *Sociological Methods & Research*. doi:10.1177/0049124121995554.

## See Also

conCovOpt, selectMax, condTbl

## 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(1,2,3,2,1,1,2,1,2,2,3,1,1,1,2,3,1,3,3,3,3,1,1,3,2,2,1,1,3,3,2,3,1,2,1,2,2,1,1,2,2,3,3,3,3),
 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(3,1,2,2,1,1,1,1,1,1,1,2,2,2,2,2,2,2,3,3,3,1,1,1,1,1,2,2,2,2,2,3,1,1,1,1,1,2,2,2,2,2,3,3,3),
 E = c(3,2,2,3,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,2,2,2,2,2,2,2,2,2,2,2,3,3,3,3,3,3,3,3,3,3,3,3,3)
)

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

# Apply DNFbuild to build the redundancy-free DNFs reaching best2.
(formula1 <- DNFbuild(best2, outcome = "D=3"))
# Both DNFs in formula1 reache the con-cov score stored in best2 for outcome "D=3".
condTbl(paste0(formula1, "<-> D=3"), datMV)
# Build only one redundancy-free DNF reaching best2.
DNFbuild(best2, outcome = "D=3", reduce = "rreduce")
# Any factor value in datMV can be treated as outcome.
(formula2 <- DNFbuild(best2, outcome =  "E=3", reduce = "rreduce"))
condTbl(paste0(formula2, "<-> E=3"), 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"), 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"))
(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"))
```

---

ereduce *Find all minimal disjunctive normal forms (DNF) of an input DNF*

---

## Description

ereduce builds all minimal disjunctive normal forms corresponding to an input DNF. It is similar to rreduce, which, however, only builds one minimal DNF at random.

## Usage

```
ereduce(cond, x = full.ct(cond), full = !missing(x),
        simplify2constant = TRUE, maxCombs = 1e7)
```

## Arguments

| | |
|---|---|
| cond | A character string specifying a disjunctive normal form (DNF); can be either crisp-set or multi-value. |
| x | A configTable or data.frame; can be either crisp-set or multi-value. |
| full | Logical; if TRUE, redundancies are eliminated relative to full.ct(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. |
| maxCombs | Maximal number of iterations that will be ran in the most time-consuming step. If the number of necessary iterations exceeds maxCombs, ereduce will stop executing and return an error message stating the necessary number of iterations. Early termination can then be avoided by increasing maxCombs accordingly. |

## Details

ereduce eliminates conjuncts and disjuncts from a DNF cond 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.ct(cond).

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

## Value

A vector of redundancy-free disjunctive normal forms (DNF).

## See Also

rreduce, full.ct, conCovOpt, DNFbuild.

## Examples

```
# Logical redundancies.
cond1 <- "A*b + a*B + A*C + B*C"
ereduce(cond1)
rreduce(cond1) # repeated calls generate different outputs
```

```
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))
# ereduce
ereduce(formula1, dat1, full = FALSE)
# rreduce
rreduce(formula1, dat1, full = FALSE)

# Redundancy elimination relative to simulated mv data.
dat2 <- data.frame(
 A = c(3,2,1,1,2,3,2,2,2,1,1,2,3,2,2,2,1,2,3,3,3,1,1,1,3,1,2,1,2,3,3,2,2,2,1,2,2,3,2,1,2,1,3,3),
 B = c(1,2,3,2,1,1,2,1,2,2,3,1,1,1,2,3,1,3,3,3,3,1,1,3,2,2,1,1,3,3,2,3,1,2,1,2,2,1,1,2,2,3,3,3,3),
 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(3,2,2,3,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,2,2,2,2,2,2,2,2,2,2,2,3,3,3,3,3,3,3,3,3,3,3)
)
cco2 <- conCovOpt(dat2, "D=3")
best2 <- selectMax(cco2)
(formula2 <- DNFbuild(best2, outcome = "D=3", reduce = FALSE))
# ereduce
ereduce(formula2, dat2, full = FALSE)
# rreduce
rreduce(formula2, 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
ereduce(cond)
rreduce(cond)
```

---

`findOutcomes` *Identify the factors that can possibly be modeled as outcomes prior to running CNA*

---

## 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,
             rm.dup.factors = FALSE, rm.const.factors = FALSE, ...)
```

## Arguments

| | |
|---|---|
| x | A data.frame or configTable. |
| con, cov | Numeric scalars between 0 and 1 specifying consistency and coverage thresholds. |
| rm.dup.factors | Logical; defaults to FALSE. If TRUE, all but the first of a set of factors with identical values in x are removed. |
| rm.const.factors | Logical; defaults to FALSE. If TRUE, factors with constant values in x are removed. |
| ... | Additional arguments passed to conCovOpt and configTable, for instance approx or case.cutoff. |

## 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 crisp-set and multi-value 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 fuzzy-set data. In case of fuzzy-set 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.ct

## Examples

```
# Crisp-set data.
findOutcomes(d.educate)
findOutcomes(d.educate, con = 0.75, cov = 0.75)
x <- configTable(d.performance[,1:8], frequency = d.performance$frequency)
findOutcomes(x, con = .7, cov = .7) # too computationally demanding
# Approximate by passing approx = TRUE to conCovOpt().
findOutcomes(x, con = .7, cov = .7, approx = TRUE)
# Approximate by passing a case cutoff to configTable().
findOutcomes(x, con = .7, cov = .7, case.cutoff = 10)
```

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

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

# Multi-value data.
findOutcomes(d.pban) # no possible outcomes at con = cov = 1
findOutcomes(d.pban, con = 0.8)
findOutcomes(d.pban, con = 0.8, cov= 0.8)

# Fuzzy-set data.
findOutcomes(d.jobsecurity) # no possible outcomes at con = cov = 1
findOutcomes(d.jobsecurity, con = 0.86)
```

---

`selectMax` *Select the con-cov optima from a '*`conCovOpt`*' object that maximize a specified optimality criterion*

---

## Description

`selectMax` selects the optima from a 'conCovOpt' object that maximize a specified optimality criterion (cf. Baumgartner and Ambuehl 2021).

## Usage

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

## Arguments

| | |
|---|---|
| `x` | An object output by `conCovOpt`. |
| `crit` | Quoted expression specifying a numeric criterion to be maximized when selecting from the con-cov optima that meet criterion `cond`, for example, `min(con,cov)` or `0.8*con + 0.2*cov`, etc. |
| `cond` | Quoted expression specifying a logical criterion to be imposed on the con-cov optima in `x` before selecting the optima maximizing `crit`, for example, `con > 0.85` or `con > cov`, etc. |
| `warn` | Logical; if `TRUE`, `selectMax()` returns a warning if no solution is found. |
| `outcome` | A character string specifying a single outcome value in the original data. |

## Details

While conCovOpt identifies *all* con-cov optima in an analyzed data set, selectMax selects those optima from a 'conCovOpt' object x that comply with a logical condition cond and fare best according to the numeric optimality criterion crit. The default is to select so-called *con-cov maxima*, meaning con-cov optima with highest product of consistency and coverage. But the argument crit allows for specifying any other numeric optimality criterion, e.g. min(con, cov), max(con, cov), or 0.8*con + 0.2*cov, etc. (see Baumgartner and Ambuehl 2021). If x contains multiple outcomes, the selection of the best con-cov optima is done separately for each outcome.

As of package version 0.5.0, the function multipleMax is obsolete. It is kept for backwards compatibility only.

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'.

## References

Baumgartner, Michael and Mathias Ambuehl. 2021. "Optimizing Consistency and Coverage in Configurational Causal Modeling." *Sociological Methods & Research*. doi:10.1177/0049124121995554.

## See Also

conCovOpt, reprodAssign

See also examples in conCovOpt.

## Examples

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

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