Package: qcaERT (via r-universe)

November 21, 2026

Title Enhanced Robustness Tests for Qualitative Comparative Analysis

Version 0.1.3

Description Provides functions for assessing and visualizing robustness in Qualitative Comparative Analysis (QCA) workflows built with the 'QCA' package, including calibration thresholds, inclusion cutoffs, frequency cutoffs, case influence, subsample stability, alternative analysis settings, theory-specific condition sets, cluster-specific patterns, and solution summaries. Methods build on Dusa (2019) <doi:10.1007/978-3-319-75668-4> and Ragin (2014, ISBN:9780520280038).

URL https://CRAN.R-project.org/package=qcaERT, https://github.com/marisguia/qcaERT, https://marisguia.github.io/qcaERT/

BugReports https://github.com/marisguia/qcaERT/issues

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Depends R (>= 3.5.0)

RoxygenNote 7.3.3

VignetteBuilder knitr, rmarkdown

Imports methods, QCA, stats, utils

Suggests ggplot2, knitr, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

NeedsCompilation no

Author Breno A. H. Marisguia [aut, cre, cph]

Maintainer Breno A. H. Marisguia <marisguiabreno@gmail.com>

Repository https://cran.r-universe.dev

Date/Publication 2026-11-20 22:30:02 UTC

RemoteUrl https://github.com/cran/qcaERT

RemoteRef HEAD

RemoteSha 67afed7047ba5417263db4a7dafe8cf2e4ccced9

Contents

qcaERT-package . . . 2
altset.test . . . 5
calib.test . . . 11
cluster.test . . . 18
incl.test . . . 22
loo.test . . . 26
ncut.test . . . 30
qcaERT_conventions . . . 33
qcaERT_plots . . . 36
qcaERT_tests . . . 41
sol.chart . . . 43
sol.df . . . 47
subsample.test . . . 51
theory.test . . . 56
whr_calib_spec . . . 59
whr_calibrated . . . 60
whr_raw . . . 61

Index **62**

`qcaERT-package` *qcaERT: Enhanced robustness tests for QCA*

Description

qcaERT extends the usual Qualitative Comparative Analysis (QCA) with tools for assessing how stable and robust a QCA analysis is. It can test alternative calibration thresholds, inclusion cutoffs, truth table frequency cutoffs, case deletion, subsampling, alternative solutions, cluster-specific analyses and different theoretical configurations. It also offers a compact way to organize and visualize QCA solutions. The package is built around the QCA package (Dusa 2019) workflow using `QCA::calibrate()`, `QCA::truthTable()`, `QCA::minimize()`, and `QCA::findRows()`. It is not intended to cover every possible QCA variant or calibration transformation.

Start Here

If you know the robustness concern but not the function name, start with `?qcaERT_tests`, which maps qcaERT's tools.

qcaERT includes three World Happiness Report demonstration objects. See `?whr_raw`, `?whr_calibrated`, and `?whr_calib_spec`.

A typical R workflow for sufficient analysis using qcaERT is:

1. calibrate data with QCA::calibrate(),
2. build a truth table with QCA::truthTable(),
3. minimize with QCA::minimize(),
4. run one or more qcaERT robustness tests, and
5. inspect print(x), as.data.frame(x), x$diagnostics, and, where available, plot(x).

Function Family

The main qcaERT functions are:

- calib.test() for calibration-threshold robustness.
- incl.test() for truth table inclusion-cutoff robustness.
- ncut.test() for truth table frequency-cutoff robustness.
- loo.test() for leave-one-out case influence.
- subsample.test() for repeated subsample stability (advanced).
- altset.test() for sampled alternative analysis settings that can combine calibration, inclusion-cutoff, and frequency-cutoff perturbations.
- theory.test() for comparing theoretically motivated configurations using the same outcome, truth-table cutoffs, solution type, exclusion handling, and model-selection settings.
- cluster.test() for cluster, group, or repeated-unit heterogeneity.
- sol.df() for converting QCA minimization objects into compact solution tables.
- sol.chart() for rendering sol.df() tables as solution charts.

Returned Objects

Most robustness functions return R objects with a common structure:

- diagnostics: the detailed table, useful for inspection and troubleshooting.
- results: a cleaner table.
- settings: the analysis settings used to create the result.
- supporting components such as baseline, bounds, by_direction, by_case, by_run, by_draw, or summary, depending on the test.

print() gives a concise summary. as.data.frame() returns the main clean table. cluster.test() and theory.test() use structured results lists: cluster.test() returns overview, clusters, and units, while theory.test() returns models, pairwise, and solutions.

Common Conventions

The family-wide argument and output conventions are described in ?qcaERT_conventions. It explains solution controls such as solution, include, dir.exp, which_M, and i_mode; exclusion handling and its function-specific exceptions; calibration specifications; and the common returned-object structure.

Plotting

Plotting requires ggplot2. Current plot methods are available for `calib.test()`, `incl.test()`, and `theory.test()` results. `sol.chart()` provides a visual presentation for `sol.df()` tables. See `?qcaERT_plots`.

Learn More

- `?qcaERT_tests` for choosing the right robustness test.
- `?qcaERT_conventions` for common argument and output conventions.
- `?qcaERT_plots` for plotting `calib_test`, `incl_test`, and `theory_test` objects, and for charting `sol.df()` tables.
- `vignette("qcaERT-overview", package = "qcaERT")` for a guided introduction.
- `vignette("qcaERT-result-objects", package = "qcaERT")` for the common returned-object structure.
- `vignette("qcaERT-calibration", package = "qcaERT")` for calibration specifications, scale-aware perturbations, and alternative sets.
- For comprehensive guidance, worked examples, and interpretation of qcaERT diagnostics, see Marisguia (2026), *A Guide to qcaERT: Robustness Diagnostics in QCA*. Read the browser-based edition at https://marisguia.github.io/qcaERT/ and cite the archived Version 1.0 at doi:10.5281/zenodo.21685193.
- `news(package = "qcaERT")` for development changes.
- `citation(package = "qcaERT")` for citation information.

Recommended citation

Marisguia, B. A. H. (2026). qcaERT: Enhanced Robustness Tests for Qualitative Comparative Analysis. R package version 0.1.3. https://CRAN.R-project.org/package=qcaERT. doi:10.32614/CRAN.package.qcaERT.

Please also cite the QCA package and the methodological sources relevant to the analysis.

Author(s)

Breno A. H. Marisguia

References

Marisguia, B. A. H. 2026. *A Guide to qcaERT: Robustness Diagnostics in QCA*. Version 1.0. Zenodo. doi:10.5281/zenodo.21685193.

Dusa, Adrian. 2019. *QCA with R. A Comprehensive Resource*. Cham: Springer. https://adriandusa.com/research/books/2019-QCA/.

Ragin, C. C. 2014. *The Comparative Method: Moving Beyond Qualitative and Quantitative Strategies*. Oakland, California: University of California Press.

See Also

Useful links:

- https://CRAN.R-project.org/package=qcaERT
- https://github.com/marisguia/qcaERT
- https://marisguia.github.io/qcaERT/
- Report bugs at https://github.com/marisguia/qcaERT/issues

`altset.test` *Alternative-set robustness test for QCA solutions*

Description

Repeatedly generates alternative analyses by jointly varying calibration anchors, the raw consistency threshold, and the frequency cutoff for truth table rows, then reruns the QCA analysis for each generated specification. For each specification, the function records whether the solution under evaluation changes, whether parameters of fit change, which conditions or outcome were recalibrated, and whether the analysis failed to produce a solution.

Usage

```
altset.test(
  raw.data,
  calib.data,
  outcome,
  conditions,
  calib_spec,
  test.conditions = conditions,
  test.outcome = FALSE,
  anchors_to_test = NULL,
  solution = "all",
  include = NULL,
  which_M = 1,
  unit_step = NULL,
  unit_step_divisor = 10,
  calib_max_steps = 20,
  incl.cut = 1,
  incl_step = 0.01,
  incl_max_steps = 20,
  n.cut = 1,
  ncut_step = 1,
  ncut_max_steps = 20,
  dir.exp = NULL,
  i_mode = c("all", "C1P1"),
  exclude_mode = c("recompute", "static", "none"),
  exclude_recompute = list(type = 2),
```

```
  exclude_static = NULL,
  n_draws = 100,
  fit_tol = 1e-06,
  seed = NULL,
  progress = TRUE,
  verbose = FALSE,
  ...
)

## S3 method for class 'altset_test'
print(x, row.names = FALSE, ...)

## S3 method for class 'altset_test'
as.data.frame(x, ...)
```

Arguments

raw.data	A data frame object containing the raw, uncalibrated condition columns, and the raw outcome column when test.outcome = TRUE.
calib.data	A data frame object containing the calibrated outcome and calibrated condition columns used for the baseline analysis. For every tested condition or outcome, the stored memberships must match those reconstructed from raw.data and calib_spec.
outcome	Name of the outcome column in calib.data. This must be a single non-empty character string.
conditions	Character vector of calibrated condition names in calib.data.
calib_spec	Named list of calibration specifications, one per condition, plus one for the outcome when test.outcome = TRUE. Each entry must contain raw, type, and thresholds, may contain method, and may contain calibrate, a named list of additional QCA::calibrate() arguments for that set. The entries for tested sets define the reference calibrations checked against calib.data.
test.conditions	Subset of conditions to perturb when generating random alternative calibration draws. May be NULL only when test.outcome = TRUE, in which case only the outcome calibration is perturbed.
test.outcome	Logical; if TRUE, also perturb the outcome calibration. The outcome must be represented in calib_spec, and must not be included in conditions.
anchors_to_test	Character vector of anchors eligible for perturbation, or NULL to use all anchors implied by each tested set's calibration method. Crisp sets use "T". Fuzzy direct three-threshold sets use "E", "C", and "I". Fuzzy indirect sets use "T1", "T2", and so on. Fuzzy direct six-threshold sets use "E1", "C1", "I1", "I2", "C2", and "E2".
solution	Solution type to monitor. Accepted values are "all", "con" or "conservative", "par" or "parsimonious", and "int" or "intermediate".
include	Optional minimization include setting. Currently, this argument accepts only NULL, "", or "?".

`which_M` Positive integer giving which solution alternative to use when minimization returns multiple models.

`unit_step` Numeric step size used to move calibration thresholds when building candidate alternative draws. Supply either one value applied to all calibrated sets or one value per condition, plus the outcome when `test.outcome = TRUE`. If `NULL`, qcaERT computes scale-aware steps for the sets selected by `test.conditions` and `test.outcome`.

`unit_step_divisor` Positive number used to compute `unit_step` automatically when `unit_step = NULL`. The default is `10`.

`calib_max_steps` Positive integer giving the maximum number of threshold moves away from the baseline used when building alternative calibration candidates.

`incl.cut` Baseline inclusion cutoff passed to `QCA::truthTable()`.

`incl_step` Positive step size used to build the candidate grid of inclusion-cutoff values around `incl.cut`.

`incl_max_steps` Positive integer giving the maximum number of stepwise moves away from the baseline `incl.cut` used to build the candidate grid.

`n.cut` Baseline truth table frequency cutoff passed to `QCA::truthTable()`.

`ncut_step` Positive integer step size used to build the candidate grid of frequency-cutoff values around `n.cut`.

`ncut_max_steps` Positive integer giving the maximum number of stepwise moves away from the baseline `n.cut` used to build the candidate grid.

`dir.exp` Directional expectations used when the monitored solution is intermediate.

`i_mode` Character string controlling intermediate-solution selection. Accepted values are `"all"` and `"C1P1"`.

`exclude_mode` Character string controlling how excluded rows are handled for parsimonious and intermediate minimization. `"recompute"` recalculates exclusions from each drawn truth table using `exclude_recompute`, `"static"` reuses `exclude_static`, and `"none"` does not use exclusions.

`exclude_recompute` Named list of arguments passed to `QCA::findRows()` when `exclude_mode = "recompute"`.

`exclude_static` Already computed exclusion object reused when `exclude_mode = "static"`.

`n_draws` Positive integer giving the number of random alternative draws to run.

`fit_tol` Non-negative tolerance used when deciding whether fit values changed.

`seed` Optional finite integer seed passed to `set.seed()` before generating the random draws.

`progress` Logical; if `TRUE` and the session is interactive, show a varying cosmetic message and a text progress bar. The display uses package-private cosmetic state and does not consume the random-number stream controlled by `seed`.

`verbose` Logical; if `TRUE`, print a completion message after each draw.

...	Additional arguments routed through the QCA workflow. Arguments matching QCA::truthTable() are forwarded to truth table construction; remaining minimization arguments are filtered and forwarded to QCA::minimize(). The function also looks in ... for include, dir.exp, or direxp if those arguments were not supplied explicitly. In print.altset_test(), ... is passed to print.data.frame(). In as.data.frame.altset_test(), ... is ignored.
x	An altset_test object returned by altset.test().
row.names	Logical; passed to print.data.frame() by print.altset_test().

Details

The function first runs a baseline analysis using calib.data, incl.cut, and n.cut. It then generates n_draws random alternative draws.

Rows in raw.data and calib.data are paired before recalibration. Informative row names must be unique and identical in the same order on both inputs. If both inputs use only default row names (1, ..., n), pairing is positional and the inputs must not be independently reordered.

Before the baseline analysis is run, the function reconstructs each tested condition or outcome from raw.data and calib_spec and verifies that the resulting memberships match the corresponding column in calib.data. A mismatch stops the test because the alternative calibrations would otherwise be centered on a different reference calibration. Conditions not selected for perturbation retain their stored memberships in calib.data and are not included in this equality check.

Each draw samples:

- one admissible inclusion cutoff from the grid defined by incl.cut, incl_step, and incl_max_steps,
- one admissible frequency cutoff from the grid defined by n.cut, ncut_step, and ncut_max_steps, and
- one admissible alternative calibration specification sampled from calib_spec by moving method-specific anchors for test.conditions, and for the outcome when test.outcome = TRUE, by integer multiples of each set's unit_step.

The function rejects a sampled draw if it is identical to the baseline on all three dimensions and resamples until it gets a non-baseline draw or reaches the internal retry limit.

Progress display is deliberately isolated from analytical randomness. Consequently, a fixed seed produces the same alternative draws and result object whether progress is TRUE or FALSE; only the interactive console display differs.

The shared solution-control, exclusion, calibration-specification, and returned-object conventions are described in ?qcaERT_conventions.

Value

An object of class altset_test with the following components:

summary A list with summary values named n_draws, n_same_solution, n_fit_compared, n_same_fit, score_solution, score_fit, score_total, score_solution_by_solution_type, and score_fit_by_solution_t

baseline A list containing the baseline analysis, baseline draw metadata, exclusion information, selected solution terms used for comparison, fit information, and status information.

`diagnostics` A detailed data frame with one row per draw. It includes the sampled `incl.cut` and `n.cut`, run status, solution-change information, fit-change information, changed-set information, and error information when a draw fails.

`results` A compact data frame with the columns `draw`, `incl.cut`, `n.cut`, `status`, `n_changed_sets`, `changed_sets`, `changed_roles`, `solution_change`, `fit_changed_types`, `n_fit_deltas`, and `max_abs_fit_delta`. `n_fit_deltas` is `NA` when no fit comparison was available and is zero when fit was compared but no difference exceeded `fit_tol`.

`by_draw` A named list with one entry per draw. Each entry stores the full run result, solution-comparison information, fit-comparison information, and draw-level fit details.

`settings` A list containing the analysis settings used to build the result object.

`print.altset_test()` prints a concise summary, the `results` table, draw availability counts, and solution-type-specific formula- and fit-preservation tables with their comparison denominators. `as.data.frame.altset_test()` returns the `results` table.

See Also

`calib.test()`, `incl.test()`, `ncut.test()`, `loo.test()`, `subsample.test()`, `theory.test()`, `cluster.test()`, `sol.df()`

Examples

```
library(QCA)
data(LR)

conditions <- c("DEV", "URB", "LIT", "IND", "STB")
outcome <- "SURV"
dir_exp <- rep("1", length(conditions))

thresholds <- list(
  DEV = findTh(LR$DEV, groups = 7),
  URB = findTh(LR$URB, groups = 4),
  LIT = findTh(LR$LIT, groups = 4),
  IND = findTh(LR$IND, groups = 4),
  STB = findTh(LR$STB, groups = 4),
  SURV = findTh(LR$SURV, groups = 4)
)

dat <- LR
dat$DEV <- calibrate(LR$DEV, type = "fuzzy", thresholds = thresholds$DEV)
dat$URB <- calibrate(LR$URB, type = "fuzzy", thresholds = thresholds$URB)
dat$LIT <- calibrate(LR$LIT, type = "fuzzy", thresholds = thresholds$LIT)
dat$IND <- calibrate(LR$IND, type = "fuzzy", thresholds = thresholds$IND)
dat$STB <- calibrate(LR$STB, type = "fuzzy", thresholds = thresholds$STB)
dat$SURV <- calibrate(LR$SURV, type = "fuzzy", thresholds = thresholds$SURV)

calib_spec <- list(
  DEV = list(raw = "DEV", type = "fuzzy", method = "direct", thresholds = thresholds$DEV),
  URB = list(raw = "URB", type = "fuzzy", method = "direct", thresholds = thresholds$URB),
  LIT = list(raw = "LIT", type = "fuzzy", method = "direct", thresholds = thresholds$LIT),
  IND = list(raw = "IND", type = "fuzzy", method = "direct", thresholds = thresholds$IND),
```

```
  STB = list(raw = "STB", type = "fuzzy", method = "direct", thresholds = thresholds$STB)
)

calib_spec_outcome <- calib_spec
calib_spec_outcome$SURV <- list(
  raw = "SURV",
  type = "fuzzy",
  method = "direct",
  thresholds = thresholds$SURV
)

# Common use: sample alternative calibration, incl.cut, and n.cut settings.
altset_out <- altset.test(
  raw.data = LR,
  calib.data = dat,
  outcome = outcome,
  conditions = conditions,
  calib_spec = calib_spec,
  test.conditions = c("DEV", "URB"),
  unit_step = NULL,
  unit_step_divisor = 10,
  calib_max_steps = 5,
  incl.cut = 0.8,
  incl_step = 0.02,
  incl_max_steps = 5,
  n.cut = 1,
  ncut_step = 1,
  ncut_max_steps = 2,
  n_draws = 50,
  seed = 123,
  solution = "all",
  dir.exp = dir_exp,
  progress = TRUE
)

altset_out
as.data.frame(altset_out)
altset_out$summary

# Special case: include the calibrated outcome among sampled calibration
# perturbations. The outcome stays out of conditions.
altset_outcome <- altset.test(
  raw.data = LR,
  calib.data = dat,
  outcome = outcome,
  conditions = conditions,
  calib_spec = calib_spec_outcome,
  test.conditions = c("DEV", "URB"),
  test.outcome = TRUE,
  unit_step = NULL,
  unit_step_divisor = 10,
  calib_max_steps = 5,
  incl.cut = 0.8,
```

```
  incl_step = 0.02,
  incl_max_steps = 5,
  n.cut = 1,
  ncut_step = 1,
  ncut_max_steps = 2,
  n_draws = 50,
  seed = 456,
  solution = "all",
  dir.exp = dir_exp,
  progress = TRUE
)

as.data.frame(altset_outcome)
```

`calib.test` *Calibration-threshold robustness test for QCA solutions*

Description

Perturbs calibration thresholds for selected conditions, and optionally the outcome, one anchor and one direction at a time, and checks when the monitored QCA solution changes. For each tested path, the function records the starting threshold, the last value that preserved the baseline solution, the first value that changed the solution or triggered an error, and the reason the path stopped.

Usage

```
calib.test(
  raw.data,
  calib.data,
  outcome,
  conditions,
  calib_spec,
  test.conditions = conditions,
  test.outcome = FALSE,
  anchors_to_test = NULL,
  solution = "all",
  include = NULL,
  which_M = 1,
  unit_step = NULL,
  unit_step_divisor = 10,
  max_steps = 20,
  incl.cut = 1,
  n.cut = 1,
  dir.exp = NULL,
  i_mode = c("all", "C1P1"),
  exclude_mode = c("recompute", "static", "none"),
```

```
  exclude_recompute = list(type = 2),
  exclude_static = NULL,
  result_shape = c("wide", "long"),
  progress = TRUE,
  ...
)

## S3 method for class 'calib_test'
print(x, row.names = FALSE, ...)

## S3 method for class 'calib_test'
as.data.frame(x, ...)
```

Arguments

raw.data	A data frame object containing the raw, uncalibrated condition columns.
calib.data	A data frame object containing the calibrated outcome and calibrated condition columns used for the analysis.
outcome	Name of the outcome column in calib.data. This can be supplied as a character string or as an unquoted name.
conditions	Character vector of calibrated condition names in calib.data. This can be supplied as a character vector or as unquoted names.
calib_spec	Named list of calibration specifications. When test.outcome = FALSE, it must contain one entry per condition. When test.outcome = TRUE, it must contain one entry per condition plus one entry named by outcome. Each entry must contain raw, type, and thresholds, may contain method, and may contain calibrate, a named list of additional QCA::calibrate() arguments.
test.conditions	Subset of conditions to perturb. Use NULL only when test.outcome = TRUE and no condition calibration should be tested.
test.outcome	Logical; if TRUE, also perturb the outcome calibration while keeping the outcome out of conditions.
anchors_to_test	Character vector of anchors to test, or NULL to test all anchors implied by each set's calibration method. Crisp sets use "T". Fuzzy direct three-threshold sets use "E", "C", and "I". Fuzzy direct six-threshold sets use "E1", "C1", "I1", "I2", "C2", and "E2". Fuzzy indirect sets use "T1", "T2", and so on.
solution	Solution type to monitor. Accepted values are "all", "con" or "conservative", "par" or "parsimonious", and "int" or "intermediate".
include	Optional minimization include setting. Currently, this argument accepts only NULL, "", or "?".
which_M	Positive integer giving which solution alternative to use when minimization returns multiple models.
unit_step	Numeric step size used to move thresholds. Supply either one value applied to all tested sets, one value per condition when test.outcome = FALSE, or one value per c(conditions, outcome) when test.outcome = TRUE. If NULL, qcaERT computes scale-aware steps for the sets selected by test.conditions and test.outcome.

`unit_step_divisor`
: Positive number used to compute `unit_step` automatically when `unit_step = NULL`. The default is `10`.

`max_steps`
: Positive integer giving the maximum number of upward or downward threshold moves to attempt for each tested anchor.

`incl.cut`
: Inclusion cutoff passed to `QCA::truthTable()`.

`n.cut`
: Frequency cutoff passed to `QCA::truthTable()`. This must be an integer-like value of at least 1.

`dir.exp`
: Directional expectations used when the monitored solution is intermediate.

`i_mode`
: Character string controlling intermediate-solution selection. Accepted values are `"all"` and `"C1P1"`.

`exclude_mode`
: Character string controlling how excluded rows are handled for parsimonious and intermediate minimization. `"recompute"` recalculates exclusions from each perturbed truth table using `exclude_recompute`, `"static"` reuses `exclude_static`, and `"none"` does not use exclusions.

`exclude_recompute`
: Named list of arguments passed to `QCA::findRows()` when `exclude_mode = "recompute"`.

`exclude_static`
: Already computed exclusion object reused when `exclude_mode = "static"`.

`result_shape`
: Layout of the clean `results` table when `solution = "all"`. `"wide"` keeps one row per tested set-anchor-direction path with solution-type-specific columns such as `con_last_safe` and `par_reason`. `"long"` returns one row per tested set-anchor-direction path and solution type, with a `solution_type` column. Single-solution-type calls keep the compact single-solution-type layout.

`progress`
: Logical; if `TRUE` and the session is interactive, show a text progress bar.

`...`
: Additional arguments routed through the QCA workflow. Arguments matching `QCA::truthTable()` are forwarded to truth table construction; remaining minimization arguments are filtered and forwarded to `QCA::minimize()`. The function also looks in `...` for `include`, `dir.exp`, or `direxp` if those arguments were not supplied explicitly. In `print.calib_test()`, `...` is passed to `print.data.frame()`. In `as.data.frame.calib_test()`, `...` is ignored.

`x`
: A `calib_test` object returned by `calib.test()`.

`row.names`
: Logical; passed to `print.data.frame()` by `print.calib_test()`.

Details

The function rebuilds calibrated condition columns, and the outcome when `test.outcome = TRUE`, from `raw.data` using `calib_spec`. It then compares perturbed analyses against the baseline solution under the selected solution type.

Rows in `raw.data` and `calib.data` are paired before recalibration. Informative row names must be unique and identical in the same order on both inputs. If both inputs use only default row names (1, ..., n), pairing is positional and the inputs must not be independently reordered.

When `solution = "all"`, conservative, parsimonious, and, when requested, intermediate paths are searched independently for each tested set, anchor, and direction.

Each tested path starts from the supplied threshold for one method-specific anchor of one set and moves in one direction at a time (`"lower"` or `"upper"`), using `unit_step` until one of four things happens:

- the monitored solution changes,
- recalibration or minimization fails,
- the raw-data range or anchor ordering blocks the next move, or
- `max_steps` is reached.

If a proposed threshold is infeasible, that path ends with reason `"feasibility_boundary"`. The `first_failing` value records the rejected threshold, which is not evaluated for solution preservation.

If recalibration fails at a proposed threshold, that path ends with reason `"calibration_error"`. The result keeps the last safe threshold and records the threshold that failed. The diagnostics include QCA's error message.

The shared solution-control, exclusion, calibration-specification, and returned-object conventions are described in `?qcaERT_conventions`.

Value

An object of class `calib_test` with the following components:

`diagnostics` A detailed data frame with one row per tested anchor-direction path. When `solution = "all"`, each monitored solution type is searched independently, so diagnostics has one row per tested anchor, direction, and solution type. It includes the starting threshold, the last safe value, the first failing value, the number of successful steps, stop reason, and change classification.

`results` A compact data frame with the columns `set`, `role`, `raw`, `type`, `method`, `anchor`, `direction`, `start`, `last_safe`, `first_failing`, `step_unit`, `steps`, `total_delta`, `pct_raw_range`, and `reason`. When `solution = "all"` and `result_shape = "wide"`, the row unit remains set-anchor-direction, but the boundary and reason columns are solution-type-specific, using prefixes `con_`, `par_`, and, when available, `int_`. When `result_shape = "long"`, `results` has one row per tested path and solution type.

`bounds` A named list of compact lower/upper bound matrices, one per tested set. When `solution = "all"`, each tested set contains a named list of solution-type-specific lower/upper matrices.

`baseline` A list containing the baseline analysis status, selected solution terms used for comparison, metadata, and solution-type-specific baseline objects.

`by_set` A named list containing per-set step results, including path-level traces.

`settings` A list containing the analysis settings used to build the result object.

`print.calib_test()` prints a concise summary and a compact display table. In that printed table, `set` and `role` are omitted and the raw source column is labelled `condition`. `as.data.frame.calib_test()` returns the stored `results` table. If `ggplot2` is installed, `plot.calib_test()` provides interval, heatmap, and trace views; see `?qcaERT_plots`.

See Also

qcaERT_plots, `incl.test()`, `ncut.test()`, `loo.test()`, `subsample.test()`, `altset.test()`, `theory.test()`, `cluster.test()`, `sol.df()`

Examples

```
library(QCA)
library(ggplot2)
data(LR)

conditions <- c("DEV", "URB", "LIT", "IND", "STB")
outcome <- "SURV"
dir_exp <- rep("1", length(conditions))

thresholds <- list(
  DEV = findTh(LR$DEV, groups = 7),
  URB = findTh(LR$URB, groups = 4),
  LIT = findTh(LR$LIT, groups = 4),
  IND = findTh(LR$IND, groups = 4),
  STB = findTh(LR$STB, groups = 4),
  SURV = findTh(LR$SURV, groups = 4)
)

dat <- LR
dat$DEV <- calibrate(LR$DEV, type = "fuzzy", thresholds = thresholds$DEV)
dat$URB <- calibrate(LR$URB, type = "fuzzy", thresholds = thresholds$URB)
dat$LIT <- calibrate(LR$LIT, type = "fuzzy", thresholds = thresholds$LIT)
dat$IND <- calibrate(LR$IND, type = "fuzzy", thresholds = thresholds$IND)
dat$STB <- calibrate(LR$STB, type = "fuzzy", thresholds = thresholds$STB)
dat$SURV <- calibrate(LR$SURV, type = "fuzzy", thresholds = thresholds$SURV)

calib_spec <- list(
  DEV = list(raw = "DEV", type = "fuzzy", method = "direct", thresholds = thresholds$DEV),
  URB = list(raw = "URB", type = "fuzzy", method = "direct", thresholds = thresholds$URB),
  LIT = list(raw = "LIT", type = "fuzzy", method = "direct", thresholds = thresholds$LIT),
  IND = list(raw = "IND", type = "fuzzy", method = "direct", thresholds = thresholds$IND),
  STB = list(raw = "STB", type = "fuzzy", method = "direct", thresholds = thresholds$STB)
)

calib_spec_outcome <- calib_spec
calib_spec_outcome$SURV <- list(
  raw = "SURV",
  type = "fuzzy",
  method = "direct",
  thresholds = thresholds$SURV
)

# Common use: test selected calibrated conditions with scale-aware steps.
calib_out <- calib.test(
  raw.data = LR,
  calib.data = dat,
  outcome = outcome,
  conditions = conditions,
  calib_spec = calib_spec,
  test.conditions = c("DEV", "URB"),
  unit_step = NULL,
  unit_step_divisor = 10,
```

```
  max_steps = 5,
  incl.cut = 0.8,
  n.cut = 1,
  solution = "all",
  dir.exp = dir_exp,
  progress = TRUE
)

calib_out
as.data.frame(calib_out)
calib_out$bounds
calib_out$diagnostics

plot(calib_out, solution_type = "conservative")
plot(calib_out, solution_type = "conservative", type = "heatmap")

# Special case: test the calibrated outcome without putting it in
# conditions.
outcome_out <- calib.test(
  raw.data = LR,
  calib.data = dat,
  outcome = outcome,
  conditions = conditions,
  calib_spec = calib_spec_outcome,
  test.conditions = NULL,
  test.outcome = TRUE,
  unit_step = NULL,
  unit_step_divisor = 10,
  max_steps = 5,
  incl.cut = 0.8,
  n.cut = 1,
  solution = "all",
  dir.exp = dir_exp,
  progress = TRUE
)

outcome_out
as.data.frame(outcome_out)
plot(outcome_out, solution_type = "conservative")

# Special case: focus on selected anchors in a six-threshold direct
# calibration. For DEV, anchors are E1, C1, I1, I2, C2, and E2.
dev_anchor_out <- calib.test(
  raw.data = LR,
  calib.data = dat,
  outcome = outcome,
  conditions = conditions,
  calib_spec = calib_spec,
  test.conditions = "DEV",
  anchors_to_test = c("E1", "C1", "I1"),
  unit_step = NULL,
  unit_step_divisor = 10,
  max_steps = 5,
```

```
  incl.cut = 0.8,
  n.cut = 1,
  solution = "all",
  dir.exp = dir_exp,
  progress = TRUE
)

as.data.frame(dev_anchor_out)

plot(
  dev_anchor_out,
  solution_type = "conservative",
  type = "trace",
  set = "DEV",
  anchor = "E1",
  direction = "lower"
)

# Special case: indirect calibration. Indirect anchors are positional:
# T1, T2, and so on.
thresholds_indirect <- thresholds
thresholds_indirect$DEV <- findTh(LR$DEV, groups = 4)

dat_indirect <- dat
dat_indirect$DEV <- calibrate(
  LR$DEV,
  type = "fuzzy",
  method = "indirect",
  thresholds = thresholds_indirect$DEV
)

calib_spec_indirect <- calib_spec
calib_spec_indirect$DEV <- list(
  raw = "DEV",
  type = "fuzzy",
  method = "indirect",
  thresholds = thresholds_indirect$DEV
)

indirect_out <- calib.test(
  raw.data = LR,
  calib.data = dat_indirect,
  outcome = outcome,
  conditions = conditions,
  calib_spec = calib_spec_indirect,
  test.conditions = "DEV",
  anchors_to_test = c("T1", "T2"),
  unit_step = NULL,
  unit_step_divisor = 10,
  max_steps = 5,
  incl.cut = 0.8,
  n.cut = 1,
  solution = "all",
```

```
  dir.exp = dir_exp,
  progress = TRUE
)

as.data.frame(indirect_out)
```

`cluster.test` *Cluster heterogeneity diagnostics for QCA solutions*

Description

Evaluates how the consistency and coverage of selected QCA solution formulas and their prime implicants vary across clusters and, when repeated units are available, across units observed in more than one cluster. The function starts from an existing truth table, minimizes it under the requested solution type, extracts the selected baseline expressions, and then compares pooled fit values with cluster-specific and within-unit fit values.

Usage

```
cluster.test(
  data,
  tt,
  cluster_id,
  unit_id = NULL,
  solution = "all",
  include = NULL,
  dir.exp = NULL,
  exclude = NULL,
  which_M = 1,
  i_mode = c("all", "C1P1"),
  progress = TRUE,
  ...
)

## S3 method for class 'cluster_test'
print(x, row.names = FALSE, ...)

## S3 method for class 'cluster_test'
as.data.frame(x, ...)
```

Arguments

`data`	A data frame object containing the columns used to build `tt` plus the cluster and, optionally, unit identifiers. `data` must have the same number of rows as `tt$recoded.data`.

tt	A truth table object of class "QCA_tt", typically created with QCA::truthTable().
cluster_id	Name of the column in data identifying clusters. This must be a single non-empty character string, and the column must contain at least two distinct non-missing cluster values.
unit_id	Optional name of the column in data identifying units that may appear in more than one cluster. If NULL, within-unit diagnostics are not available. Rows with missing unit identifiers remain eligible for pooled and cluster-specific fit, but are excluded from within-unit diagnostics. Each non-missing unit-cluster combination must identify at most one row.
solution	Solution type to evaluate. Accepted values are "all", "con" or "conservative", "par" or "parsimonious", and "int" or "intermediate".
include	Optional minimization include setting. Currently, this argument accepts only NULL, "", or "?".
dir.exp	Directional expectations used when the monitored solution is intermediate. When solution = "all", supplying dir.exp adds intermediate solutions to the monitored set.
exclude	Optional exclusion specification passed to QCA::minimize() for parsimonious and intermediate minimization.
which_M	Positive integer giving which solution alternative to use when minimization returns multiple models.
i_mode	Character string controlling intermediate-solution selection. Accepted values are "all" and "C1P1".
progress	Logical; if TRUE and the session is interactive, show a text progress bar while minimizing the monitored solution types.
...	Additional arguments. In cluster.test(), these are filtered and forwarded to QCA::minimize(). The function also looks in ... for include, dir.exp or direxp, and exclude or omit if those arguments were not supplied explicitly. In print.cluster_test(), ... is passed to print.data.frame(). In as.data.frame.cluster_test(), ... is ignored.
x	A cluster_test object returned by cluster.test().
row.names	Logical; passed to print.data.frame() by print.cluster_test().

Details

The function recovers the outcome membership from tt$recoded.data, runs minimization for the requested solution type or solution types, extracts the selected baseline solution formulas, and computes pooled fit values for each complete formula and prime implicant.

Rows in data and tt$recoded.data are paired before cluster diagnostics. Informative row names must be unique and identical in the same order on both inputs. Default row names (1, ..., n) are treated as positional because QCA::truthTable() materializes them in tt$recoded.data; positionally paired inputs must not be independently reordered.

Pooled fit uses all original rows with non-missing configuration and outcome memberships. It does not require non-missing cluster or unit identifiers. Cluster-specific fit is computed directly from the eligible original rows belonging to each distinct non-missing cluster_id; missing unit_id values

do not remove rows from these calculations. Rows with missing `cluster_id` remain eligible for pooled fit but cannot contribute to cluster-specific or within-unit fit.

Within-unit diagnostics use a separate occurrence table formed from rows with non-missing `unit_id` and `cluster_id`. A unit is repeated when it is observed in more than one distinct cluster, regardless of its selected expression or outcome membership values. Each non-missing unit-cluster combination must occur at most once; duplicate combinations are rejected rather than having their fuzzy memberships summed. Within-unit fit uses the eligible membership pairs among each repeated unit's observed cluster occurrences.

Configuration extraction uses the data-frame `pims` component produced by `QCA::minimize()`. The shared solution-control, QCA solution-object, and returned-object conventions are described in `?qcaERT_conventions`.

Value

An object of class `cluster_test` with the following components:

`diagnostics` A detailed data frame with one row per selected solution formula. It records the solution type, internal configuration key, component count, status, pooled consistency and coverage, maximum and mean absolute cluster-level deltas, the clusters with the worst consistency and coverage values, whether within-unit diagnostics were available, the number of repeated units, and error information when a configuration could not be evaluated.

`results` A named list with three tables:

- `overview` One row per selected solution formula, summarizing pooled fit, maximum cluster deltas, worst clusters, and within-unit availability.
- `clusters` One row per expression-component-cluster combination, giving cluster-specific consistency, coverage, and deltas from the pooled expression values. The complete solution formula is stored as `component = "solution"`; separate term-level rows are included only for multi-term solutions.
- `units` One row per expression-component-unit combination for units observed in more than one cluster, giving within-unit consistency, coverage, and deltas from the pooled expression values. Separate term-level rows are included only for multi-term solutions.

`baseline` A list containing the input truth table, minimization results by solution type, selected solution terms used for comparison, metadata, and the extracted expression definitions used for the heterogeneity diagnostics.

`by_cluster` A named list of detailed cluster-level diagnostics for each selected solution formula.

`by_unit` A named list of detailed unit-level diagnostics for each selected solution formula when repeated units are available, or `NULL` otherwise.

`settings` A list containing the analysis settings used to build the result object.

`print.cluster_test()` prints a compact overview and, when the object has one displayed solution formula, cluster and within-unit details. `as.data.frame.cluster_test()` returns `results$overview`.

See Also

`calib.test()`, `incl.test()`, `ncut.test()`, `loo.test()`, `subsample.test()`, `altset.test()`, `theory.test()`, `sol.df()`

Examples

```
library(QCA)
data(LR)

conditions <- c("DEV", "URB", "LIT", "IND", "STB")
outcome <- "SURV"
dir_exp <- rep("1", length(conditions))

thresholds <- list(
  DEV = findTh(LR$DEV, groups = 7),
  URB = findTh(LR$URB, groups = 4),
  LIT = findTh(LR$LIT, groups = 4),
  IND = findTh(LR$IND, groups = 4),
  STB = findTh(LR$STB, groups = 4),
  SURV = findTh(LR$SURV, groups = 4)
)

dat <- LR
dat$DEV <- calibrate(LR$DEV, type = "fuzzy", thresholds = thresholds$DEV)
dat$URB <- calibrate(LR$URB, type = "fuzzy", thresholds = thresholds$URB)
dat$LIT <- calibrate(LR$LIT, type = "fuzzy", thresholds = thresholds$LIT)
dat$IND <- calibrate(LR$IND, type = "fuzzy", thresholds = thresholds$IND)
dat$STB <- calibrate(LR$STB, type = "fuzzy", thresholds = thresholds$STB)
dat$SURV <- calibrate(LR$SURV, type = "fuzzy", thresholds = thresholds$SURV)
dat$development_group <- ifelse(
  LR$DEV >= median(LR$DEV, na.rm = TRUE),
  "higher development",
  "lower development"
)
dat$case_id <- rownames(dat)

tt <- truthTable(
  data = dat,
  outcome = outcome,
  conditions = conditions,
  incl.cut = 0.8,
  n.cut = 1,
  complete = TRUE,
  show.cases = TRUE
)
enhanced <- findRows(tt, type = 2)

out <- cluster.test(
  data = dat,
  tt = tt,
  cluster_id = "development_group",
  unit_id = "case_id",
  solution = "intermediate",
  dir.exp = dir_exp,
  exclude = enhanced,
  which_M = 1,
  progress = TRUE
```

```
)

out
as.data.frame(out)
out$results$clusters
out$results$units
```

`incl.test` *Inclusion-cutoff robustness test for QCA solutions*

Description

Perturbs the truth table inclusion cutoff used in the analysis and checks when the monitored QCA solution changes. Starting from a baseline `incl.cut`, the function searches downward and upward in fixed steps and records the last value that preserved the baseline solution, the first value that was tested and changed the solution or triggered an error, and the reason the search stopped in each direction. An out-of-range proposal is not reported as a failing value because it is not analyzed.

Usage

```
incl.test(
  data,
  outcome,
  conditions = NULL,
  incl.cut = 1,
  step = 0.01,
  max_steps = 20,
  n.cut = 1,
  solution = "all",
  include = NULL,
  dir.exp = NULL,
  which_M = 1,
  i_mode = c("all", "C1P1"),
  exclude_mode = c("recompute", "static", "none"),
  exclude_recompute = list(type = 2),
  exclude_static = NULL,
  result_shape = c("wide", "long"),
  progress = TRUE,
  ...
)

## S3 method for class 'incl_test'
print(x, row.names = FALSE, ...)

## S3 method for class 'incl_test'
as.data.frame(x, ...)
```

Arguments

data	A non-empty data frame object containing the outcome and condition columns used in the QCA analysis.
outcome	Name of the outcome. This must be a single non-empty character string.
conditions	Optional character vector of condition names. If NULL, the condition set is left to QCA::truthTable().
incl.cut	Baseline inclusion cutoff passed to QCA::truthTable(). This must be a single finite number in [0, 1].
step	Positive numeric step size used to move incl.cut downward and upward from the baseline value.
max_steps	Positive integer giving the maximum number of stepwise moves to attempt in each direction.
n.cut	Frequency cutoff passed to QCA::truthTable().
solution	Solution type to monitor. Accepted values are "all", "con" or "conservative", "par" or "parsimonious", and "int" or "intermediate".
include	Optional minimization include setting. Currently, this argument accepts only NULL, "", or "?".
dir.exp	Directional expectations used when the monitored solution is intermediate.
which_M	Positive integer giving which solution alternative to use when minimization returns multiple models.
i_mode	Character string controlling intermediate-solution selection. Accepted values are "all" and "C1P1".
exclude_mode	Character string controlling how excluded rows are handled for parsimonious and intermediate minimization. "recompute" recalculates exclusions from each truth table, "static" reuses exclude_static, and "none" does not use exclusions.
exclude_recompute	Named list of arguments passed to QCA::findRows() when exclude_mode = "recompute".
exclude_static	Already computed exclusion object reused when exclude_mode = "static".
result_shape	Layout of the clean results table when solution = "all". "wide" keeps one row per direction with solution-type-specific columns such as con_last_safe and par_reason. "long" returns one row per direction and solution type, with a solution_type column. Single-solution-type calls keep the compact single-solution-type layout.
progress	Logical; if TRUE and the session is interactive, show a text progress bar.
...	Additional arguments routed through the QCA workflow. Arguments matching QCA::truthTable() are forwarded to truth table construction; remaining minimization arguments are filtered and forwarded to QCA::minimize(). The function also looks in ... for include, dir.exp, or direxp if those arguments were not supplied explicitly. In print.incl_test(), ... is passed to print.data.frame(). In as.data.frame.incl_test(), ... is ignored.
x	An incl_test object returned by incl.test().
row.names	Logical; passed to print.data.frame() by print.incl_test().

Details

The baseline analysis is built with `QCA::truthTable()` using the supplied `incl.cut` and `n.cut`, followed by minimization under the requested solution type. The function then tests lower and upper values of `incl.cut` one step at a time. When `solution = "all"`, conservative, parsimonious, and, when requested, intermediate paths are searched independently. Cutoffs are calculated from the baseline and step number. A cutoff that differs from 0 or 1 only by floating-point rounding is evaluated at that endpoint.

A directional search stops when one of the following occurs:

- the next proposed value falls outside `[0, 1]`, in which case it is not tested and `first_failing` remains missing,
- truth table construction fails,
- exclusion recomputation fails,
- minimization fails, or
- the monitored solution changes relative to the baseline.

The shared solution-control, exclusion, and returned-object conventions are described in `?qcaERT_conventions`.

Value

An object of class `incl_test` with the following components:

`diagnostics` A detailed data frame with one row for the lower search and one row for the upper search. When `solution = "all"`, each monitored solution type is searched independently, so diagnostics has one row per direction and solution type. It includes the baseline `incl.cut`, the last safe value, the first failing value, the number of successful steps, stop reason, change classification, and exclusion information for the baseline, last safe, and first failing runs.

`results` A compact data frame with the columns `direction`, `start`, `last_safe`, `first_failing`, `steps`, `total_delta`, and `reason`. When `solution = "all"` and `result_shape = "wide"`, the row unit remains `direction`, but the boundary and reason columns are solution-type-specific, using prefixes `con_`, `par_`, and, when available, `int_`. When `result_shape = "long"`, `results` has one row per direction and solution type.

`bounds` A named numeric vector with `Lower` and `Upper` elements taken from the last safe values in each search direction. When `solution = "all"`, this is a lower/upper matrix with one column per monitored solution type.

`baseline` A list containing the baseline truth table, minimization results, selected solution terms used for comparison, exclusion set used, and status information.

`by_direction` A named list with one entry for the lower search and one for the upper search, each containing the search trace and stopping information.

`settings` A list containing the analysis settings used to build the result object.

`print.incl_test()` prints a concise summary and the `results` table. `as.data.frame.incl_test()` returns the `results` table. If `ggplot2` is installed, `plot.incl_test()` provides interval and trace views; see `?qcaERT_plots`.

See Also

qcaERT_plots, calib.test(), ncut.test(), loo.test(), subsample.test(), altset.test(), theory.test(), cluster.test(), sol.df()

Examples

```
library(QCA)
data(LR)

conditions <- c("DEV", "URB", "LIT", "IND", "STB")
outcome <- "SURV"
dir_exp <- rep("1", length(conditions))

thresholds <- list(
  DEV = findTh(LR$DEV, groups = 7),
  URB = findTh(LR$URB, groups = 4),
  LIT = findTh(LR$LIT, groups = 4),
  IND = findTh(LR$IND, groups = 4),
  STB = findTh(LR$STB, groups = 4),
  SURV = findTh(LR$SURV, groups = 4)
)

dat <- LR
dat$DEV <- calibrate(LR$DEV, type = "fuzzy", thresholds = thresholds$DEV)
dat$URB <- calibrate(LR$URB, type = "fuzzy", thresholds = thresholds$URB)
dat$LIT <- calibrate(LR$LIT, type = "fuzzy", thresholds = thresholds$LIT)
dat$IND <- calibrate(LR$IND, type = "fuzzy", thresholds = thresholds$IND)
dat$STB <- calibrate(LR$STB, type = "fuzzy", thresholds = thresholds$STB)
dat$SURV <- calibrate(LR$SURV, type = "fuzzy", thresholds = thresholds$SURV)

out <- incl.test(
  data = dat,
  outcome = outcome,
  conditions = conditions,
  incl.cut = 0.8,
  step = 0.05,
  max_steps = 5,
  n.cut = 1,
  solution = "all",
  dir.exp = dir_exp,
  result_shape = "long",
  progress = TRUE
)

out
as.data.frame(out)
plot(out, solution_type = "conservative")
```

loo.test *Leave-one-out robustness test for QCA solutions*

Description

Deletes one case at a time, reruns the QCA analysis, and records whether the monitored solution changes, if selected fit measures change, and whether the reduced-data run ends with an error. The tested cases can be identified by row index or by case label.

Usage

```
loo.test(
  data,
  outcome,
  conditions = NULL,
  cases = NULL,
  case_labels = NULL,
  calib = c("fixed", "recompute"),
  raw.data = NULL,
  calib_spec = NULL,
  incl.cut = 1,
  n.cut = 1,
  solution = "all",
  include = NULL,
  dir.exp = NULL,
  which_M = 1,
  i_mode = c("all", "C1P1"),
  exclude_mode = c("recompute", "static", "none"),
  exclude_recompute = list(type = 2),
  exclude_static = NULL,
  fit_measures = c("inclS", "PRI", "covS"),
  fit_tol = 0,
  progress = TRUE,
  ...
)

## S3 method for class 'loo_test'
print(x, row.names = FALSE, ...)

## S3 method for class 'loo_test'
as.data.frame(x, ...)
```

Arguments

data	A data frame object containing the outcome and condition columns used in the QCA analysis. data must have at least two rows.
outcome	Name of the outcome. This must be a single non-empty character string.

conditions	Optional character vector of condition names. If NULL, the condition set is left to QCA::truthTable(). When calib = "recompute", conditions must be supplied explicitly.
cases	Cases to test. Use NULL to test all rows, a numeric vector of row indices, or a character vector of case labels.
case_labels	Optional character vector of case labels with length nrow(data). If NULL, the function uses rownames(data) when available; otherwise it uses row numbers converted to character.
calib	Calibration handling for reduced-data runs. "fixed" uses the calibrated values already present in data. "recompute" recalibrates the outcome and conditions after deleting each case, using raw.data and calib_spec.
raw.data	Raw-data frame object used when calib = "recompute". It must have the same number of rows as data.
calib_spec	Calibration specification used when calib = "recompute". This must be a named list keyed by the analysis sets c(outcome, conditions). Each entry must describe the raw source column, the set type, and the calibration inputs used to rebuild the calibrated set. Use the same calib_spec structure described for calib.test(). An entry may additionally contain findTh, a named list of QCA::findTh() arguments; when supplied, thresholds are re-estimated after each case deletion instead of reusing the baseline thresholds.
incl.cut	Inclusion cutoff passed to QCA::truthTable(). This must be a single finite number in [0, 1].
n.cut	Truth table frequency cutoff passed to QCA::truthTable(). This must be an integer-like value of at least 1.
solution	Solution type to monitor. Accepted values are "all", "con" or "conservative", "par" or "parsimonious", and "int" or "intermediate".
include	Optional minimization include setting. Currently, this argument accepts only NULL, "", or "?".
dir.exp	Directional expectations used when the monitored solution is intermediate.
which_M	Positive integer giving which solution alternative to use when minimization returns multiple models.
i_mode	Character string controlling intermediate-solution selection. Accepted values are "all" and "C1P1".
exclude_mode	Character string controlling how excluded rows are handled for parsimonious and intermediate minimization. "recompute" recalculates exclusions from each reduced truth table, "static" reuses exclude_static, and "none" does not use exclusions.
exclude_recompute	Named list of arguments passed to QCA::findRows() when exclude_mode = "recompute".
exclude_static	Already computed exclusion object reused when exclude_mode = "static".
fit_measures	Character vector of fit-measure names to compare between the baseline run and each reduced-data run. Use NULL to disable fit comparison.
fit_tol	Non-negative tolerance used when deciding whether fit values changed.

progress	Logical; if TRUE and the session is interactive, show a text progress bar.
...	Additional arguments. In loo.test(), named arguments matching QCA::truthTable() formals are passed to QCA::truthTable(), and the remaining named arguments are forwarded to QCA::minimize() after removing names reserved by loo.test(). The function also looks in ... for include, dir.exp, or direxp if those arguments were not supplied explicitly. In print.loo_test(), ... is passed to print.data.frame(). In as.data.frame.loo_test(), ... is ignored.
x	A loo_test object returned by loo.test().
row.names	Logical; passed to print.data.frame() by print.loo_test().

Details

The function first runs the baseline analysis on the full dataset, then removes one selected case at a time and reruns the analysis on the reduced data.

When calib = "fixed", the reduced-data runs use the calibrated values already present in data. When calib = "recompute", the function rebuilds the calibrated outcome and condition columns after each case deletion using QCA::findTh() and QCA::calibrate() as specified in calib_spec.

With calib = "recompute", rows in raw.data and data are paired before deletion and recalibration. Informative row names must be unique and identical in the same order on both inputs. If data has only default row names and case_labels is supplied, raw.data must use those labels as unique row names in the same order. If neither input has identifiable labels, pairing is positional and the inputs must not be independently reordered.

For each tested case, the function records whether the monitored solution changed, whether monitored fit measures changed beyond fit_tol, and whether the reduced-data run stopped with a calibration, truth table, exclusion, or minimization error.

The shared solution-control, exclusion, and returned-object conventions are described in ?qcaERT_conventions.

Value

An object of class loo_test with the following components:

diagnostics A detailed data frame with one row per tested case. It records the tested row index, case label, run status, solution-change classification, fit-change classification, the number of changed fit measures, the largest absolute fit change, and error information when a reduced-data run fails.

results A compact data frame with the columns row_index, case_label, status, solution_change, fit_changed_types, n_fit_deltas, and max_abs_fit_delta. If fit comparison is requested, n_fit_deltas is NA when no fit measures can be compared between the selected models.

baseline A list containing the baseline analysis built from the full dataset, including the truth table, minimization output, selected solution terms used for comparison, fit values, exclusion information, and calibration information.

by_case A named list with one entry per tested case. Each entry stores the baseline run, the reduced-data run when available, change summaries, fit deltas, exclusion information, calibration information, and any error information for that case.

settings A list containing the analysis settings used to build the result object.

`print.loo_test()` prints a concise summary and the `results` table. `as.data.frame.loo_test()` returns the `results` table.

See Also

`calib.test()`, `incl.test()`, `ncut.test()`, `subsample.test()`, `altset.test()`, `theory.test()`, `cluster.test()`, `sol.df()`

Examples

```
library(QCA)
data(LR)

conditions <- c("DEV", "URB", "LIT", "IND", "STB")
outcome <- "SURV"
dir_exp <- rep("1", length(conditions))

thresholds <- list(
  DEV = findTh(LR$DEV, groups = 7),
  URB = findTh(LR$URB, groups = 4),
  LIT = findTh(LR$LIT, groups = 4),
  IND = findTh(LR$IND, groups = 4),
  STB = findTh(LR$STB, groups = 4),
  SURV = findTh(LR$SURV, groups = 4)
)

dat <- LR
dat$DEV <- calibrate(LR$DEV, type = "fuzzy", thresholds = thresholds$DEV)
dat$URB <- calibrate(LR$URB, type = "fuzzy", thresholds = thresholds$URB)
dat$LIT <- calibrate(LR$LIT, type = "fuzzy", thresholds = thresholds$LIT)
dat$IND <- calibrate(LR$IND, type = "fuzzy", thresholds = thresholds$IND)
dat$STB <- calibrate(LR$STB, type = "fuzzy", thresholds = thresholds$STB)
dat$SURV <- calibrate(LR$SURV, type = "fuzzy", thresholds = thresholds$SURV)

out <- loo.test(
  data = dat,
  outcome = outcome,
  conditions = conditions,
  cases = seq_len(nrow(dat)),
  case_labels = rownames(dat),
  calib = "fixed",
  incl.cut = 0.8,
  n.cut = 1,
  solution = "all",
  dir.exp = dir_exp,
  fit_measures = c("inclS", "PRI", "covS"),
  progress = TRUE
)

out
as.data.frame(out)
```

`ncut.test` *Truth table frequency-cutoff robustness test for QCA solutions*

Description

Perturbs the truth table frequency cutoff used in the analysis and checks when the monitored QCA solution changes. Starting from a baseline `n.cut`, the function searches downward and upward in fixed integer steps and records the last value that preserved the baseline solution, the first value that was tested and changed the solution or triggered an error, and the reason the search stopped in each direction. An out-of-range proposal is not reported as a failing value because it is not analyzed.

Usage

```
ncut.test(
  data,
  outcome,
  conditions = NULL,
  n.cut = 1,
  step = 1,
  max_steps = 20,
  incl.cut = 1,
  solution = "all",
  include = NULL,
  dir.exp = NULL,
  which_M = 1,
  i_mode = c("all", "C1P1"),
  exclude_mode = c("recompute", "static", "none"),
  exclude_recompute = list(type = 2),
  exclude_static = NULL,
  result_shape = c("wide", "long"),
  progress = TRUE,
  ...
)

## S3 method for class 'ncut_test'
print(x, row.names = FALSE, ...)

## S3 method for class 'ncut_test'
as.data.frame(x, ...)
```

Arguments

`data`	A non-empty data frame containing the outcome and condition columns used in the QCA analysis.
`outcome`	Name of the outcome. This must be a single non-empty character string.
`conditions`	Optional character vector of condition names. If `NULL`, the condition set is left to `QCA::truthTable()`.

n.cut	Baseline truth table frequency cutoff passed to QCA::truthTable(). This must be an integer-like value from 1 through the number of rows in data.
step	Positive integer step size used to move n.cut downward and upward from the baseline value.
max_steps	Maximum number of stepwise moves to attempt in each direction.
incl.cut	Inclusion cutoff passed to QCA::truthTable(). This must be a single finite number in [0, 1].
solution	Solution type to monitor. Accepted values are "all", "con" or "conservative", "par" or "parsimonious", and "int" or "intermediate".
include	Optional minimization include setting. Currently, this argument accepts only NULL, "", or "?".
dir.exp	Directional expectations used when the monitored solution is intermediate.
which_M	Positive integer giving which solution alternative to use when minimization returns multiple models.
i_mode	Character string controlling intermediate-solution selection. Accepted values are "all" and "C1P1".
exclude_mode	Character string controlling how excluded rows are handled for parsimonious and intermediate minimization. "recompute" recalculates exclusions from each truth table, "static" reuses exclude_static, and "none" does not use exclusions.
exclude_recompute	Named list of arguments passed to QCA::findRows() when exclude_mode = "recompute".
exclude_static	Already computed exclusion object reused when exclude_mode = "static".
result_shape	Layout of the clean results table when solution = "all". "wide" keeps one row per direction with solution-type-specific columns such as con_last_safe and par_reason. "long" returns one row per direction and solution type, with a solution_type column. Single-solution-type calls keep the compact single-solution-type layout.
progress	Logical; if TRUE and the session is interactive, show a text progress bar.
...	Additional arguments routed through the QCA workflow. Arguments matching QCA::truthTable() are forwarded to truth table construction; remaining minimization arguments are filtered and forwarded to QCA::minimize(). The function also looks in ... for include, dir.exp, or direxp if those arguments were not supplied explicitly. In print.ncut_test(), ... is passed to print.data.frame(). In as.data.frame.ncut_test(), ... is ignored.
x	An ncut_test object returned by ncut.test().
row.names	Logical; passed to print.data.frame() by print.ncut_test().

Details

The baseline analysis is built with QCA::truthTable() using the supplied incl.cut and n.cut, followed by minimization under the requested solution type. The function then tests lower and

upper values of `n.cut` one step at a time. When `solution = "all"`, conservative, parsimonious, and, when requested, intermediate paths are searched independently.

The search range is bounded below by `1` and above by the number of rows in `data`.

A directional search stops when one of the following occurs:

- the next proposed value falls outside the allowed range, in which case it is not tested and `first_failing` remains missing,
- truth table construction fails,
- exclusion recomputation fails,
- minimization fails, or
- the monitored solution changes relative to the baseline.

The shared solution-control, exclusion, and returned-object conventions are described in `?qcaERT_conventions`.

Value

An object of class `ncut_test` with the following components:

`diagnostics` A detailed data frame with one row for the lower search and one row for the upper search. When `solution = "all"`, each monitored solution type is searched independently, so diagnostics has one row per direction and solution type. It includes the baseline `n.cut`, the last safe value, the first failing value, the number of successful steps, stop reason, change classification, exclusion information for the baseline, last safe, and first failing runs, and the computed upper limit.

`results` A compact data frame with the columns `direction`, `start`, `last_safe`, `first_failing`, `steps`, `total_delta`, and `reason`. When `solution = "all"` and `result_shape = "wide"`, the row unit remains `direction`, but the boundary and reason columns are solution-type-specific, using prefixes `con_`, `par_`, and, when available, `int_`. When `result_shape = "long"`, `results` has one row per direction and solution type.

`bounds` A named integer vector with `Lower` and `Upper` elements taken from the last safe values in each search direction. When `solution = "all"`, this is a lower/upper matrix with one column per monitored solution type.

`baseline` A list containing the baseline truth table, minimization results, selected solution terms used for comparison, exclusion set used, and status information.

`by_direction` A named list with one entry for the lower search and one for the upper search, each containing the search trace and stopping information.

`settings` A list containing the analysis settings used to build the result object, including the computed upper limit.

`print.ncut_test()` prints a concise summary and the `results` table. `as.data.frame.ncut_test()` returns the `results` table.

See Also

`calib.test()`, `incl.test()`, `loo.test()`, `subsample.test()`, `altset.test()`, `theory.test()`, `cluster.test()`, `sol.df()`

Examples

```
library(QCA)
data(LR)

conditions <- c("DEV", "URB", "LIT", "IND", "STB")
outcome <- "SURV"
dir_exp <- rep("1", length(conditions))

thresholds <- list(
  DEV = findTh(LR$DEV, groups = 7),
  URB = findTh(LR$URB, groups = 4),
  LIT = findTh(LR$LIT, groups = 4),
  IND = findTh(LR$IND, groups = 4),
  STB = findTh(LR$STB, groups = 4),
  SURV = findTh(LR$SURV, groups = 4)
)

dat <- LR
dat$DEV <- calibrate(LR$DEV, type = "fuzzy", thresholds = thresholds$DEV)
dat$URB <- calibrate(LR$URB, type = "fuzzy", thresholds = thresholds$URB)
dat$LIT <- calibrate(LR$LIT, type = "fuzzy", thresholds = thresholds$LIT)
dat$IND <- calibrate(LR$IND, type = "fuzzy", thresholds = thresholds$IND)
dat$STB <- calibrate(LR$STB, type = "fuzzy", thresholds = thresholds$STB)
dat$SURV <- calibrate(LR$SURV, type = "fuzzy", thresholds = thresholds$SURV)

out <- ncut.test(
  data = dat,
  outcome = outcome,
  conditions = conditions,
  n.cut = 1,
  step = 1,
  max_steps = 4,
  incl.cut = 0.8,
  solution = "all",
  dir.exp = dir_exp,
  result_shape = "long",
  progress = TRUE
)

out
as.data.frame(out)
```

---

qcaERT_conventions *qcaERT argument and output conventions*

Description

The qcaERT robustness functions are siblings. They share a common public structure for solution controls, exclusion handling, returned objects, printing, and data-frame coercion. Individual

function pages describe the function-specific workflow; this page describes the package-wide conventions that should be read consistently across the function family.

Solution Controls

Most functions accept `solution`, `include`, `dir.exp`, `which_M`, and `i_mode`.

`solution` may be `"all"`, `"con"`/`"conservative"`, `"par"`/`"parsimonious"`, or `"int"`/`"intermediate"`. For single-solution-type calls, `include` is optional and is resolved from `solution`: conservative uses `include = ""`, while parsimonious and intermediate use `include = "?"`. Intermediate solutions require `dir.exp`.

When `solution = "all"`, do not supply `include`. The monitored set contains conservative and parsimonious solutions by default. If `dir.exp` is supplied, the intermediate solution is also monitored. In stepwise boundary searches such as `incl.test()`, `ncut.test()`, and `calib.test()`, these solution types are searched independently; one solution type changing does not stop the search for another solution type. When a baseline or step fails in one monitored solution type, inspect the solution-type-specific rows in `diagnostics`, `results`, or supporting `by_*` components before interpreting an all-solution run as a single combined failure.

`which_M` selects the model or solution alternative to use when QCA minimization returns multiple alternatives. `i_mode` controls which intermediate branches are included; accepted values are `"all"` and `"C1P1"`. Most robustness functions default to allowing all intermediate branches. `theory.test()` defaults to `"C1P1"` because comparative theory testing needs one comparable intermediate branch per theory by default.

Outcome Negation

For tests that build truth tables, `neg.out = TRUE` requests the negation of a single outcome set. qcaERT passes the negated outcome to QCA using a leading `~`. Supply one form of negation per call.

Use the original column name with `neg.out = TRUE` when the test looks up that column for calibration or stratification. `settings$outcome` records the outcome used to build the truth tables.

Exclusions

Functions that can monitor parsimonious or intermediate solutions use a common exclusion convention. `exclude_mode = "recompute"` recalculates excluded rows for each perturbed, reduced, or sampled truth table using `QCA::findRows()` and `exclude_recompute`. `exclude_mode = "static"` reuses the supplied `exclude_static` object. `exclude_mode = "none"` avoids exclusion handling.

The family default for recomputation is `exclude_recompute = list(type = 2)`. `incl.test()`, `ncut.test()`, `calib.test()`, `loo.test()`, `subsample.test()`, `altset.test()`, and `theory.test()` share this `exclude_mode`, `exclude_recompute`, and `exclude_static` convention. `cluster.test()` starts from an existing truth table and therefore accepts `exclude` directly instead of exposing `exclude_mode`.

Calibration Specifications

Calibration-family functions use `calib_spec` for calibration information. A `calib_spec` entry is named by calibrated set, contains the raw source name in `raw`, the calibration `type`, and `thresholds`, and may contain `method` and a `calibrate` list forwarded to `QCA::calibrate()`.

For condition-only calibration tests, `calib_spec` is named by `conditions`. When `calib.test()` or `altset.test()` is called with `test.outcome = TRUE`, `calib_spec` is named by `c(conditions, outcome)`. The outcome must not be included in `conditions`; outcome calibration testing is requested explicitly with `test.outcome = TRUE`.

Crisp conditions use one threshold. Fuzzy direct calibration supports three thresholds in `c(E, C, I)` order or six thresholds in `c(E1, C1, I1, I2, C2, E2)` order. Fuzzy indirect calibration treats its thresholds as ordered cutpoints.

These calibration-perturbation routines are designed for QCA workflows using crisp and fuzzy sets. They are not multi-value calibration tools. In `loo.test()` and `subsample.test()` with `calib = "recompute"`, a `calib_spec` entry may additionally contain `findTh`, a named list of `QCA::findTh()` arguments, to re-estimate thresholds on each reduced or subsampled raw dataset. Without `findTh`, the baseline `thresholds` stored in `calib_spec` are reused.

Result Objects

Most robustness functions return an S3 object with `diagnostics`, `results`, and `settings`, plus supporting components such as `baseline`, `bounds`, `by_direction`, `by_case`, `by_run`, `by_draw`, or `summary`.

`diagnostics` is the detailed internal table. `results` is the clean table. The `print()` method shows a concise summary plus the results, and `as.data.frame()` returns the clean `results` table.

For `incl.test()`, `ncut.test()`, and `calib.test()`, `result_shape` controls the layout of the clean `results` table when `solution = "all"`. The default `"wide"` layout keeps one row per tested path and uses solution-type-prefixed columns such as `con_last_safe` and `par_reason`. The `"long"` layout uses one row per tested path and solution type, with a `solution_type` column. The raw `diagnostics` table is unchanged.

`cluster.test()` and `theory.test()` use structured `results` lists because their clean output has more than one natural table. `cluster.test()` returns `overview`, `clusters`, and `units`; `as.data.frame()` returns `results$overview`. `theory.test()` returns `models`, `pairwise`, and `solutions`; `as.data.frame()` returns `results$models`.

Progress and Reproducibility

Functions using the shared progress display show a varying introductory message and a text progress bar only in interactive sessions when `progress = TRUE`. Message selection uses private cosmetic state initialized from wall-clock time and the R process identifier, then advanced with an internal invocation counter. It does not call R's random-number generator, create or modify `.Random.seed`, or alter the sampling algorithms and seed arguments used by `altset.test()` or `subsample.test()`.

This separation is intentional: changing a display option must not change the random analytical specifications or subsamples produced from a fixed seed. The displayed message itself is cosmetic and is not intended to be reproducible; messages may vary or repeat across calls while analytical results remain unchanged.

Plotting

Plotting requires `ggplot2`. `plot()` methods are provided for `incl_test`, `calib_test`, and `theory_test` objects. Interval and trace views map directly to stored boundary-search diagnostics; theory plots map selected theory-specific models into consistency/coverage space. `sol.chart()` renders `sol.df()` tables as solution charts. See `?qcaERT_plots` and `?sol.chart`.

QCA Solution Objects

`cluster.test()` and `sol.df()` consume QCA minimization objects. `sol.df()` extracts a compact data frame, and `sol.chart()` gives a visual display of that extracted table. Case and solution-expression reconstruction use the data-frame `pims` component produced by `QCA::minimize()`.

`qcaERT_plots` *Plot and chart qcaERT results*

Description

Plot methods provide visual inspection helpers for qcaERT result objects. They are simplified views of the existing result object: the detailed `diagnostics` table supplies the interval and heatmap views, while the path-level trace components supply trace views. `sol.chart()` provides a matching visual display for `sol.df()` solution tables.

Usage

```
## S3 method for class 'calib_test'
plot(
  x,
  type = c("interval", "heatmap", "trace"),
  sets = NULL,
  roles = NULL,
  anchors = NULL,
  directions = c("lower", "upper"),
  stop_reason = NULL,
  changed_types = NULL,
  solution_type = NULL,
  solution = NULL,
  monitored_solutions = NULL,
  metric = c("raw", "pct", "steps"),
  value = c("delta", "last_safe", "failing"),
  abs_delta = FALSE,
  cell = c("anchor_direction", "anchor"),
  show_text = FALSE,
  set = NULL,
  anchor = NULL,
  direction = NULL,
  show_stop = TRUE,
  order_sets = c("input", "most_sensitive", "least_sensitive"),
  legend = TRUE,
  theme = .plot_theme(),
  ...
)

## S3 method for class 'incl_test'
```

```
plot(
  x,
  type = c("interval", "trace"),
  directions = c("lower", "upper"),
  stop_reason = NULL,
  changed_types = NULL,
  solution_type = NULL,
  solution = NULL,
  monitored_solutions = NULL,
  i_mode = NULL,
  direction = NULL,
  show_stop = TRUE,
  legend = TRUE,
  theme = .plot_theme(),
  ...
)

## S3 method for class 'theory_test'
plot(
  x,
  solution_type = NULL,
  intermediate_branch = NULL,
  show_labels = TRUE,
  label_line = TRUE,
  label_line_alpha = 0.45,
  label_line_width = 0.35,
  point_size = 3.5,
  text_size = 3.5,
  label_nudge_x = 0.008,
  label_nudge_y = 0.006,
  legend = TRUE,
  theme = .plot_theme(),
  ...
)
```

Arguments

x	An incl_test object returned by incl.test(), a calib_test object returned by calib.test(), or a theory_test object returned by theory.test().
type	Plot type. incl_test supports "interval" and "trace". calib_test supports "interval", "heatmap", and "trace". theory_test has one consistency-coverage plot and does not use type.
sets	For calib_test, optional character vector selecting calibrated sets.
roles	For calib_test, optional character vector selecting set roles. Supported values are "condition" and "outcome".
anchors	For calib_test, optional character vector selecting calibration anchors. Supported anchors are taken from the result object and may include crisp ("T"),

fuzzy direct three-threshold ("E", "C", "I"), fuzzy direct six-threshold ("E1", "C1", "I1", "I2", "C2", "E2"), or fuzzy indirect ("T1", "T2", ...) anchors.

directions — Character vector selecting lower and/or upper searches.

stop_reason — Optional character vector used to filter diagnostic rows by stop reason before plotting.

changed_types — Optional solution-type filter applied to comma-coded changed_types diagnostic values. Accepted tokens follow the common qcaERT solution-type conventions: "con"/"conservative", "par"/"parsimonious", and "int"/"intermediate".

solution_type — Solution type to plot. Required when more than one solution type is present. Accepted values follow the common qcaERT solution-type conventions, excluding "all".

solution — Reserved. Plot methods do not accept solution; use solution_type.

monitored_solutions — Reserved. Plot methods do not accept monitored_solutions; use solution_type.

metric — For calib_test heatmaps, the quantity used for the fill color: "raw", "pct", or "steps".

value — For calib_test heatmaps with metric = "raw", the raw threshold value to show: "delta", "last_safe", or "failing".

abs_delta — Logical. If TRUE, heatmaps using raw or percentage deltas display absolute values.

cell — For calib_test heatmaps, whether cells are split by anchor-direction path ("anchor_direction") or by anchor only ("anchor").

show_text — Logical. If TRUE, add rounded heatmap values as text.

set — For calib_test trace plots, the single calibrated set to plot.

anchor — For calib_test trace plots, the single calibration anchor to plot.

direction — For type = "trace", the single search direction to plot: "lower" or "upper".

show_stop — Logical. If TRUE, draw reference lines for baseline and, when present, first-failing values. In a calibration trace, an infeasible threshold is labelled at the reference line and omitted from the trace.

order_sets — For calib_test, set ordering in interval and heatmap views. "input" preserves result order; "most_sensitive" and "least_sensitive" order by the smallest solution-changing percentage delta when available.

legend — Logical. If FALSE, hide the plot legend.

theme — A ggplot2 theme object added to the plot.

... — Additional graphical arguments forwarded to the primary ggplot2 geometry.

i_mode — For incl_test, optional filter applied to the i_mode diagnostic column. Accepted values are "all" and "C1P1".

intermediate_branch — For theory_test intermediate plots, the intermediate branch to plot when more than one branch is present.

show_labels — For theory_test, logical. If TRUE, print theory names next to their consistency/coverage points.

label_line — For theory_test, logical. If TRUE, draw a line from each consistency/coverage point to its theory-name label.

label_line_alpha — For theory_test, transparency of label connector lines.

label_line_width — For theory_test, width of label connector lines.

point_size — For theory_test, point size.

text_size — For theory_test, theory-name label size.

label_nudge_x — For theory_test, horizontal nudge for theory-name labels.

label_nudge_y — For theory_test, vertical nudge for theory-name labels.

Details

Plotting is optional. The qcaERT analysis functions do not require ggplot2, but these methods do.

Value

A ggplot object.

Plot Types

For incl_test, "interval" shows the baseline inclusion cutoff and the lower/upper last-safe values. "trace" shows the stepwise path for one search direction.

For calib_test, "interval" shows baseline and lower/upper last-safe threshold values by set and anchor. "heatmap" summarizes sensitivity across anchor paths. "trace" shows the stepwise path for one set, anchor, and direction.

For theory_test, the plot compares theory-specific selected models in consistency/coverage space for one selected solution type. Theory names are shown near the points, and the legend pairs each theory with its selected solution terms.

For sol.df() tables, sol.chart() draws a table-like chart of sufficient configurations, using filled and open circles for condition presence and absence.

Family Consistency

These methods follow the common qcaERT output conventions described in ?qcaERT_conventions: plots read from diagnostics and path-level supporting components, while print() and as.data.frame() keep their existing concise and tabular behavior. sol.chart() follows the same division of labor by reading from the already-extracted sol.df() table.

Examples

```
library(QCA)
library(ggplot2)

data(LR)

conditions <- c("DEV", "URB", "LIT", "IND", "STB")
outcome <- "SURV"
```

```
dir_exp <- rep("1", length(conditions))

thresholds <- list(
  DEV = findTh(LR$DEV, groups = 7),
  URB = findTh(LR$URB, groups = 4),
  LIT = findTh(LR$LIT, groups = 4),
  IND = findTh(LR$IND, groups = 4),
  STB = findTh(LR$STB, groups = 4),
  SURV = findTh(LR$SURV, groups = 4)
)

dat <- LR
dat$DEV <- calibrate(LR$DEV, type = "fuzzy", thresholds = thresholds$DEV)
dat$URB <- calibrate(LR$URB, type = "fuzzy", thresholds = thresholds$URB)
dat$LIT <- calibrate(LR$LIT, type = "fuzzy", thresholds = thresholds$LIT)
dat$IND <- calibrate(LR$IND, type = "fuzzy", thresholds = thresholds$IND)
dat$STB <- calibrate(LR$STB, type = "fuzzy", thresholds = thresholds$STB)
dat$SURV <- calibrate(LR$SURV, type = "fuzzy", thresholds = thresholds$SURV)

incl_out <- incl.test(
  data = dat,
  outcome = outcome,
  conditions = conditions,
  incl.cut = 0.8,
  step = 0.05,
  max_steps = 5,
  n.cut = 1,
  solution = "all",
  dir.exp = dir_exp,
  progress = TRUE
)

calib_spec <- list(
  DEV = list(raw = "DEV", type = "fuzzy", method = "direct", thresholds = thresholds$DEV),
  URB = list(raw = "URB", type = "fuzzy", method = "direct", thresholds = thresholds$URB),
  LIT = list(raw = "LIT", type = "fuzzy", method = "direct", thresholds = thresholds$LIT),
  IND = list(raw = "IND", type = "fuzzy", method = "direct", thresholds = thresholds$IND),
  STB = list(raw = "STB", type = "fuzzy", method = "direct", thresholds = thresholds$STB)
)

calib_out <- calib.test(
  raw.data = LR,
  calib.data = dat,
  outcome = outcome,
  conditions = conditions,
  calib_spec = calib_spec,
  test.conditions = c("DEV", "URB"),
  unit_step = NULL,
  unit_step_divisor = 10,
  max_steps = 5,
  incl.cut = 0.8,
  n.cut = 1,
  solution = "all",
```

```
  dir.exp = dir_exp,
  progress = TRUE
)

theories <- list(
  development = c("DEV", "URB", "LIT"),
  industrial = c("DEV", "URB", "IND"),
  broad = c("DEV", "URB", "LIT", "IND", "STB")
)

theory_out <- theory.test(
  data = dat,
  outcome = outcome,
  theories = theories,
  incl.cut = 0.8,
  n.cut = 1,
  solution = "all",
  dir.exp = list(
    development = c("1", "1", "1"),
    industrial = c("1", "1", "1"),
    broad = c("1", "1", "1", "1", "1")
  ),
  progress = TRUE
)

plot(incl_out, solution_type = "conservative")
plot(incl_out, solution_type = "conservative", type = "trace", direction = "lower")
plot(calib_out, solution_type = "conservative")
plot(calib_out, solution_type = "conservative", type = "heatmap")
plot(
  calib_out,
  solution_type = "conservative",
  type = "trace",
  set = "DEV",
  anchor = "E1",
  direction = "lower"
)
plot(theory_out, solution_type = "conservative")
```

---

qcaERT_tests *Choose a qcaERT robustness tool*

Description

This page maps common QCA robustness tests to their respective functions

Calibration

Use `calib.test()` when you want to know whether a QCA solution is sensitive to calibration thresholds. It perturbs one calibration anchor at a time and reports how far each threshold can move before the monitored solution changes or the search reaches a boundary.

You can also use `altset.test()` when you want calibration perturbations to be combined with other perturbations in sampled alternative analysis settings, such as alternative inclusion cutoffs or frequency cutoffs.

Inclusion or n.cut

Use `incl.test()` to check the inclusion (consistency) cutoff for sufficiency. It searches below and above the baseline `incl.cut` and records the last safe cutoff and, when an admissible cutoff is tested and fails, the first failing cutoff in each direction.

Use `ncut.test()` when the concern is the minimum number of cases under which a truth table row is declared as a remainder. It searches lower and upper `n.cut` values and records when the monitored solution changes or the feasible boundary is reached.

Cases

Use `loo.test()` when you suspect individual cases may drive the solution. It removes selected cases one at a time and compares each reduced analysis with the baseline.

Use `subsample.test()` when you want repeated partial-sample checks. It draws subsamples, rebuilds the QCA analysis, and summarizes how often the baseline solution is preserved across runs. This is a stringent, punishing robustness check and is most useful when sample-composition sensitivity is substantively important.

Groups or clusters

Use `cluster.test()` when you expect heterogeneity across clusters, groups, or repeated units. It compares the consistency and coverage of selected sufficient solution formulas and prime implicants across clusters and, when requested, repeated units.

Theory Specifications

Use `theory.test()` when you want to compare several theoretically motivated condition sets under the same outcome, truth-table cutoffs, solution type, exclusion handling, and settings for `which_M` and `i_mode`.

Reporting QCA Solutions

Use `sol.df()` when you want QCA minimization objects turned into a compact, data frame. Use `sol.chart()` when you want that table rendered as a visual chart of prime implicants.

How To Read The Results

Most qcaERT robustness functions return `diagnostics`, `results`, and `settings`. `diagnostics` is the detailed table; `results` is the clean table returned by `as.data.frame()`. `print()` shows a concise summary. The shared output structure is described in `?qcaERT_conventions`.

For `calib.test()`, `incl.test()`, and `theory.test()` objects, `plot()` methods provide optional visual inspection helpers when `ggplot2` is installed. `sol.chart()` provides the corresponding visual presentation for `sol.df()` tables. See `?qcaERT_plots`.

Examples

```
?qcaERT_tests
?qcaERT_conventions
?qcaERT_plots
```

`sol.chart` *Draw a chart from a sol.df table*

Description

Turns the solution table returned by `sol.df()` into a visual chart of sufficient configurations. The chart uses one column per aligned prime implicant and shows condition presence, condition absence, prime implicant fit, solution fit, and optionally cases.

Usage

```
sol.chart(
  x,
  conditions = NULL,
  solution_types = c("conservative", "intermediate", "parsimonious"),
  model = NULL,
  intermediate_branch = NULL,
  colors = c(conservative = "#222222", intermediate = "#E69F00", parsimonious =
    "#0072B2"),
  show_cases = TRUE,
  show_pi_fit = TRUE,
  show_solution_fit = TRUE,
  digits = 2,
  title = NULL,
  note = TRUE,
  legend = FALSE,
  point_size = 3.2,
  text_size = 3.3,
  theme = .plot_theme()
)
```

Arguments

`x`	A data frame returned by `sol.df()`.
`conditions`	Optional character vector giving the condition order. If `NULL`, conditions are inferred from `Prime_Implicants` in order of first appearance.

`solution_types`	Solution types to display, in plotting order. Accepted values follow the common qcaERT solution-type conventions, excluding `"all"`. `"complex"` is accepted as an alias for `"conservative"` for display use.
`model`	Positive integer selecting the numbered model to display. Required when `x` contains more than one model number. Rows without a model number are retained alongside the selected model.
`intermediate_branch`	Optional intermediate branch to display when `x` contains more than one intermediate branch.
`colors`	Named character vector with colors for `"conservative"`, `"intermediate"`, and `"parsimonious"`.
`show_cases`	Logical; if `TRUE`, include a cases row. Cases are taken from the most detailed available configuration in each column.
`show_pi_fit`	Logical; if `TRUE`, include prime-implicant consistency, raw coverage, unique coverage, and PRI rows.
`show_solution_fit`	Logical; if `TRUE`, include solution-level consistency, coverage, and PRI rows.
`digits`	Non-negative integer used to format fit statistics.
`title`	Optional plot title. If `NULL`, a default title is used.
`note`	Logical; if `TRUE`, include a caption explaining symbols and colors.
`legend`	Logical; if `TRUE`, include ggplot legends for solution type and condition state.
`point_size`	Size of presence/absence symbols.
`text_size`	Size of table text.
`theme`	A ggplot2 theme object added to the chart.

Details

`sol.chart()` is a visual display for `sol.df()` tables. It does not extract solutions from QCA minimization objects. Use `sol.df()` first, then pass the resulting table to `sol.chart()`.

Prime implicant columns are aligned according to the following hierarchy: conservative/complex, intermediate, and parsimonious. When several detailed configurations simplify into the same intermediate or parsimonious prime implicant, the simpler prime implicant is repeated across the relevant columns so that the detailed empirical configurations remain visible.

Filled circles denote condition presence. Open circles denote condition absence. Empty cells denote conditions that are irrelevant to that prime implicant at the displayed solution type.

Value

A ggplot object.

See Also

`sol.df()`, qcaERT_plots

Examples

```
library(QCA)
data(LC)

conditions <- c("DEV", "URB", "LIT", "IND", "STB")
dir_exp <- rep("1", length(conditions))

tt <- truthTable(
  data = LC,
  outcome = "SURV",
  conditions = conditions,
  incl.cut = 0.8,
  n.cut = 1
)

enhanced <- findRows(tt, type = 2)

con <- minimize(
  input = tt,
  include = "",
  details = TRUE,
  show.cases = FALSE
)

par <- minimize(
  input = tt,
  include = "?",
  exclude = enhanced,
  details = TRUE,
  show.cases = FALSE
)

int <- minimize(
  input = tt,
  include = "?",
  dir.exp = dir_exp,
  exclude = enhanced,
  details = TRUE,
  show.cases = FALSE
)

solution_table <- sol.df(
  conservative = con,
  parsimonious = par,
  intermediate = int,
  solution = "all",
  which_M = 1,
  i_mode = "C1P1",
  include_cases = FALSE,
  digits = 2
)
```

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  library(ggplot2)

  sol.chart(
    solution_table,
    conditions = conditions,
    solution_types = c("conservative", "intermediate", "parsimonious"),
    show_cases = FALSE
  )
}


library(QCA)
library(ggplot2)
data(LR)

conditions <- c("DEV", "URB", "LIT", "IND", "STB")
outcome <- "SURV"
dir_exp <- rep("1", length(conditions))

thresholds <- list(
  DEV = findTh(LR$DEV, groups = 7),
  URB = findTh(LR$URB, groups = 4),
  LIT = findTh(LR$LIT, groups = 4),
  IND = findTh(LR$IND, groups = 4),
  STB = findTh(LR$STB, groups = 4),
  SURV = findTh(LR$SURV, groups = 4)
)

dat <- LR
dat$DEV <- calibrate(LR$DEV, type = "fuzzy", thresholds = thresholds$DEV)
dat$URB <- calibrate(LR$URB, type = "fuzzy", thresholds = thresholds$URB)
dat$LIT <- calibrate(LR$LIT, type = "fuzzy", thresholds = thresholds$LIT)
dat$IND <- calibrate(LR$IND, type = "fuzzy", thresholds = thresholds$IND)
dat$STB <- calibrate(LR$STB, type = "fuzzy", thresholds = thresholds$STB)
dat$SURV <- calibrate(LR$SURV, type = "fuzzy", thresholds = thresholds$SURV)

tt <- truthTable(
  data = dat,
  outcome = outcome,
  conditions = conditions,
  incl.cut = 0.8,
  n.cut = 1,
  complete = TRUE,
  show.cases = TRUE
)
enhanced <- findRows(tt, type = 2)

con <- minimize(tt, include = "", details = TRUE, show.cases = FALSE)
par <- minimize(tt, include = "?", exclude = enhanced, details = TRUE, show.cases = FALSE)
int <- minimize(
  tt,
  include = "?",
```

```
  dir.exp = dir_exp,
  exclude = enhanced,
  details = TRUE,
  show.cases = FALSE
)

solution_table <- sol.df(
  conservative = con,
  intermediate = int,
  parsimonious = par,
  solution = "all",
  which_M = 1,
  i_mode = "C1P1",
  include_cases = TRUE,
  digits = 2
)

sol.chart(solution_table)
```

`sol.df` *Build a compact solution table from QCA minimization output*

Description

Extracts prime implicants, fit statistics, model identifiers, intermediate branch labels, and optional case membership strings from supplied `QCA::minimize()` result objects and returns them as one combined data frame.

Usage

```
sol.df(
  conservative = NULL,
  intermediate = NULL,
  parsimonious = NULL,
  solution = "all",
  which_M = 1,
  i_mode = c("all", "C1P1"),
  branches = NULL,
  include_cases = TRUE,
  pi_incl_cut = NULL,
  digits = NULL,
  na_string = "-",
  verbose = FALSE
)
```

Arguments

conservative	Optional conservative minimization result of class "QCA_min".
intermediate	Optional intermediate minimization result of class "QCA_min".
parsimonious	Optional parsimonious minimization result of class "QCA_min".
solution	Which solution type to extract. Accepted values are "all", "con" or "conservative", "par" or "parsimonious", and "int" or "intermediate". When solution = "all", the function extracts every supplied object among conservative, parsimonious, and intermediate.
which_M	Positive integer giving which model to extract when a supplied minimization object contains multiple models.
i_mode	Character string controlling which intermediate branches to extract when intermediate is supplied and branches = NULL. Accepted values are "all" and "C1P1".
branches	Optional character vector of intermediate branch names to extract from intermediate$i.sol. When supplied, this overrides i_mode.
include_cases	Logical; if TRUE, include a Cases column using the case information stored in the minimization object when available. Case reconstruction expects the data-frame pims component produced by QCA::minimize().
pi_incl_cut	Optional lower cutoff applied to the prime-implicant consistency column. Rows with Consistency_PI < pi_incl_cut are removed.
digits	Optional non-negative integer used to round numeric columns in the returned table.
na_string	Single character string used to replace missing values in non-numeric output fields. The default is "-".
verbose	Logical; if TRUE, print a short progress message while processing each requested solution type.

Details

Supply at least one of conservative, parsimonious, or intermediate.

For conservative and parsimonious solutions, the function extracts rows from the IC component of the supplied minimization object. For intermediate solutions, it extracts rows from the selected i.sol branches.

If a supplied minimization object contains multiple models in IC$individual, which_M selects the model position to extract. If the requested model does not exist, the function stops with an error.

When include_cases = TRUE, the function first uses any cases column already stored in the relevant incl.cov table. If that is unavailable, it tries to reconstruct case strings from the corresponding pims data frame stored by QCA::minimize().

The QCA solution-object conventions are described in ?qcaERT_conventions.

Value

A data frame with one row per extracted prime implicant and the following columns:

Solution Solution type label: "Conservative", "Parsimonious", or "Intermediate".

`Model` Selected model number when model-specific output is available.

`Intermediate_CnPn` Intermediate branch label when rows come from `intermediate$i.sol`.

`Prime_Implicants` Prime implicant name.

`Consistency_PI` Prime-implicant consistency (`inclS`).

`PRI_PI` Prime-implicant PRI.

`Raw_Coverage_PI` Prime-implicant raw coverage (`covS`).

`Unique_Coverage_PI` Prime-implicant unique coverage (`covU`).

`Solution_Consistency` Solution-level consistency.

`Solution_PRI` Solution-level PRI.

`Solution_Coverage` Solution-level coverage.

`Cases` Case labels or case strings when available and requested.

The function returns a regular data frame, not a custom result object.

See Also

`sol.chart()`, `calib.test()`, `incl.test()`, `ncut.test()`, `loo.test()`, `subsample.test()`, `altset.test()`, `theory.test()`, `cluster.test()`

Examples

```
library(QCA)
data(LC)

conditions <- c("DEV", "URB", "LIT", "IND", "STB")
dir_exp <- rep("1", length(conditions))

tt <- truthTable(
  data = LC,
  outcome = "SURV",
  conditions = conditions,
  incl.cut = 0.8,
  n.cut = 1
)

enhanced <- findRows(tt, type = 2)

con <- minimize(
  input = tt,
  include = "",
  details = TRUE,
  show.cases = FALSE
)

par <- minimize(
  input = tt,
  include = "?",
  exclude = enhanced,
```

```
  details = TRUE,
  show.cases = FALSE
)

int <- minimize(
  input = tt,
  include = "?",
  dir.exp = dir_exp,
  exclude = enhanced,
  details = TRUE,
  show.cases = FALSE
)

sol.df(
  conservative = con,
  parsimonious = par,
  intermediate = int,
  solution = "all",
  which_M = 1,
  i_mode = "C1P1",
  include_cases = FALSE,
  digits = 2
)


library(QCA)
data(LR)

conditions <- c("DEV", "URB", "LIT", "IND", "STB")
outcome <- "SURV"
dir_exp <- rep("1", length(conditions))

thresholds <- list(
  DEV = findTh(LR$DEV, groups = 7),
  URB = findTh(LR$URB, groups = 4),
  LIT = findTh(LR$LIT, groups = 4),
  IND = findTh(LR$IND, groups = 4),
  STB = findTh(LR$STB, groups = 4),
  SURV = findTh(LR$SURV, groups = 4)
)

dat <- LR
dat$DEV <- calibrate(LR$DEV, type = "fuzzy", thresholds = thresholds$DEV)
dat$URB <- calibrate(LR$URB, type = "fuzzy", thresholds = thresholds$URB)
dat$LIT <- calibrate(LR$LIT, type = "fuzzy", thresholds = thresholds$LIT)
dat$IND <- calibrate(LR$IND, type = "fuzzy", thresholds = thresholds$IND)
dat$STB <- calibrate(LR$STB, type = "fuzzy", thresholds = thresholds$STB)
dat$SURV <- calibrate(LR$SURV, type = "fuzzy", thresholds = thresholds$SURV)

tt <- truthTable(
  data = dat,
  outcome = outcome,
  conditions = conditions,
```

```
  incl.cut = 0.8,
  n.cut = 1,
  complete = TRUE,
  show.cases = TRUE
)
enhanced <- findRows(tt, type = 2)

con <- minimize(tt, include = "", details = TRUE, show.cases = FALSE)
par <- minimize(tt, include = "?", exclude = enhanced, details = TRUE, show.cases = FALSE)
int <- minimize(
  tt,
  include = "?",
  dir.exp = dir_exp,
  exclude = enhanced,
  details = TRUE,
  show.cases = FALSE
)

sol.df(
  conservative = con,
  parsimonious = par,
  intermediate = int,
  solution = "all",
  which_M = 1,
  i_mode = "C1P1",
  include_cases = TRUE,
  digits = 3
)
```

`subsample.test` *Subsample robustness test for QCA solutions*

Description

Draws repeated subsamples without replacement, reruns the QCA analysis on each subsample, and records whether the monitored solution changes, whether selected fit measures change, and whether a subsample run ends with an error. The function can sample without stratification, stratify by the baseline outcome, or stratify by a grouping column. This is a stringent, punishing robustness check and is most useful when sample-composition sensitivity is substantively important.

Usage

```
subsample.test(
  data,
  outcome,
  conditions = NULL,
  calib = c("fixed", "recompute"),
```

```
  raw.data = NULL,
  calib_spec = NULL,
  sample_n = NULL,
  sample_prop = NULL,
  reps = 100,
  stratify = c("none", "outcome", "user"),
  strata = NULL,
  seed = NULL,
  case_labels = NULL,
  incl.cut = 1,
  n.cut = 1,
  solution = "all",
  include = NULL,
  dir.exp = NULL,
  which_M = 1,
  i_mode = c("all", "C1P1"),
  exclude_mode = c("recompute", "static", "none"),
  exclude_recompute = list(type = 2),
  exclude_static = NULL,
  fit_measures = c("inclS", "PRI", "covS"),
  fit_tol = 0,
  progress = TRUE,
  ...
)

## S3 method for class 'subsample_test'
print(x, row.names = FALSE, ...)

## S3 method for class 'subsample_test'
as.data.frame(x, ...)
```

Arguments

`data`	A data frame object containing the outcome and condition columns used in the QCA analysis. `data` must have at least three rows.
`outcome`	Name of the outcome. This must be a single non-empty character string.
`conditions`	Optional character vector of condition names. If `NULL`, the condition set is left to `QCA::truthTable()`. When `calib = "recompute"`, `conditions` must be supplied explicitly.
`calib`	Calibration handling for subsample runs. `"fixed"` uses the calibrated values already present in `data`. `"recompute"` recalibrates the outcome and conditions within each subsample using `raw.data` and `calib_spec`.
`raw.data`	Raw-data frame object used when `calib = "recompute"`. It must have the same number of rows as `data`.
`calib_spec`	Calibration specification used when `calib = "recompute"`. This must be a named list keyed by `c(outcome, conditions)`. Each entry must describe the raw source column, the set type, and the calibration inputs used to rebuild the

calibrated set. Use the same calib_spec structure described for calib.test(). An entry may additionally contain findTh, a named list of QCA::findTh() arguments; when supplied, thresholds are re-estimated within each subsample instead of reusing the baseline thresholds.

sample_n — Integer subsample size of at least 2 and strictly smaller than nrow(data). Supply exactly one of sample_n or sample_prop.

sample_prop — Proportion used to determine the realized subsample size. Supply exactly one of sample_n or sample_prop. The realized sample size is round(sample_prop * nrow(data)).

reps — Positive integer giving the number of subsample replications.

stratify — Stratification mode. "none" samples from the full dataset without stratification, "outcome" stratifies on the baseline analysis outcome, and "user" stratifies on strata.

strata — Optional stratification column used when stratify = "user". It must have length nrow(data) and contain no missing values.

seed — Optional finite integer seed passed to set.seed() before drawing the subsamples.

case_labels — Optional character vector of case labels with length nrow(data). If NULL, the function uses rownames(data) when available; otherwise it uses row numbers converted to character.

incl.cut — Inclusion cutoff passed to QCA::truthTable(). This must be a single finite number in [0, 1].

n.cut — Truth table frequency cutoff passed to QCA::truthTable(). This must be an integer-like value of at least 1.

solution — Solution type to monitor. Accepted values are "all", "con" or "conservative", "par" or "parsimonious", and "int" or "intermediate".

include — Optional minimization include setting. Currently, this argument accepts only NULL, "", or "?".

dir.exp — Directional expectations used when the monitored solution is intermediate.

which_M — Positive integer giving which solution alternative to use when minimization returns multiple models.

i_mode — Character string controlling intermediate-solution selection. Accepted values are "all" and "C1P1".

exclude_mode — Character string controlling how excluded rows are handled for parsimonious and intermediate minimization. "recompute" recalculates exclusions from each subsample truth table, "static" reuses exclude_static, and "none" does not use exclusions.

exclude_recompute — Named list of arguments passed to QCA::findRows() when exclude_mode = "recompute".

exclude_static — Already computed exclusion object reused when exclude_mode = "static".

fit_measures — Character vector of fit-measure names to compare between the baseline run and each subsample run. Use NULL to disable fit comparison.

fit_tol	Non-negative tolerance used when deciding whether fit values changed.
progress	Logical; if TRUE and the session is interactive, show a text progress bar.
...	Additional arguments. In subsample.test(), named arguments matching QCA::truthTable() formals are passed to QCA::truthTable(), and the remaining named arguments are forwarded to QCA::minimize() after removing names reserved by subsample.test(). The function also looks in ... for include, dir.exp, or direxp if those arguments were not supplied explicitly. In print.subsample_test(), ... is passed to print.data.frame(). In as.data.frame.subsample_test(), ... is ignored.
x	A subsample_test object returned by subsample.test().
row.names	Logical; passed to print.data.frame() by print.subsample_test().

Details

The function first runs the baseline analysis on the full dataset. It then draws reps subsamples without replacement and reruns the analysis on each subsample.

Exactly one of sample_n or sample_prop must be supplied. The realized subsample size must be at least 2 and strictly smaller than nrow(data).

When stratify = "outcome", the function stratifies on the baseline analysis outcome and therefore requires that outcome to be crisp/binary 0/1. When stratify = "user", the function stratifies on the supplied strata vector.

When calib = "fixed", the subsample runs use the calibrated values already present in data. When calib = "recompute", the function rebuilds the calibrated outcome and condition columns within each subsample using QCA::findTh() and QCA::calibrate() as specified in calib_spec.

With calib = "recompute", rows in raw.data and data are paired before sampling and recalibration. Informative row names must be unique and identical in the same order on both inputs. If data has only default row names and case_labels is supplied, raw.data must use those labels as unique row names in the same order. If neither input has identifiable labels, pairing is positional and the inputs must not be independently reordered.

The shared solution-control, exclusion, and returned-object conventions are described in ?qcaERT_conventions.

Value

An object of class subsample_test with the following components:

diagnostics A detailed data frame with one row per subsample run. It records the replication number, subsample size, holdout size, run status, solution-change classification, fit-change classification, exact-match status relative to the baseline solution, term-set Jaccard similarity to the baseline solution, and error information when a run fails.

results A compact data frame with the columns rep, n_sample, n_holdout, status, exact_match_baseline, term_jaccard_baseline, solution_change, fit_changed_types, n_fit_deltas, and max_abs_fit_delta. exact_match_baseline is TRUE when all requested solution types match the baseline. It is FALSE when a comparison shows a change, and NA when the comparison is incomplete and no change has been established. If fit comparison is requested, n_fit_deltas is NA when no fit measures can be compared between the selected models.

`summary` A list with summary objects named `exact_solution`, `term_stability`, `fit_stability`, `similarity`, and `calibration`.

`baseline` A list containing the baseline analysis built from the full dataset, including the truth table, minimization output, selected solution terms used for comparison, fit values, exclusion information, and calibration information.

`by_run` A named list with one entry per subsample run. Each entry stores the sampled and held-out cases, run status, baseline and subsample analyses, change summaries, fit deltas, exclusion information, calibration information, and any error information for that run.

`settings` A list containing the analysis settings used to build the result object.

`print.subsample_test()` prints a concise summary and the `results` table. `as.data.frame.subsample_test()` returns the `results` table.

See Also

`calib.test()`, `incl.test()`, `ncut.test()`, `loo.test()`, `altset.test()`, `theory.test()`, `cluster.test()`, `sol.df()`

Examples

```
library(QCA)
data(LR)

conditions <- c("DEV", "URB", "LIT", "IND", "STB")
outcome <- "SURV"
dir_exp <- rep("1", length(conditions))

thresholds <- list(
  DEV = findTh(LR$DEV, groups = 7),
  URB = findTh(LR$URB, groups = 4),
  LIT = findTh(LR$LIT, groups = 4),
  IND = findTh(LR$IND, groups = 4),
  STB = findTh(LR$STB, groups = 4),
  SURV = findTh(LR$SURV, groups = 4)
)

dat <- LR
dat$DEV <- calibrate(LR$DEV, type = "fuzzy", thresholds = thresholds$DEV)
dat$URB <- calibrate(LR$URB, type = "fuzzy", thresholds = thresholds$URB)
dat$LIT <- calibrate(LR$LIT, type = "fuzzy", thresholds = thresholds$LIT)
dat$IND <- calibrate(LR$IND, type = "fuzzy", thresholds = thresholds$IND)
dat$STB <- calibrate(LR$STB, type = "fuzzy", thresholds = thresholds$STB)
dat$SURV <- calibrate(LR$SURV, type = "fuzzy", thresholds = thresholds$SURV)

out <- subsample.test(
  data = dat,
  outcome = outcome,
  conditions = conditions,
  calib = "fixed",
  sample_prop = 0.8,
  reps = 50,
```

```
  seed = 123,
  case_labels = rownames(dat),
  incl.cut = 0.8,
  n.cut = 1,
  solution = "all",
  dir.exp = dir_exp,
  fit_measures = c("inclS", "PRI", "covS"),
  progress = TRUE
)

out
as.data.frame(out)
```

`theory.test` *Theory-specification robustness for QCA models*

Description

Compare several theoretically oriented QCA condition sets while holding the outcome, truth-table cutoffs, solution type, exclusion handling, and model-selection settings constant. Each theory supplies its own condition set.

Usage

```
theory.test(
  data,
  outcome,
  theories,
  incl.cut = 1,
  n.cut = 1,
  solution = "all",
  include = NULL,
  dir.exp = NULL,
  which_M = 1,
  i_mode = "C1P1",
  exclude_mode = c("recompute", "static", "none"),
  exclude_recompute = list(type = 2),
  exclude_static = NULL,
  progress = TRUE,
  ...
)

## S3 method for class 'theory_test'
print(x, row.names = FALSE, ...)

## S3 method for class 'theory_test'
as.data.frame(x, ...)
```

Arguments

data	A non-empty data frame containing the calibrated outcome and calibrated condition columns.
outcome	Name of the calibrated outcome column in data.
theories	Named list of theory-specific condition sets. Each entry must be a non-empty character vector of condition names in data. Conditions may be shared across theories, but no theory may contain the outcome as a condition.
incl.cut	Inclusion cutoff to be used for every theory-specific truth table.
n.cut	Frequency cutoff to be used for every theory-specific truth table.
solution	Solution type to compare. Accepted values are "all", "con" or "conservative", "par" or "parsimonious", and "int" or "intermediate".
include	Optional minimization include setting. Currently, this argument accepts only NULL, "", or "?".
dir.exp	Theory-specific directional expectations. Use NULL when no intermediate solutions are monitored. Otherwise supply a named list with one entry per theory. Each entry may be an ordered character vector aligned to that theory's conditions or a named character vector whose names match that theory's conditions.
which_M	Positive integer giving which solution alternative to use when minimization returns multiple models.
i_mode	Character string controlling intermediate-solution selection. Accepted values are "C1P1" and "all". The default is "C1P1" because comparative theory testing needs a single comparable intermediate branch per theory by default.
exclude_mode	Character string controlling how excluded rows are handled for parsimonious and intermediate minimization. "recompute" recalculates exclusions separately for each theory-specific truth table, "static" reuses theory-specific static exclusions supplied through exclude_static, and "none" does not use exclusions.
exclude_recompute	Named list of arguments passed to QCA::findRows() when exclude_mode = "recompute".
exclude_static	Static exclusions used when exclude_mode = "static". For theory.test(), this must be a named list with one entry per theory when parsimonious or intermediate solutions are monitored.
progress	Logical; if TRUE, show a text progress bar during per-theory QCA runs in interactive sessions.
...	Additional arguments split between QCA::truthTable() and QCA::minimize(). The function also looks in ... for include, dir.exp, or direxp if those arguments were not supplied explicitly. In print.theory_test(), ... is passed to print.data.frame(). In as.data.frame.theory_test(), ... is ignored.
x	A theory_test object returned by theory.test().
row.names	Logical; if TRUE, print row names in the selected solutions table printed by print.theory_test().

Details

The function builds a separate truth table for each theory, recomputes exclusions separately when requested, and runs the monitored solution types under the same analytic settings. The raw per-theory QCA objects are stored in `by_theory`. `results$models` gives the clean model-level comparison table, and `results$solutions` gives the selected prime implicants or solution terms by theory and solution type. `results$pairwise` compares selected solution memberships across theories using fuzzy Jaccard similarity and mean absolute membership differences.

Value

An object of class `theory_test` with the following components:

`diagnostics` A detailed internal diagnostics table with one row per theory and monitored solution type.

`results` A named list with `models`, `pairwise`, and `solutions` tables. `models` is populated with model-level diagnostics and fit values; `solutions` is populated with selected solution terms and term-level fit values; `pairwise` is populated with pairwise theory comparisons by solution type.

`by_theory` A named list containing the theory-specific condition sets, directional expectations, truth tables, exclusions, minimization objects, selected solution terms used for comparison, and current run status.

`settings` A list containing the analysis settings used in the call.

`print.theory_test()` prints a compact theory-specification summary and the selected solutions table. `as.data.frame.theory_test()` returns `results$models`.

See Also

`calib.test()`, `incl.test()`, `ncut.test()`, `loo.test()`, `subsample.test()`, `altset.test()`, `cluster.test()`, `sol.df()`

Examples

```
library(QCA)
data(LR)

outcome <- "SURV"

thresholds <- list(
  DEV = findTh(LR$DEV, groups = 7),
  URB = findTh(LR$URB, groups = 4),
  LIT = findTh(LR$LIT, groups = 4),
  IND = findTh(LR$IND, groups = 4),
  STB = findTh(LR$STB, groups = 4),
  SURV = findTh(LR$SURV, groups = 4)
)

dat <- LR
dat$DEV <- calibrate(LR$DEV, type = "fuzzy", thresholds = thresholds$DEV)
dat$URB <- calibrate(LR$URB, type = "fuzzy", thresholds = thresholds$URB)
```

```
dat$LIT <- calibrate(LR$LIT, type = "fuzzy", thresholds = thresholds$LIT)
dat$IND <- calibrate(LR$IND, type = "fuzzy", thresholds = thresholds$IND)
dat$STB <- calibrate(LR$STB, type = "fuzzy", thresholds = thresholds$STB)
dat$SURV <- calibrate(LR$SURV, type = "fuzzy", thresholds = thresholds$SURV)

theories <- list(
  development = c("DEV", "URB", "LIT"),
  industrial = c("DEV", "URB", "IND"),
  broad = c("DEV", "URB", "LIT", "IND", "STB")
)

dir_exp <- list(
  development = c("1", "1", "1"),
  industrial = c("1", "1", "1"),
  broad = c("1", "1", "1", "1", "1")
)

theory_out <- theory.test(
  data = dat,
  outcome = outcome,
  theories = theories,
  incl.cut = 0.8,
  n.cut = 1,
  solution = "all",
  dir.exp = dir_exp,
  progress = TRUE
)

theory_out
as.data.frame(theory_out)
theory_out$results$solutions
theory_out$results$pairwise
```

`whr_calib_spec` *Calibration specifications for the WHR demonstration data*

Description

A named list of qcaERT calibration specifications for the five condition sets in whr_raw and whr_calibrated.

Usage

```
whr_calib_spec
```

Format

A named list with entries for `GDP`, `SOC`, `LIFE`, `FREE`, and `GEN`. Each entry contains:

raw The column in whr_raw used for calibration.

type The calibration type, either fuzzy or crisp.

method The QCA calibration method.

thresholds The thresholds or ordered cutpoints passed to `QCA::calibrate()`.

This object covers the condition sets. If the outcome is also tested in `calib.test()` or `altset.test()`, add an entry for `WELL`.

Source

World Happiness Report 2025, data for Figure 2.1, used with permission. See https://www.worldhappiness.report/data-sharing/.

Examples

```
data(whr_calib_spec)
whr_calib_spec
```

`whr_calibrated` *Calibrated World Happiness Report 2025 demonstration data*

Description

A calibrated version of whr_raw for qcaERT demonstrations. It contains the same countries and columns as whr_raw, but the values are calibrated set memberships.

Usage

```
whr_calibrated
```

Format

A data frame with 145 countries and 6 calibrated sets:

WELL Calibrated outcome. Direct fuzzy calibration with thresholds 2.181, 5.3555, and 7.0505.

GDP Direct fuzzy calibration with six thresholds.

SOC Indirect fuzzy calibration with six ordered cutpoints.

LIFE Direct fuzzy calibration with three thresholds.

FREE Crisp calibration with one threshold.

GEN Direct fuzzy calibration with three thresholds.

The condition calibration specifications are stored in whr_calib_spec.

Source

World Happiness Report 2025, data for Figure 2.1, used with permission. See https://www.worldhappiness.report/data-sharing/.

Examples

```
data(whr_calibrated)
head(whr_calibrated)
```

`whr_raw` *World Happiness Report 2025 demonstration data*

Description

A country-level data set derived from the World Happiness Report 2025 data for Figure 2.1. It is included to support qcaERT demonstrations based on a real QCA workflow.

Usage

```
whr_raw
```

Format

A data frame with 145 countries and 6 columns:

WELL Life evaluation, three-year average.

GDP Contribution of log GDP per capita.

SOC Contribution of social support.

LIFE Contribution of healthy life expectancy.

FREE Contribution of freedom to make life choices.

GEN Contribution of generosity.

Country names are stored as row names. The explanatory-factor columns are World Happiness Report decomposition components, not raw survey responses.

Source

World Happiness Report 2025, data for Figure 2.1, used with permission. See https://www.worldhappiness.report/data-sharing/.

Helliwell, J. F., Layard, R., Sachs, J. D., De Neve, J.-E., Aknin, L. B., & Wang, S. (Eds.). (2025). *World Happiness Report 2025*. University of Oxford: Wellbeing Research Centre.

Examples

```
data(whr_raw)
head(whr_raw)
```

Index

∗ **datasets**
whr_calib_spec, 59
whr_calibrated, 60
whr_raw, 61

altset.test, 5
altset.test(), *3*, *8*, *14*, *20*, *25*, *29*, *32*, *34, 35*, *42*, *49*, *55*, *58*, *60*
as.data.frame.altset_test *(*altset.test*)*, 5
as.data.frame.altset_test(), *8*
as.data.frame.calib_test *(*calib.test*)*, 11
as.data.frame.calib_test(), *13*
as.data.frame.cluster_test *(*cluster.test*)*, 18
as.data.frame.cluster_test(), *19*
as.data.frame.incl_test *(*incl.test*)*, 22
as.data.frame.incl_test(), *23*
as.data.frame.loo_test *(*loo.test*)*, 26
as.data.frame.loo_test(), *28*
as.data.frame.ncut_test *(*ncut.test*)*, 30
as.data.frame.ncut_test(), *31*
as.data.frame.subsample_test *(*subsample.test*)*, 51
as.data.frame.subsample_test(), *54*
as.data.frame.theory_test *(*theory.test*)*, 56
as.data.frame.theory_test(), *57*

calib.test, 11
calib.test(), *3, 4*, *9*, *13*, *20*, *25*, *27*, *29*, *32*, *34, 35*, *37*, *42, 43*, *49*, *53*, *55*, *58*, *60*
cluster.test, 18
cluster.test(), *3*, *9*, *14*, *19*, *25*, *29*, *32*, *42*, *49*, *55*, *58*

incl.test, 22
incl.test(), *3, 4*, *9*, *14*, *20*, *23*, *29*, *32*, *34, 35*, *37*, *42, 43*, *49*, *55*, *58*

loo.test, 26
loo.test(), *3*, *9*, *14*, *20*, *25*, *28*, *32*, *34, 35*, *42*, *49*, *55*, *58*

ncut.test, 30
ncut.test(), *3*, *9*, *14*, *20*, *25*, *29*, *31*, *34, 35*, *42*, *49*, *55*, *58*

plot.calib_test *(*qcaERT_plots*)*, 36
plot.incl_test *(*qcaERT_plots*)*, 36
plot.theory_test *(*qcaERT_plots*)*, 36
print.altset_test *(*altset.test*)*, 5
print.altset_test(), *8*
print.calib_test *(*calib.test*)*, 11
print.calib_test(), *13*
print.cluster_test *(*cluster.test*)*, 18
print.cluster_test(), *19*
print.data.frame(), *8*, *13*, *19*, *23*, *28*, *31*, *54*, *57*
print.incl_test *(*incl.test*)*, 22
print.incl_test(), *23*
print.loo_test *(*loo.test*)*, 26
print.loo_test(), *28*
print.ncut_test *(*ncut.test*)*, 30
print.ncut_test(), *31*
print.subsample_test *(*subsample.test*)*, 51
print.subsample_test(), *54*
print.theory_test *(*theory.test*)*, 56
print.theory_test(), *57*

QCA::calibrate(), *2, 3*, *6*, *12*, *28*, *34*, *54*, *60*
QCA::findRows(), *2*, *7*, *13*, *23*, *27*, *31*, *34*, *53*, *57*
QCA::findTh(), *27, 28*, *35*, *53, 54*
QCA::minimize(), *2, 3*, *8*, *13*, *19, 20*, *23*, *28*, *31*, *36*, *47, 48*, *54*, *57*
QCA::truthTable(), *2, 3*, *7, 8*, *13*, *19*, *23, 24*, *27, 28*, *30, 31*, *52–54*, *57*
qcaERT *(*qcaERT-package*)*, 2

qcaERT-package, 2
qcaERT_conventions, 33
qcaERT_plots, *14*, *25*, 36, *44*
qcaERT_tests, 41

set.seed(), *7*, *53*
sol.chart, 43
sol.chart(), *3, 4*, *35, 36*, *39*, *42, 43*, *49*
sol.df, 47
sol.df(), *3, 4*, *9*, *14*, *20*, *25*, *29*, *32*, *35, 36*, *39*, *42–44*, *55*, *58*
subsample.test, 51
subsample.test(), *3*, *9*, *14*, *20*, *25*, *29*, *32*, *34, 35*, *42*, *49*, *54*, *58*

theory.test, 56
theory.test(), *3, 4*, *9*, *14*, *20*, *25*, *29*, *32*, *34*, *37*, *42, 43*, *49*, *55*, *57*

whr_calib_spec, 59, *60*
whr_calibrated, *59*, 60
whr_raw, *59, 60*, 61