Package 'ShortForm'

September 10, 2026

Type Package

Title Automatic Short Form Creation

Version 1.0.0

Date 2026-09-09

Description Performs automatic creation of short forms of scales with an ant colony optimization algorithm and a Tabu search. As implemented in the package, the ant colony algorithm randomly selects items to build a model of a specified length, then updates the probability of item selection according to the fit of the best model within each set of searches. The algorithm continues until the same items are selected by multiple ants a given number of times in a row. On the other hand, the Tabu search changes one parameter at a time to be either free, constrained, or fixed while keeping track of the changes made and putting changes that result in worse fit in a ``tabu" list so that the algorithm does not revisit them for some number of searches. See Leite, Huang, & Marcoulides (2008) <doi:10.1080/00273170802285743> for an applied example of the ant colony algorithm, and Marcoulides & Falk (2018) <doi:10.1080/10705511.2017.1409074> for an applied example of the Tabu search.

License LGPL (>= 2.0, < 3) | Mozilla Public License

LazyData TRUE

Suggests knitr, MplusAutomation (>= 0.7), rmarkdown, testthat

Imports lavaan (>= 0.5-22), stringr, methods, doSNOW, parallel, foreach, rlang

Depends R (>= 4.1.0)

URL https://github.com/AnthonyRaborn/ShortForm

BugReports https://github.com/AnthonyRaborn/ShortForm/issues

Encoding UTF-8

Config/roxygen2/version 8.0.0

VignetteBuilder knitr

NeedsCompilation no

Author Anthony Raborn [aut, cre] (ORCID:
<https://orcid.org/0000-0002-8083-4739>),
Walter Leite [aut]

Maintainer Anthony Raborn <anthony.w.raborn@gmail.com>

Repository CRAN

Date/Publication 2026-09-10 03:30:08 UTC

Contents

.onAttach *Package Attach Hook Function*

Description

Hook triggered when package attached.

Usage

```
.onAttach(lib, pkg)
```

Arguments

lib	a character string giving the library directory where the package defining the namespace was found
pkg	a character string giving the name of the package

Details

Idea taken from https://github.com/ntguardian/MCHT/blob/master/R/StartupMessage.R

Examples

```
ShortForm:::.onAttach(.libPaths()[1], "ShortForm")
```

ACO-class *An S4 class for the Ant Colony Optimization Algorithm*

Description

An S4 class for the Ant Colony Optimization Algorithm

Value

An S4 object of class 'ACO'.

Slots

`function_call` The original function call.

`summary` A summary 'data.frame' indicating the algorithm results for each iteration.

`final_solution` A 'matrix' with the final solution information, including fit indices, selected items, and pheromone level.

`best_model` A 'lavaan' object of the final solution.

`best_syntax` A 'character' vector of the final solution model syntax.

`runtime` A 'difftime' object of the total run time of the function.

`add.param` *Adds a parameter to the given search table. Checks whether parameter is involved in any (in)equality constraints in a fitted lavaan model*

Description

Adds a parameter to the given search table. Checks whether parameter is involved in any (in)equality constraints in a fitted lavaan model

Usage

```
add.param(
  fitted.model,
  ptab,
  syntax,
  nullval = NULL,
  free = NULL,
  block = NULL
)
```

Arguments

`fitted.model`	fitted lavaan model
`ptab`	search table
`syntax`	model.syntax specifying the parameter to add to the current table
`nullval`	optional numeric value specifying what the parameter should be fixed to (when fixed)
`free`	optional logical value specifying whether the parameter should initially be set free (or not)
`block`	optional numeric value specifying the group number to which the parameter corresponds

Value

A `data.frame` with lavaan-formatted parameter values.

Author(s)

Carl F. Falk

References

doi:10.1080/10705511.2017.1409074

See Also

Other Tabu Search: `refit.model()`, `search.prep()`

Examples

```
## Not run:
# load simulation data and select columns used in this example
data(simulated_test_data)
tabuData <- simulated_test_data[, c(1:10)]

# specify an improper model (improper because data is unidimensional)
tabuModel <- "
Ability =~ Item1 + Item2 + Item3 + Item4
FakeAbility =~ Item5 + Item6 + Item7 + Item8
Ability ~ Outcome
FakeAbility ~ 0*Outcome"

# run the initial misspecified model for Tabu

init.model <- lavaan::lavaan(
  model = tabuModel, data = tabuData,
  auto.var = TRUE, auto.fix.first = FALSE, std.lv = TRUE, auto.cov.lv.x = TRUE
)

# Use search.prep to prepare for the Tabu search
ptab <- search.prep(fitted.model = init.model, loadings = TRUE, fcov = TRUE, errors = FALSE)

# add an additional (mispecified) parameter
additional.param <- "Item1 ~~ 0.5*Item3"
ptab <- add.param(fitted.model = init.model, ptab = ptab, syntax = additional.param)

# Perform Tabu Search
trial <- tabu.sem(init.model = init.model, ptab = ptab, obj = AIC, niter = 2, tabu.size = 5)

## End(Not run)
```

`antColony` *A function to implement the ant colony optimization algorithm for short form specification searches with the package lavaan.*

Description

The Ant Colony Optimization (ACO) algorithm (Dorigo & Stutzle, 2004) can produce short forms of scales that are optimized with respect to characteristics selected by the developer, such as model fit and predictive relationships with other variables. The algorithm is based on the foraging behavior of a group of ants, which start searching for food in a variety of directions and then eventually all ants converge to the shortest distance to the food source. This behavior occurs because ants leave a pheronome trail behind as they search for food and ants in shorter paths leave stronger pheronome trails, which are detected by other ants and that will lead them to follow the shortest trail.

Usage

```
antColony(
  data = NULL,
  sample.cov = NULL,
  sample.nobs = NULL,
  ants = 20,
  evaporation = 0.9,
  initialModel,
  items = NULL,
  itemsPerFactor = NULL,
  bifactor = NULL,
  steps = 50,
 lavaan.model.specs = list(model.type = "cfa", auto.var = T, estimator = "default",
   ordered = NULL, int.ov.free = TRUE, int.lv.free = FALSE, auto.fix.first = TRUE,
   auto.fix.single = TRUE, auto.var = TRUE, auto.cov.lv.x = TRUE, auto.th = TRUE,
   auto.delta = TRUE, auto.cov.y = TRUE, std.lv = F, group = NULL, group.label = NULL,
    group.equal = "loadings", group.partial = NULL, group.w.free = FALSE),
  pheromone.calculation = "gamma",
  fit.indices = c("cfi", "tli", "rmsea"),
  fit.statistics.test = "(cfi > 0.95)&(tli > 0.95)&(rmsea < 0.06)",
  maxIterations = 1000,
  parallel = T,
  verbose = TRUE
)
```

Arguments

`data`	The data being used in data frame format. Default value is `null`. Only one of `data` or `sample.cov` should be used.
`sample.cov`	The sample covariance matrix. See lavaan for the specific format needed. Default value is `null`. Only one of `data` or `sample.cov` should be used.
`sample.nobs`	A numeric value indicating the number of observations in the sample covariance matrix. If `sample.cov` is used, this must be filled in. Default value is `null`.
`ants`	A numeric value indicating the number of ants to send (e.g., number of short forms to evaluate) per iteration. Default value is 20.
`evaporation`	A numeric value which sets the percentage of the pheromone that is retained after evaporation between steps of the algorithm. Default value is 0.9, indicating 10 (0,1), exclusive.
`initialModel`	The lavaan formatted model. See lavaan for more details. Defaults to the default lavaan values. NOTE: Each factor and/or regression needs to be specified on a single line. Newline breaks and carriage returns WILL break the function.
`items`	A ‘character‘ vector of candidate item names. Defaults to ‘NULL‘, which uses all column names in ‘data‘ (required if ‘data‘ is ‘NULL‘, i.e. when using ‘sample.cov‘/‘sample.nobs‘ instead). Every candidate item must appear on its factor’s line in ‘initialModel‘; an item cross-loading on multiple factors should appear on each of those factors’ lines.

`itemsPerFactor` Numeric vector with the target number of items to retain per factor, in the same order the factors appear in 'initialModel'.

`bifactor` Either the name of the factor that all of the chosen items will load on (as character), or 'NULL' if the model is not a bifactor model.

`steps` A numeric value that sets the stopping rule, which is the number of ants in a row for which the model does not change.

`lavaan.model.specs`
A list which contains the specifications for the lavaan model. The default values are the defaults for lavaan to perform a CFA. These are automatically set internally, then updated by the user-provided values – a partial list is accepted, and any element you omit falls back to the default for that element. Every name you do supply must match one of the recognized element names, or the call errors. Note that this drastically affects the algorithm, and care must be taken to ensure that the algorithm can fit valid models as it searches for the best model. See the default arguments for examples of what you can change and lavaan for more details on what arguments are available to change.

`pheromone.calculation`
A character string specifying the method for calculating the pheromone strength. Must be one of "`gamma`" (standardized latent regression coefficients), "`beta`" (standardized observed regression coefficients), "`regression`" (both latent and observed regression coefficients, if they exist) or "`variance`" (proportion of variance explained by model). You must specify the entire string. Default is `gamma`.

`fit.indices` The fit indices (in lavaan format) extracted for model optimization. See lavaan for more details.

`fit.statistics.test`
A character vector of the logical test being used for model optimization. The default is "`(cfi > 0.95)&(tli > 0.95)&(rmsea < 0.06)`". The format for the logical test should match 1) the names of the indices being used in lavaan and 2) the default provided above. At least one fit index must be included.

`maxIterations` The maximum number of ants to run before the algorithm stops. This includes failed iterations as well. Default is 1000.

`parallel` An option for using parallel processing. If `TRUE`, the function will utilize all available cores (up to the number of ants). Default is `TRUE`.

`verbose` Logical. If 'TRUE' (the default), prints per-ant progress to the console. The full per-run history is always available afterward via the returned object's 'summary' and 'final_solution' slots regardless of this setting.

Details

This function sends a specified number of ants per iteration, which randomly select items to build a model, then evaluates the model based on pheromone levels. The pheromone levels are updated after each iteration according to the best-fitting model of that iteration. The algorithm's stopping rule is to end the search when a certain solution is the same for a given number of ants in a row.

PREPARATORY STEPS: For the ACO algorithm implementation for short for selection, the following decisions are needed:

1. Determine the target size for the short form.

2. Determine which characteristics should be optimized.

3. Define how the pheronome level will be computed: This is a function of the characteristics of the short form that will be optimized. In Leite, Huang and Marcoulides (2008), the pheromone level was zero if model fit indices did not meet Hu and Bentler's (1999) suggested thresholds, and equal to the sum of path coefficients of a predictor variable if model fit indices met thresholds. Currently, the package only implements pheromone calculation based on regression coefficients or variance explained, with user-selected model fit index thresholds.

4. Define how many short forms should be evaluated before the best-so-far pheronome level is examined. Leite, Huang and Marcoulides (2008) used 10 short forms.

5. Define the percentage of pheronome evaporation, if any. Leite, Huang and Marcoulides (2008) used 5%.

6. Define convergence criterion. Leite, Huang and Marcoulides (2008) set the algorithm to converge if the short form did not improve in 100 x number of short forms in step 4.

IMPLEMENTATION: Once these decisions are made, the ACO algorithm selects short forms with the following steps:

Step 1. All items are assigned an initial weight of 1.

Step 2. A set of n short forms is selected by sampling with probability proportional to the item weights.

Step 3. Fit the latent variable model to the n short forms.

Step 4. Calculate the pheromone levels for the n short forms. Define the best-so-far pheronome level (if iteration 1) or compare the current best pheronome from the set of n short forms to the best-so-far pheronome.

Step 5. If the pheromone level of the best short form from step 4 exceeds the best-so-far pheronome level, update the best-so-far pheromone level and add it to the current weight of the items of the best short form.

Step 6. Return to step 2 until convergence criterion is reached.

Value

An S4 object of class 'ACO', with (among other slots) 'final_solution' holding a named matrix with the final model's best fit indices, the final pheromone level (either the mean of the standardized regression coefficients (gammas, betas, or both), or the mean variance explained), and a series of 0/1 values indicating the items selected in the final solution; 'summary' holding the summary data.frame of the best fit statistic value(s) for each run, the items chosen for said best fit, the mean gamma, beta, and variance explained for the best fit, and the item pheromone levels after each run; 'best_model' holding the best-fitting lavaan model object; and 'best_syntax' holding the best-fitting model syntax.

Author(s)

Anthony W Raborn, <anthony.w.raborn@gmail.com>

See Also

antcolony.mplus

Other Ant Colony Algorithms: antcolony.mplus()

Examples

```
# a 3-factor example using the HolzingerSwineford1939 data from `lavaan`

# some changes to the default values
# notice that in this example we are recreating the original model
abilityShortForm <- antColony(
  data = lavaan::HolzingerSwineford1939,
  ants = 2, evaporation = 0.7,
  initialModel = " visual  =~ x1 + x2 + x3
                   textual =~ x4 + x5 + x6
                   speed   =~ x7 + x8 + x9 ",
  itemsPerFactor = c(3, 3, 3), steps = 2, fit.indices =
    c("cfi"), fit.statistics.test = "(cfi > 0.6)",
    maxIterations = 2, parallel = FALSE
)
## Not run:
# using simulated test data and the default values for lavaan.model.specs
# first, read in the original or "full" model
data(exampleAntModel) # a character vector for a lavaan model

# load the data
data(simulated_test_data)

# finally, call the function with some minor changes to the default values.
# every candidate item (all 56, in this case) must already appear on its
# factor's line in exampleAntModel -- see ?exampleAntModel
abilityShortForm <- antColony(
  data = simulated_test_data,
  ants = 5, evaporation = 0.7, initialModel = exampleAntModel,
  itemsPerFactor = 20,
  steps = 3, fit.indices = c("cfi", "rmsea"),
  fit.statistics.test = "(cfi > 0.95)&(rmsea < 0.05)",
  maxIterations = 500
)

abilityShortForm # print the results of the final short form

# an example using binary (ordered) data
# create the simulated full model and model data
sim_model <- "
f1 =~ x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9 + x10
f2 =~ x11 + x12 + x13 + x14 + x15 + x16 + x17 + x18 + x19 + x20
f3 =~ x21 + x22 + x23 + x24 + x25 + x26 + x27 + x28 + x29 + x30"

sim_data <-
  cbind(
    psych::sim.rasch(nvar = 10)$items,
```

```
    psych::sim.rasch(nvar = 10)$items,
    psych::sim.rasch(nvar = 10)$items
  )

colnames(sim_data) = paste0("x", 1:30)
# fit with antColony
# note that ONLY the estimator and ordered args
# of lavaan.model.specs are changed. This retains
# the default args, fitting a CFA but with ordered data.
example <-
antColony(
  data = sim_data,
  ants = 5, evaporation = 0.7,
  initialModel = sim_model,
  lavaan.model.specs =
    list(estimator = "wlsmv", ordered = T),
  itemsPerFactor = c(5, 5, 5),
  steps = 20,
  fit.indices = c("cfi.scaled"),
  fit.statistics.test = "(cfi.scaled > 0.90)",
  maxIterations = 500,
  parallel = T
)
# note that this example will take a bit of time to run
# as ordered data factor analysis is computationally expensive.

## End(Not run)
```

`antcolony.mplus` *A function to implement the ant colony optimization algorithm for short form specification searches, either using MPlus directly via* `system` *calls or using Mplus indirectly with the package MplusAutomation.*

Description

The Ant Colony Optimization (ACO) algorithm (Dorigo & Stutzle, 2004) can produce short forms of scales that are optimized with respect to characteristics selected by the developer, such as model fit and predictive relationships with other variables. The algorithm is based on the foraging behavior of a group of ants, which start searching for food in a variety of directions and then eventually all ants converge to the shortest distance to the food source. This behavior occurs because ants leave a pheromone trail behind as they search for food and ants in shorter paths leave stronger pheromone trails, which are detected by other ants and that will lead them to follow the shortest trail.

Usage

```
antcolony.mplus(
  ants = 20,
  evaporation = 0.95,
```

```
  mplus = NULL,
  list.items = NULL,
  full = NULL,
  i.per.f = NULL,
  factors = NULL,
  steps = 50,
  max.run = 1000,
  resultfile = NULL,
  summaryfile = "summary.txt",
  min.CFI = 0.95,
  min.TLI = 0.95,
  max.RMSEA = 0.06,
  feedbackfile = "iteration.html",
  loc.gammas,
  loc.variances,
  predictors,
  var.predictors,
  Mplus.Automation = FALSE,
  dataOut = "tempModel.dat",
  modelOut = "tempModel.inp"
)
```

Arguments

`ants`	A numeric value indicating the number of ants to send send (short forms to evaluate) per iteration. Default value is 20.
`evaporation`	A numeric value which sets the percentage of the pheremone that is retained after evaporation between steps of the algorithm. Default value is 0.9, indicating 10 (0,1), exclusive.
`mplus`	When `Mplus.Automation=FALSE`, this is a character value indicating the name of the MPlus input file without the file extension ".inp". If not in the current working directory, include the full file path where it is located. This file will be changed during the ant colony search, so it's suggested to make a backup of the original file before running the function. When `Mplus.Automation=TRUE`, this is an object of class mplusObject created by MplusAutomation and containing the initial model.
`list.items`	A list containing one or more character vectors of item names for each factor, where each factor is a separate element of the list. The items should be input in the order in which the factors are input in `i.per.f` and `factors`.
`full`	A numeric value indicating the total number of unique items in the test or scale.
`i.per.f`	A vector with number of items per factor (e.g. target number), in the same order of `list.items` and `factors`.
`factors`	A character vector with the names of the factors in the same order of `list.items` and `i.per.f`.
`steps`	A numeric value that sets the stopping rule, which is the number of ants in a row for which the model does not change.

`max.run` The maximum number of ants to run before the algorithm stops. This includes failed iterations as well. Default is 1000.

`resultfile` A character vector containing the file path where the MPlus results for the current ant model is saved. If the file is not in the current working directery, the full path must be specified. Not used when `Mplus.Automation=FALSE`.

`summaryfile` A character vector containing the name of the summary file generated. A .txt file is suggested. Default is "summary.txt" and writes into the current working directory. This file writes a line for each ant within each step and includes (a) a vector of a 0/1 value for each item indicating whether the item was selected by that ant, (b) the run number, (c) the count number, (d) the ant number, and (e) the current pheromone level.

`min.CFI` A numeric value indicating the minimum CFI for "acceptable" model fit. Models with CFI less than this value are automatically rejected. Default is 0.95.

`min.TLI` A numeric value indicating the minimum TLI for "acceptable" model fit. Models with TLI less than this value are automatically rejected. Default is 0.95.

`max.RMSEA` A numeric value indicating the maximum RMSEA for "acceptable" model fit. Models with RMSEA greater than this value are automatically rejected. Default is 0.06

`feedbackfile` A character vector containing the name of the feedback file generated. An .html file is suggested. Default is "iteration.html" and writes into the current working directory. This file saves the result of each run, which includes (a) the run number, (b) the count number, (c) the ant number, (d) the step number (if the current run is successful) or "Failure" (if the current run is unsuccessful), and for successful runs (f) the value of CFI, TLI, and RMSEA fit indices, the average of the gammas (standardized regression coefficients), and the overall variance explained of the current run.

`loc.gammas` A numeric vector with the line numbers where the regression coefficients of the MIMIC model start and end (locations). Not used with `Mplus.Automation=TRUE`

`loc.variances` A numeric vector with the line numbers of the residual variances of the latent factors. Not used with `Mplus.Automation=TRUE`

`predictors` Character vector with names of predictor variables, if any.

`var.predictors` A numeric vector with variances of the predictor(s), if any. Not used with `Mplus.Automation=TRUE`

`Mplus.Automation`
Logical. If `TRUE`, uses the `MplusAutomation` package to modify the model as the algorithm procedes. The "mplus" option will then be used as Defaults to `FALSE` as it is in the process of being built.

`dataOut` A character vector specifying the location and name of the data file generated by `MplusAutomation` for each iteration of the algorithm. Default is "tempData.dat" and saves to the current working directory. When specifying the name, be sure to use a data format that Mplus can read. You must change the working directory to the location in which this file will be saved. Only used when `Mplus.Automation=TRUE`.

`modelOut` A character vector specifying the location and name of the Mplus model file generated by `MplusAutomation` for each iteration of the algorithm. Default is

"tempModel.inp" and saves to the current working directory. When specifying the name of the model file, it must be a ".inp" extension. You must change the working directory to the location in which this file will be saved. Only used when `Mplus.Automation=TRUE`.

Details

This function sends a specified number of ants per iteration, which randomly select items to build a model, then evaluates the model based on pheromone levels. The pheromone levels are updated after each iteration according to the best-fitting model of that iteration. The algorithm's stopping rule is to end the search when a certain solution is the same for a given number of ants in a row. When constructing the mplus dataset and when `Mplus.Automation=FALSE`, make sure that items in 'categorical are' and 'usevariables' are specifications that take the same number of lines per short form.

PREPARATORY STEPS: For the ACO algorithm implementation for short for selection, the following decisions are needed:

1. Determine the target size for the short form.

2. Determine which characteristics should be optimized.

3. Define how the pheromone level will be computed: This is a function of the characteristics of the short form that will be optimized. In Leite, Huang and Marcoulides (2008), the pheromone level was zero if model fit indices did not meet Hu and Bentler's (1999) suggested thresholds, and equal to the sum of path coefficients of a predictor variable if model fit indices met thresholds. Currently, the package only implements pheromone calculation based on regression coefficients or variance explained, with user-selected model fit index thresholds.

4. Define how many short forms should be evaluated before the best-so-far pheromone level is examined. Leite, Huang and Marcoulides (2008) used 10 short forms.

5. Define the percentage of pheromone evaporation, if any. Leite, Huang and Marcoulides (2008) used 5%.

6. Define convergence criterion. Leite, Huang and Marcoulides (2008) set the algorithm to converge if the short form did not improve in 100 x number of short forms in step 4.

IMPLEMENTATION: Once these decisions are made, the ACO algorithm selects short forms with the following steps:

Step 1. All items are assigned an initial weight of 1.

Step 2. A set of n short forms is selected by sampling with probability proportional to the items' weights.

Step 3. Fit latent variable model to the n short forms.

Step 4. Calculate the pheromone levels for the n short forms. Define the best-so-far pheromone level (if iteration 1) or compare the current best pheromone from the set of n short forms to the best-so-far pheromone.

Step 5. If the pheromone level of the best short form from step 4 exceeds the best-so-far pheromone level, update the best-so-far pheromone level and add it to the current weight of the items of the best short form.

Step 6. Return to step 2 until convergence criterion is reached.

Value

A named matrix containing final model's best RMSEA, CFI, and TLI values, the final pheromone level (the mean of the standardized regression coefficients (gammas)), and a series of 0/1 values indicating the items selected in the final solution.

Author(s)

Walter Leite; Anthony W Raborn, <anthony.w.raborn@gmail.com>

References

doi:10.1080/00273170802285743

See Also

antColony

Other Ant Colony Algorithms: antColony()

Examples

```
## Not run:
# use MplusAutomation to find a 15-item short form of a simulated 56-item unidimensional test
# first, create the list of the items by the factors
# in this case, all items load onto the general 'Ability' factor
list.items <- list(c(
  "Item1", "Item2", "Item3", "Item4", "Item5",
  "Item6", "Item7", "Item8", "Item9", "Item10",
  "Item11", "Item12", "Item13", "Item14", "Item15",
  "Item16", "Item17", "Item18", "Item19", "Item20",
  "Item21", "Item22", "Item23", "Item24", "Item25",
  "Item26", "Item27", "Item28", "Item29", "Item30",
  "Item31", "Item32", "Item33", "Item34", "Item35",
  "Item36", "Item37", "Item38", "Item39", "Item40",
  "Item41", "Item42", "Item43", "Item44", "Item45",
  "Item46", "Item47", "Item48", "Item49", "Item50",
  "Item51", "Item52", "Item53", "Item54", "Item55",
  "Item56"
))
# then, load the data
data(simulated_test_data)

# Create the mplusObject with MplusAutomation
# notice the explicit call of each item, instead of the shorthand "Item1-Item56"
initial.MplusAutomation.model <- MplusAutomation::mplusObject(
  TITLE = "Trial ACO MpluAutomation with FERA 2016 Data;",
  MODEL = "Ability BY Item1 Item2 Item3 Item4 Item5
  Item6 Item7 Item8 Item9 Item10 Item11 Item12
  Item13 Item14 Item15 Item16 Item17 Item18
  Item19 Item20 Item21 Item22 Item23 Item24
  Item25 Item26 Item27 Item28 Item29 Item30
  Item31 Item32 Item33 Item34 Item35 Item36
```

```
  Item37 Item38 Item39 Item40 Item41 Item42
  Item43 Item44 Item45 Item46 Item47 Item48
  Item49 Item50 Item51 Item52 Item53 Item54
  Item55 Item56;",
  ANALYSIS = "ESTIMATOR = WLSMV;",
  VARIABLE = "CATEGORICAL = Item1 Item2 Item3 Item4 Item5
  Item6 Item7 Item8 Item9 Item10 Item11 Item12
  Item13 Item14 Item15 Item16 Item17 Item18
  Item19 Item20 Item21 Item22 Item23 Item24
  Item25 Item26 Item27 Item28 Item29 Item30
  Item31 Item32 Item33 Item34 Item35 Item36
  Item37 Item38 Item39 Item40 Item41 Item42
  Item43 Item44 Item45 Item46 Item47 Item48
  Item49 Item50 Item51 Item52 Item53 Item54
  Item55 Item56;",
  OUTPUT = "stdyx;",
  rdata = simulated_test_data
)

# finally, call the function with some minor changes to the default values.
abilityShortForm <- antcolony.mplus(
  ants = 3, evaporation = 0.7,
  mplus = initial.MplusAutomation.model, list.items = list.items, full = 56,
  i.per.f = 15, factors = "Ability", steps = 3, max.run = 50, resultfile = NULL,
  summaryfile = "C:/Users/lordmaxwell/Desktop/summary.txt",
  min.CFI = 0.95, min.TLI = 0.95, max.RMSEA = 0.06,
  feedbackfile = "C:/Users/lordmaxwell/Desktop/iteration.html", Mplus.Automation = TRUE,
  dataOut = "exampleModel.dat",
  modelOut = "exampleModel.inp"
)

## End(Not run)
```

`exampleAntModel` *Model syntax for the example in the* `antColony` *function.*

Description

A character vector containing the model syntax used for the one factor, 56-item example in the `antColony`.

Usage

```
data(exampleAntModel)
```

Format

A character vector.

modelCheck-class *An S4 class for the modelCheck object*

Description

An S4 class for the modelCheck object

Value

An S4 object of class ‘ACO‘.

Slots

`model.output` A ‘lavaan‘ object.

`warnings` A ‘character‘ vector of any warnings.

`errors` A ‘character‘ vector of any errors.

`model.syntax` A ‘character‘ vector of the modelCheck model syntax.

plot,ACO,ANY-method *Plot method for class ‘ACO‘*

Description

Plot method for class ‘ACO‘

Usage

```
## S4 method for signature 'ACO,ANY'
plot(x, y, type = c("all", "pheromone", "gamma", "beta", "variance"), ...)
```

Arguments

x, y	An S4 object of class ‘ACO‘
type	A ‘character‘ value specifying the plot type. One of ‘’all’‘ (for all plots), ‘’pheromone’‘, ‘’gamma’‘, ‘’beta’‘, or ‘’variance’‘.
...	Not used.

plot,SA,ANY-method *Plot method for class 'SA'*

Description

Plots the model fit results from the simulated annealing algorithm. Up to 8 chains are plotted. Any infinite value in the fit history (e.g. from a chain's initial model failing to fit) is coerced to 'NA', excluding it from the plot rather than letting it collapse the fit-value axis range.

Usage

```
## S4 method for signature 'SA,ANY'
plot(x, y, burn_in = 0, ...)
```

Arguments

x, y	An S4 object of class 'SA'. 'y' is included for method compatibility and is not used.
burn_in	The number of fit results, starting with the first, to discard before plotting – e.g. to exclude an unstable early period of the chain(s), as is common practice for Monte Carlo-style methods. Must be a non-negative integer less than the number of recorded steps. Default is '0' (no burn-in).
...	Not used.

plot,TS,ANY-method *Plot method for class 'TS'*

Description

Plots the objective function's value across iterations.

Usage

```
## S4 method for signature 'TS,ANY'
plot(x, y, ...)
```

Arguments

x, y	An S4 object of class 'TS'. 'y' is included for method compatibility and is not used.
...	Not used.

`refit.model` *Given a fitted lavaan model and a search table, refits the model using the search table as specifying what changes should be done (parameters fixed/freed).*

Description

This is not meant to be called explicitly as tabu.sem uses this internally for model refitting.

Usage

```
refit.model(fitted.model, ptab)
```

Arguments

`fitted.model`	fitted model of class lavaan
`ptab`	search table

Value

An object of class `lavaan` if the new model fits, or an object of class `try-error` if the model update fails.

Author(s)

Carl F. Falk

References

doi:10.1080/10705511.2017.1409074

See Also

Other Tabu Search: `add.param()`, `search.prep()`

`SA-class` *An S4 class for the Simulated Annealing Algorithm*

Description

An S4 class for the Simulated Annealing Algorithm

Value

An S4 object of class 'SA'.

Slots

`function_call` The original function call.

`chains` The number of chains used.

`chain_results` A ‘matrix‘ (for multiple chains) or a ‘list‘ (for a single chain) of the chain results.

`all_fit` A summary ‘vector‘ indicating the model fit results for each iteration.

`best_fit` The best model fit result using the selected ‘fitStatistic‘.

`best_model` A ‘modelCheck‘ object of the final solution.

`best_syntax` A ‘character‘ vector of the final solution model syntax.

`runtime` A ‘difftime‘ object of the total run time of the function.

`search.prep` *Given a fitted lavaan model (e.g., CFA), prepares a table that contains parameters that can be fixed/freed as part of a model specification search.*

Description

Given a fitted lavaan model (e.g., CFA), prepares a table that contains parameters that can be fixed/freed as part of a model specification search.

Usage

```
search.prep(fitted.model, loadings = TRUE, fcov = TRUE, errors = FALSE)
```

Arguments

`fitted.model`	- an object of class "lavaan" that contains the initially fitted model for the search
`loadings`	- a logical value that indicates whether cross-loadings will be part of the search
`fcov`	- a logical value indicating whether factor covariances will be part of the search
`errors`	- a logical value indicating whether error covariances will be part of the search

Value

A `data.frame` with lavaan-formatted parameter values.

Author(s)

Carl F. Falk

References

doi:10.1080/10705511.2017.1409074

See Also

Other Tabu Search: add.param(), refit.model()

Examples

```
## Not run:
# load simulation data and select columns used in this example
data(simulated_test_data)
tabuData <- simulated_test_data[, c(1:10)]

# specify an improper model (improper because data is unidimensional)
tabuModel <- "
Ability =~ Item1 + Item2 + Item3 + Item4
FakeAbility =~ Item5 + Item6 + Item7 + Item8
Ability ~ Outcome
FakeAbility ~ 0*Outcome"

# run the initial misspecified model for Tabu

init.model <- lavaan::lavaan(
  model = tabuModel, data = tabuData,
  auto.var = TRUE, auto.fix.first = FALSE, std.lv = TRUE, auto.cov.lv.x = TRUE
)

# Use search.prep to prepare for the Tabu search
ptab <- search.prep(fitted.model = init.model, loadings = TRUE, fcov = TRUE, errors = FALSE)

# add an additional (mispecified) parameter
additional.param <- "Item1 ~~ 0.5*Item3"
ptab <- add.param(fitted.model = init.model, ptab = ptab, syntax = additional.param)

# Perform Tabu Search
trial <- tabu.sem(init.model = init.model, ptab = ptab, obj = AIC, niter = 2, tabu.size = 5)

## End(Not run)
```

---

shortExampleAntModel *Model syntax for the short example in the antColony function.*

Description

A character vector containing the model syntax used for the one factor, 15-item, example in the antColony.

Usage

```
data(shortExampleAntModel)
```

Format

A character vector.

`ShortForm` `ShortForm` *package*

Description

Automated Item Selection Algorithms for Short Forms

Details

See the README on GitHub for more information.

`ShortFormStartup` *Create Package Startup Message*

Description

Makes package startup message.

Usage

```
ShortFormStartup()
```

Details

Idea taken from https://github.com/ntguardian/MCHT/blob/master/R/StartupMessage.R

Examples

```
ShortForm:::ShortFormStartup()
```

`show,ACO-method` *Print method for class 'ACO'*

Description

Print method for class 'ACO'

Usage

```
## S4 method for signature 'ACO'
show(object)
```

Arguments

`object`	An S4 object of class 'ACO'

show,SA-method *Print method for class 'SA'*

Description

Print method for class 'SA'

Usage

```
## S4 method for signature 'SA'
show(object)
```

Arguments

object	An S4 object of class 'SA'.

show,TS-method *Print method for class 'TS'*

Description

Print method for class 'TS'

Usage

```
## S4 method for signature 'TS'
show(object)
```

Arguments

object	An S4 object of class 'TS'.

`simulatedAnnealing` *An adaptation of the simulated annealing algorithm for psychometric models.*

Description

Simulated annealing mimics the physical process of annealing metals together. Kirkpatrick et al. (1983) introduces this analogy and demonstrates its use; the implementation here follows this demonstration closely, with some modifications to make it better suited for psychometric models.

Usage

```
simulatedAnnealing(
  initialModel,
  originalData,
  maxIterations,
  criterion = "cfi",
  temperature = "linear",
  negateCriterion = TRUE,
  Kirkpatrick = TRUE,
  randomNeighbor = TRUE,
 lavaan.model.specs = list(model.type = "cfa", auto.var = TRUE, estimator = "default",
    ordered = NULL, int.ov.free = TRUE, int.lv.free = FALSE, std.lv = TRUE,
   auto.fix.first = FALSE, auto.fix.single = TRUE, auto.cov.lv.x = TRUE, auto.th = TRUE,
    auto.delta = TRUE, auto.cov.y = TRUE),
  maxChanges = 5,
  restartCriteria = "consecutive",
  maximumConsecutive = 25,
  itemsPerFactor = NULL,
  items = NULL,
  bifactor = NULL,
  setChains = 1,
  shortForm = T,
  ...
)
```

Arguments

`initialModel`	The initial model as a `character` vector with lavaan model.syntax.
`originalData`	The original `data.frame` with variable names.
`maxIterations`	The number of iterations for which the algorithm will run.
`criterion`	Either a `character` fit-measure name recognized by fitmeasures (e.g. `"cfi"`), or a function that takes a fitted `lavaan` object and returns a single numeric value.
`temperature`	Either an acceptable `character` value, or a user-defined temperature function. The acceptable values are "linear", "quadratic", or "logistic".

`negateCriterion`
: Logical. Should the search look for the largest value of `criterion` (`TRUE`, e.g. CFI, where larger is better), or the smallest value of `criterion` (`FALSE`, e.g. RMSEA, where smaller is better)?

`Kirkpatrick`
: Either `TRUE` to use Kirkpatrick et al. (1983) acceptance probability, or a user-defined function for accepting proposed models.

`randomNeighbor`
: Either `TRUE` to use the included function for randomNeighbor selection, or a user-defined function for creating random models.

`lavaan.model.specs`
: A `list` which contains the specifications for the lavaan model. The default values are the defaults for lavaan to perform a CFA. See lavaan for more details. A partial list is accepted – any element you omit falls back to this function's default for that element – but every name you do supply must match one of the recognized element names, or the call errors.

`maxChanges`
: An `integer` value greater than 1 setting the maximum number of parameters to change within randomNeighbor. When creating a short form, should be no greater than the smallest reduction in items loading on one factor; e.g., when reducing a 2-factor scale from 10 items on each factor to 8 items on the first and 6 items on the second, maxChanges should be no greater than 2.

`restartCriteria`
: Either "consecutive" to restart after maxConsecutiveSelection times with the same model chosen in a row, or a user-defined function.

`maximumConsecutive`
: A positive `integer` value used with restartCriteria.

`itemsPerFactor`
: When creating a short form, a `vector` of the number of items per factor you want the short form to contain. Defaults to `NULL`.

`items`
: A `character` vector of candidate item names. Defaults to `NULL`, which uses all column names in `originalData`. Ignored if `itemsPerFactor` is `NULL`.

`bifactor`
: Either the name of the factor that all of the retained items will load on (as a `character` value), or `NULL` if the model is not a bifactor model. Ignored if `itemsPerFactor` is `NULL`.

`setChains`
: Numeric. Sets the number of parallel chains to run. Default to `1`, which also sets the algorithm to run serially (e.g., on a single processor). Values greater than `1` result in the chains running on parallel processes using the `doSNOW` and `foreach` packages.

`shortForm`
: Deprecated; no longer has any effect. Whether short-form (item-swap) or full-model (parameter free/fix) neighbors are generated is now determined automatically by whether `itemsPerFactor` is supplied.

`...`
: Further arguments to be passed to other functions. Not implemented for any of the included functions.

Details

Outline of the Pieces of the Simulated Annealing Algorithm

- initialModel – the initial, full form

- currentModel – the model of the current step
- maxIterations – the maximum number of steps (iterations)
- currentStep – the current step
- currentTemp – the current temperature. A function of the number of steps (such that temp = 0 at maxIterations), and values that control the shape of the overall temperature. A part of the function that determines the acceptance probability of newly – generated models
- randomNeighbor – a function that determines how the form is changed at each step. Should be able to change one or more parameters, and should have a way to control how many are changed.
- goal – a function that determines the "goodness" of the currentModel. Typically in SA goodness is defined as minimization! Sometimes called an energy function
- selectionFunction – a function that determines if a randomNeighbor change is accepted. Uses the goal function that determines the "goodness" of the currentModel and the "goodness" of the randomNeighbor, and the currentTemp to generate a probability of acceptance, then compares this probability to a Uniform(0,1) variable to determine if accepted or not. A standard version of this is: $$P(model_2|goal_1, goal_2, currentTemp) = \begin{cases} (\exp \frac{-(goal_2 - goal_1)}{currentTemp}), & (goal_1 > goal_2) \\ 1, & (goal_1 \leq goal_2) \end{cases}$$ (Kirkpatrick et al., 1983)
- bestModel – the model with the best value of the goal function achieved so far
- bestGoal – the best value of the goal function achieved so far
- restartCriteria – if utilized, this would "restart" the SA process by changing currentModel to bestModel and continuing the process. Could be based on (1) the currentStep value, (2) the difference between goal(currentModel) and goal(bestModel), (3) randomness (i.e., could randomly restart, could randomly restart based on some values, etc), (4) other criteria.

Value

An S4 object of class `SA`, with (among other slots) `best_model` holding the best `modelCheck` found, `best_fit` the corresponding fit statistic value, and `all_fit` the fit statistic values found across all iterations.

Examples

```
## Not run:
data(exampleAntModel)
data(simulated_test_data)
trial1 <- simulatedAnnealing(
  initialModel = lavaan::cfa(
    model = exampleAntModel,
    data = simulated_test_data
  ),
  originalData = simulated_test_data, maxIterations = 3,
  criterion = "rmsea", negateCriterion = FALSE
)
summary(trial1) # shows the resulting model
```

```
trial2 <- simulatedAnnealing(
  initialModel = exampleAntModel,
  originalData = simulated_test_data,
  maxIterations = 2, itemsPerFactor = 30, items = paste0("Item", 1:56)
)
summary(trial2) # shows the resulting model

## End(Not run)
```

---

simulated_test_data *A simulated data set based on a standardized test.*

Description

Simulated response patterns, abilities, and outcomes based on a uni-dimensional state-issued standardized test.

Usage

```
data(simulated_test_data)
```

Format

An object of class `data.frame` with 1000 rows and 58 columns.

Details

@format A data frame of 1000 rows (observations) and 58 columns (variables):

Outcome a binary external criterion variable correlated with TrueAbility

TrueAbility the simulated true ability parameter used to generate response patterns

Item1-Item56 binary responses to items generated using the TrueAbility parameters and simulated 3PL item parameters generated from the distribution of parameters estimated from a state-issued standardized test

---

summary,ACO-method *Summary method for class 'ACO'*

Description

Summary method for class 'ACO'

Usage

```
## S4 method for signature 'ACO'
summary(object)
```

Arguments

object	An S4 object of class 'ACO'

---

summary,SA-method *Summary method for class 'SA'*

Description

Summary method for class 'SA'

Usage

```
## S4 method for signature 'SA'
summary(object)
```

Arguments

object	An S4 object of class 'SA'.

---

summary,TS-method *Summary method for class 'TS'*

Description

Summary method for class 'TS'

Usage

```
## S4 method for signature 'TS'
summary(object)
```

Arguments

object	An S4 object of class 'TS'.

`tabu.sem` *Given a fitted lavaan model, a search table, and an objective criterion, performs a Tabu model specification search. Currently only supports neighbors that are 1 move away from the current model.*

Description

Given a fitted lavaan model, a search table, and an objective criterion, performs a Tabu model specification search. Currently only supports neighbors that are 1 move away from the current model.

Usage

```
tabu.sem(
  init.model,
  ptab,
  criterion,
  niter = 30,
  tabu.size = 5,
  negateCriterion = FALSE
)
```

Arguments

`init.model`	initial fitted model of class lavaan
`ptab`	search table (e.g., created by search.prep) that lists candidate parameters that can be modified as part of the search and how the parameters can be modified (fixed to what values)
`criterion`	The objective to be minimized (or maximized, if ‘negateCriterion = TRUE‘). Either a ‘character‘ fit-measure name recognized by fitmeasures (e.g. ‘"cfi"‘), or a function that takes a lavaan object as its sole argument and returns a numeric value.
`niter`	number of Tabu iterations to perform
`tabu.size`	size of Tabu list
`negateCriterion`	Logical. Should the search look for the smallest value of ‘criterion‘ (‘FALSE‘, e.g. AIC, where smaller is better), or the largest (‘TRUE‘, e.g. cfi, where larger is better)? Default is ‘FALSE‘.

Value

An S4 object of class ‘TS‘, with (among other slots) ‘best_fit‘ holding the best (minimal, or maximal if ‘negateCriterion = TRUE‘) objective function value achieved, ‘best_model‘ the corresponding final lavaan model, and ‘best_syntax‘ a data.frame of the lavaan-formatted parameter table for the final model.

Author(s)

Carl F. Falk

References

doi:10.1080/10705511.2017.1409074

Examples

```
# load simulation data and select columns used in this example
data(simulated_test_data)
tabuData <- simulated_test_data[, c(1:10)]

# specify an improper model (improper because data is unidimensional)
tabuModel <- "
Ability =~ Item1 + Item2 + Item3 + Item4
FakeAbility =~ Item5 + Item6 + Item7 + Item8
Ability ~ Outcome
FakeAbility ~ 0*Outcome"

# run the initial misspecified model for Tabu

init.model <- lavaan::lavaan(
  model = tabuModel, data = tabuData,
  auto.var = TRUE, auto.fix.first = FALSE, std.lv = TRUE, auto.cov.lv.x = TRUE
)

# Use search.prep to prepare for the Tabu search
ptab <- search.prep(fitted.model = init.model, loadings = TRUE, fcov = TRUE, errors = FALSE)

# Perform Tabu Search
trial <- tabu.sem(init.model = init.model, ptab = ptab, criterion = AIC, niter = 2, tabu.size = 5)
```

---

tabuSearch *Short Form Tabu Search*

Description

Given an initial (full) lavaan model string, the original data, a criterion function to minimize, and some additional specifications, performs a Tabu model specification search. Currently only supports neighbors that are 1 move away from the current model.

Usage

```
tabuSearch(
  originalData,
  initialModel,
  itemsPerFactor,
  items = NULL,
```

```
  criterion = "cfi",
  maxIterations = 20,
  tabu.size = 5,
  negateCriterion = TRUE,
 lavaan.model.specs = list(int.ov.free = TRUE, int.lv.free = FALSE, std.lv = TRUE,
  auto.fix.first = FALSE, auto.fix.single = TRUE, auto.var = TRUE, auto.cov.lv.x =
    TRUE, auto.th = TRUE, auto.delta = TRUE, auto.cov.y = TRUE, ordered = NULL,
    model.type = "cfa", estimator = "default"),
  bifactor = NULL,
  verbose = FALSE,
  parallel = T
)
```

Arguments

originalData	The original data frame with variable names.
initialModel	The initial model (typically the full form) as a character vector with lavaan model.syntax.
itemsPerFactor	A numeric vector indicating the number of items to retain for each factor.
items	A ‘character‘ vector of candidate item names. Defaults to ‘NULL‘, which uses all column names in ‘originalData‘.
criterion	The objective to optimize (minimized unless ‘negateCriterion = TRUE‘). Either a ‘character‘ fit-measure name recognized by fitmeasures (e.g. ‘"cfi"‘, the default), or a function that takes a fitted ‘lavaan‘ object and returns a single numeric value.
maxIterations	A numeric value indicating the number of iterations (model specification selections) to perform. Default is 50.
tabu.size	A numeric value indicating the size of Tabu list. Default is 5.
negateCriterion	Logical. Should the search look for the smallest value of ‘criterion‘ (‘FALSE‘, e.g. chisq or AIC, where smaller is better), or the largest (‘TRUE‘, e.g. the default cfi criterion, where larger is better)? Default is ‘TRUE‘, matching the default ‘criterion‘. Set to ‘FALSE‘ if you supply a custom ‘criterion‘ that should be minimized directly, like the built-in chisq/AIC examples below.
lavaan.model.specs	A list which contains the specifications for the lavaan model. The default values are the defaults for lavaan to perform a CFA. See lavaan for more details. A partial list is accepted – any element you omit falls back to this function's default for that element – but every name you do supply must match one of the recognized element names, or the call errors (this catches typos, e.g. ‘autovar‘ instead of ‘auto.var‘, instead of silently ignoring them).
bifactor	Either the name of the factor that all of the retained items will load on (as a ‘character‘ value), or ‘NULL‘ if the model is not a bifactor model.
verbose	Logical. If ‘TRUE‘, prints out the initial short form and the selected short form at the end of each iteration.
parallel	An option for using parallel processing. If TRUE, the function will utilize all available cores. Default is TRUE.

Value

An S4 object of class ‘TS‘, with (among other slots) ‘best_fit‘ holding the best objective function value achieved and ‘best_model‘ the corresponding final lavaan model.

Examples

```
shortAntModel <- "
Ability =~ Item1 + Item2 + Item3 + Item4 + Item5 + Item6 + Item7 + Item8
Ability ~ Outcome
"

data(simulated_test_data)
tabuResult <- tabuSearch(
  initialModel = shortAntModel,
  originalData = simulated_test_data, itemsPerFactor = 7,
  maxIterations = 1, tabu.size = 3, parallel = FALSE
)
summary(tabuResult) # shows the resulting model
## Not run:
# create simulation data from the `psych` package
# four factors, 12 items each, 48 total items
# factor loading matrix - not quite simple structure
fxMatrix <-
  matrix(
    data = c(
      rep(x = c(.9, .7, .5, .3), times = 3),
      rep(0.2, times = 3 * 4 * 3), # first factor loadings

      rep(0.2, times = 3 * 4),
      rep(x = c(.9, .7, .5, .3), times = 3),
      rep(0.2, times = 3 * 4 * 2), # second factor loadings

      rep(0.2, times = 3 * 4 * 2),
      rep(x = c(.9, .7, .5, .3), times = 3),
      rep(0.2, times = 3 * 4), # third factor loadings

      rep(0.2, times = 3 * 4 * 3),
      rep(x = c(.9, .7, .5, .3), times = 3) # fourth factor loadings
    ),
    ncol = 4
  )
# factor correlation matrix - all factors uncorrelated
PhiMatrix <-
  matrix(data = c(
    1, 0, 0, 0,
    0, 1, 0, 0,
    0, 0, 1, 0,
    0, 0, 0, 1
  ), ncol = 4)
tabuData <-
  psych::sim(
    fx = fxMatrix,
```

```
    Phi = PhiMatrix,
    n = 1000,
    raw = TRUE
  )$observed # observed is the simulated observed data

# NOTE: you must specify the model such that each factor is on a single line!
# otherwise, the algorithm will not work correctly!
tabuModel <- "
Trait1 =~ Item1 + Item2 + Item3 + Item4 + Item5 + Item6 +
Item7 + Item8 + Item9 + Item10 + Item11 + Item12
Trait2 =~ Item13 + Item14 + Item15 + Item16 + Item17 +
Item18 + Item19 + Item20 + Item21 + Item22 + Item23 + Item24
Trait3 =~ Item25 + Item26 + Item27 + Item28 + Item29 + Item30 +
Item31 + Item32 + Item33 + Item34 + Item35 + Item36
Trait4 =~ Item37 + Item38 + Item39 + Item40 + Item41 +
Item42 + Item43 + Item44 + Item45 + Item46 + Item47 + Item48
"

colnames(tabuData) <- paste0("Item", 1:48)
# specify the criterion function that the Tabu Search minimizes
# wrap this in a tryCatch in case a model does not converge!
# specify an appropriate error value: if minimizing, error value must be large
tabuCriterion <- function(x) {
  tryCatch(lavaan::fitmeasures(object = x, fit.measures = "chisq"),
    error = function(e) Inf
  )
}

# use the tabuSearch function
# reduce form to the best 12 items
tabuShort <- tabuSearch(
  initialModel = tabuModel, originalData = tabuData,
  itemsPerFactor = c(3, 3, 3, 3),
  criterion = tabuCriterion,
  # smaller chisq is better, so this should be minimized directly
  # (unlike the default cfi criterion, which should be maximized)
  negateCriterion = FALSE,
  maxIterations = 20, tabu.size = 10
)

## End(Not run)
```

TS-class *An S4 class for the Tabu Search Algorithm*

Description

An S4 class for the Tabu Search Algorithm

Value

An S4 object of class ‘TS‘.

Slots

`function_call` The original function call, with every argument resolved (whether the caller supplied it explicitly or it fell back to its default) – see ‘resolvedCall()‘. In particular, ‘negateCriterion‘ (which controls whether the search looks for the largest or smallest value of the criterion/objective function, and is used by the ‘plot‘ method to label which direction is better) is read from here rather than from a dedicated slot.

`all_fit` A summary ‘vector‘ indicating the model fit results for each iteration.

`best_fit` The best model fit result using the selected ‘fitStatistic‘. A numeric value or vector, possibly named.

`best_model` A ‘lavaan‘ object of the final solution.

`best_syntax` A ‘character‘ vector of the final solution model syntax.

`runtime` A ‘difftime‘ object of the total run time of the function.

`final_tabu_list` The final list of Tabu models. Each element of the list is a ‘lavaan‘ object.

Index