Package 'ShortForm'

October 10, 2019

Type Package

Title Automatic Short Form Creation

Version 0.4.4

Date 2019-10-9

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

RoxygenNote 6.1.1

Suggests knitr, MplusAutomation (>= 0.7), rmarkdown

Imports lavaan (>= 0.5-22), ggplot2, tidyr, stringr

Depends R (>= 3.0.0)

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

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

Encoding UTF-8

NeedsCompilation no

Author Anthony Raborn [aut, cre],
Walter Leite [aut]

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

Repository CRAN

Date/Publication 2019-10-10 04:20:02 UTC

R topics documented:

`.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")
```

`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

https://doi.org/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.lavaan` *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.lavaan(data = NULL, sample.cov = NULL, sample.nobs = NULL,
  ants = 20, evaporation = 0.9, antModel, list.items = NULL,
```

```
full = NULL, i.per.f = NULL, factors = 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.cov.lv.x =
TRUE, auto.th = TRUE, auto.delta = TRUE, auto.cov.y = TRUE, std.lv = F),
pheromone.calculation = "gamma", fit.indices = c("cfi", "tli",
"rmsea"),
fit.statistics.test = "(cfi > 0.95)&(tli > 0.95)&(rmsea < 0.06)",
summaryfile = NULL, feedbackfile = NULL, max.run = 1000,
verbose = FALSE)
```

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 pheremone that is retained after evaporation between steps of the algorithm. Default value is 0.9, indicating 10 (0,1), exclusive.
`antModel`	The lavaan formatted model. See lavaan for more details. Defaults to the default lavaan values.
`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`	Vector with number of items per factor (e.g. target number), in the same order of `list.items` and `factors`.
`factors`	Character vector with names of factors in the same order of `list.items` and `i.per.f`.
`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. See lavaan for more details.
`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.

summaryfile 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.

feedbackfile 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 chosen fit statistics (from fit.indices), the average of the gammas and betas (standardized regression coefficients), and the overall variance explained of the current run.

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

verbose An option for increasing the amount of information displayed while the function runs. If TRUE, the function will display steps, ants, counts, and current run for each attempt as well as printing "Failed iteration!" for runs that do not converge and the model fit information for runs that do converge successfully. Default is FALSE.

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 pheronomone 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

A list with four elements: the first containing a named matrix with 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, the second element containing tbe summary matrix 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, the third containing the best-fitting lavaan model object, and the fourth containing 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.lavaan(data = lavaan::HolzingerSwineford1939,
ants = 1, evaporation = 0.7,
antModel = ' visual  =~ x1 + x2 + x3
             textual =~ x4 + x5 + x6
             speed   =~ x7 + x8 + x9 ',
list.items = list(c('x1',
'x2', 'x3'), c('x4', 'x5', 'x6'), c('x7', 'x8', 'x9')), full = 9, i.per.f =
c(3,3,3), factors = c('visual','textual','speed'), steps = 1, fit.indices =
c('cfi'), fit.statistics.test = "(cfi > 0.6)", summaryfile =
NULL, feedbackfile = NULL, max.run = 2)

## 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

# then, 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'))

# load the data
data(simulated_test_data)

# finally, call the function with some minor changes to the default values.
abilityShortForm = antcolony.lavaan(data = simulated_test_data,
ants = 5, evaporation = 0.7, antModel = exampleAntModel,
list.items = list.items, full = 56, i.per.f = 20,
factors = 'Ability', steps = 3, fit.indices = c('cfi', 'rmsea'),
fit.statistics.test = "(cfi > 0.95)&(rmsea < 0.05)",
summaryfile = 'summary.txt',
feedbackfile = 'iteration.html',
max.run = 500)

abilityShortForm[[1]] # print the results of the final short form

## 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 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.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 ector 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 ector 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 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 pheronomone 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 items' weights.

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

Step 4. Calculate the pheronome 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 pheronome 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

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

https://doi.org/10.1080/00273170802285743

See Also

antcolony.lavaan

Other Ant Colony Algorithms: antcolony.lavaan

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.lavaan` *function.*

Description

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

Usage

```
exampleAntModel
```

Format

A character vector.

`plot` *Plot Functions for ShortForm Objects*

Description

These are the plot functions for the results objects created by the **ShortForm** package (objects of class `antcolony`, `tabu`, and `simulatedAnnealing`).

Usage

```
## S3 method for class 'antcolony'
plot(x, type = "all", ...)

## S3 method for class 'tabu'
plot(x, ...)

## S3 method for class 'simulatedAnnealing'
plot(x, ...)
```

Arguments

`x`	An object with one of the following classes: `antcolony`, `tabu`, or `simulatedAnnealing`.
`type`	A character string. One of "all", "pheromone", "gamma", "beta", or "variance". Matched literally. Only used with objects of class `antcolony`.
`...`	Not used with the current S3 method implementation.

Details

Objects of classes `tabu` and `simulatedAnnealing` produce a single plot which show the changes in the objective function across each iteration of the algorithm. Objects of class `antcolony` can produce up to four plots which show the changes in the pheromone levels for each item, changes in the average standardized regression coefficients of the model (gammas and betas), and changes in the amount of variance explained in the model across each iteration of the algorithm.

These functions do not currently allow users to modify the resulting plots directly, but the objects produces are **ggplot2** objects which should allow for additional user customization.

`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 (parmeters fixed/freed).*

Description

This is not meant to be called explicity 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

https://doi.org/10.1080/10705511.2017.1409074

See Also

Other Tabu Search: add.param, search.prep

---

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

https://doi.org/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.lavaan *function.*

Description

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

Usage

```
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()
```

`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, maxSteps,
  fitStatistic = "cfi", temperature = "linear", maximize = 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, maxItems = NULL, items = NULL,
  bifactor = FALSE, progress = "bar", ...)
```

Arguments

`initialModel`	The initial model as a character vector with lavaan model.syntax.
`originalData`	The original data frame with variable names.
`maxSteps`	The number of iterations for which the algorithm will run.
`fitStatistic`	Either a single model fit statistic produced by lavaan, or a user-defined fit statistic function.
`temperature`	Either an acceptable character value, or a user-defined temperature function. The acceptable values are "linear", "quadratic", or "logistic".
`maximize`	Logical indicating if the goal is to maximize (TRUE) the fitStatistic for model selection.
`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.
`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 numeric value used with restartCriteria.

`maxItems`
: 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 item names. Defaults to `NULL`. Ignored if `maxItems==FALSE`.

`bifactor`
: Logical. Indicates if the latent model is a bifactor model. If `TRUE`, assumes that the last latent variable in the provided model syntax is the bifactor (i.e., all of the retained items will be set to load on the last latent variable). Ignored if `maxItems==FALSE`.

`progress`
: Character. If `'bar'`, the function prints a progress bar indicating how far along it is. If `'text'`, prints the current step value. Otherwise, nothing is printed to indicate the progress of the function.

`...`
: 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
- maxSteps – 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 maxSteps), 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: (Kirkpatrick et al., 1983)

$$P(model_2|goal_1, goal_2, currentTemp) = \begin{cases} (\exp \frac{-(goal_2 - goal_1)}{currentTemp}), & (goal_1 > goal_2) \\ 1, & (goal_1 \leq goal_2) \end{cases}$$

- 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 critera.

Value

A named list: the 'bestModel' found, the 'bestFit', and 'allFit' values found by the algorithm.

Examples

```
## Not run:
data(exampleAntModel)
data(simulated_test_data)
trial1 <- simulatedAnnealing(initialModel = lavaan::cfa(model = exampleAntModel,
                                                        data = simulated_test_data),
                             originalData = simulated_test_data, maxSteps = 3,
                             fitStatistic = 'rmsea', maximize = FALSE)
# lavaan::summary(trial1[[1]]) # shows the resulting model

trial2 <- simulatedAnnealing(initialModel = exampleAntModel,
originalData = simulated_test_data,
maxSteps = 2, maxItems = 30, items = paste0("Item", 1:56))
# lavaan::summary(trial2[[1]]) # 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 unidimensional state-issued standardized test.

Usage

```
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 reponses 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

---

`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, obj, niter = 30, tabu.size = 5)
```

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)
`obj`	objective function to be MINIMIZED. Any function that takes a lavaan object as the sole argument and returns a numeric value can be used.
`niter`	number of Tabu iterations to perform
`tabu.size`	size of Tabu list

Value

A list with three elements: best.obj, the numerical value of the best (minimal) objective function achieved; best.mod, the final lavaan model, and best.binvec, a data.frame of the lavaan-formatted parameter table for the final model.

Author(s)

Carl F. Falk

References

https://doi.org/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, obj = AIC, niter = 2, tabu.size = 5)
```

---

tabuShortForm *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

```
tabuShortForm(initialModel, originalData, numItems, allItems,
  criterion = function(x) lavaan::fitmeasures(object = x, fit.measures =
  "rmsea"), niter = 30, tabu.size = 5,
  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 = FALSE)
```

Arguments

initialModel	The initial model (typically the full form) as a character vector with lavaan model.syntax.

`originalData` The original data frame with variable names.

`numItems` A numeric vector indicating the number of items to retain for each factor.

`allItems` For unidimensional models, a character vector of the item names. For multifactor models, a list of the item names, where each element of the list is a factor.

`criterion` A function calculating the objective criterion to minimize. Default is to use the built-in 'rmsea' value from 'lavaan::fitmeasures()'.

`niter` 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.

`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.

`bifactor` Logical. Indicates if the latent model is a bifactor model. If 'TRUE', assumes that the last latent variable in the provided model syntax is the bifactor (i.e., all of the retained items will be set to load on the last latent variable).

Value

A named list with the best value of the objective function ('best.obj') and the best lavaan model object ('best.mod').

Examples

```
shortAntModel = '
Ability =~ Item1 + Item2 + Item3 + Item4 + Item5 + Item6 + Item7 + Item8
Ability ~ Outcome
'
data(simulated_test_data)
tabuResult <- tabuShortForm(initialModel = shortAntModel,
                            originalData = simulated_test_data, numItems = 7,
                            allItems = colnames(simulated_test_data)[3:11],
                            niter = 1, tabu.size = 3)
lavaan::summary(tabuResult$best.mod) # shows the resulting model
```

Index