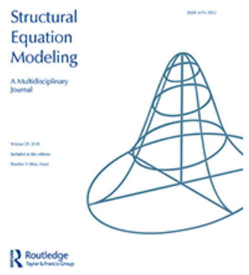

Structural Equation Modeling: A Multidisciplinary Journal

ISSN: 1070-5511 (Print) 1532-8007 (Online) Journal homepage: https://www.tandfonline.com/loi/hsem20

# Model Specification Searches in Structural Equation Modeling with *R*

Katerina M. Marcoulides & Carl F. Falk

**To cite this article:** Katerina M. Marcoulides & Carl F. Falk (2018) Model Specification Searches in Structural Equation Modeling with *R*, Structural Equation Modeling: A Multidisciplinary Journal, 25:3, 484-491, DOI: 10.1080/10705511.2017.1409074

**To link to this article:** https://doi.org/10.1080/10705511.2017.1409074

Published online: 08 Jan 2018.

*Structural Equation Modeling: A Multidisciplinary Journal*, 25: 484–491, 2018
Copyright © Taylor & Francis Group, LLC
ISSN: 1070-5511 print / 1532-8007 online
DOI: https://doi.org/10.1080/10705511.2017.1409074

# Model Specification Searches in Structural Equation Modeling with *R*

Katerina M. Marcoulides[1] and Carl F. Falk[2]
[1]*University of Florida*
[2]*McGill University*

This article introduces and demonstrates the application of an R statistical programming environment code for conducting structural equation modeling (SEM) specification searches. The implementation and flexibility of the provided code is demonstrated using the Tabu search procedure, although the underlying code can also be directly modified to implement other search procedures like Ant Colony Optimization, Genetic Algorithms, Ruin-and-Recreate, or Simulated Annealing. The application is illustrated using data with a known common factor structure. The results demonstrate the capabilities of the program for conducting specification searches in SEM. The programming codes are provided as open-source R functions.

**Keywords**: specification search, structural equation modeling, Tabu search

Research in exploratory data mining over the past few decades has focused on the development and application of computerized algorithms capable of finding different hidden patterns and structures from data (McArdle & Ritschard, 2014). With the sudden increase in "big data" repositories, exploratory data mining has become a very hot field in many scientific areas, especially in the behavioral, educational, and social sciences. A general categorization used to describe these countless data-mining algorithms denotes them as "supervised," "semi-supervised," or "unsupervised" (Marcoulides & Leite, 2014). With supervised algorithms, the exploration of the data is guided by prior knowledge about the studied variables. With unsupervised algorithms, the exploration builds knowledge based exclusively on the inspected data (which is the reason they are also sometimes referred to as machine learning algorithms). Semi-supervised algorithms are halfway between supervised and unsupervised exploration, as the algorithms are initially provided with some supervised information (i.e., some preliminary knowledge about the studied variables), which is then extended based on the inspected data.

Exploratory data-mining algorithms have also received significant attention in the structural equation modeling (SEM) literature (e.g., Marcoulides & Drezner, 2001, 2003; Marcoulides, Drezner, & Schumacker, 1998; Marcoulides & Ing, 2012; Marcoulides & Leite, 2014; Scheines, Spirtes, Glymour, Meek, & Richardson, 1998). A phrase that is often used to describe these activities within SEM is *conducting a specification search* (Long, 1983). The basic idea behind conducting a specification search is to find a better fitting model after comparing distinct SEM models using a chi-square difference test, information criterion (e.g., Akaike information criteria (AIC) or Bayesian information criteria (BIC)), or other selected fit indices (Akaike, 1987). Although researchers have indicated that specification searches are useful for detecting and correcting any potential specification errors between an initially proposed model and the true model describing the population and variables under study, they have also cautioned users to acknowledge that such activities are purely exploratory in nature. As a consequence, exploratory specification searches in SEM can capitalize on chance and require some form of cross-validation before their validity can be claimed (Kaplan, 1988; MacCallum, 1986; Marcoulides & Leite, 2012; Whittaker & Marcoulides, 2007).

Correspondence should be addressed to Katerina M. Marcoulides, University of Florida, Research and Evaluation Methodology, 1215 Norman Hall, PO Box 117049, Gainesville, FL, 32611, USA. E-mail: k.marcoulides@coe.ufl.edu

Rudimentary forms of specification searches in SEM may consist of manually fitting alternative models and comparing fit, with model changes based on substantive theory, modification indices to determine which parameters to free (also known as score tests or Lagrange Multiplier tests; Sörbom, 1989), Wald-based tests to determine which parameters to fix, or some synthesis of these approaches. Such approaches are arguably useful to the extent that misspecification is not too great and theoretical considerations help guide model modification (Green & Thompson, 2010). Most existing commercial SEM programs such as AMOS (Arbuckle, 2012), EQS (Bentler, 2004), LISREL (Jöreskog & Sörbom, 2005), and M*plus* (Muthén & Muthén, 2014) provide researchers with these options for conducting specification searches. However, such basic search strategies are impractical for large models where there are many possible modifications since manually refitting models is time-consuming and will not do a decent job of traversing the complete model parameter space. Even consideration of all possible combinations of theoretically interesting model modifications may require a very large number of refitted models. In such cases, a specification search in SEM is better viewed as a combinatorial optimization problem whereby the best-fitting model is selected based on the evaluated goodness of fit of that model compared to any number of others.

*Metaheuristics* consist of a collection of algorithms that can find good solutions to such optimization problems in a reasonable amount of computational time (Bianchi, Dorigo, Gambardellal, & Gutjahr, 2009; Marcoulides & Ing, 2012). Such algorithms include ant colony optimization (Doringo & Stützle, 2004; Marcoulides & Drezner, 2002), genetic algorithms (Holland, 1975; Marcoulides & Drezner, 2001), Tabu search (Glover, 1986; Marcoulides et al., 1998), and simulated annealing (Černý, 1985; Kirkpatrick, Gelatt, & Vecchi, 1983; Marcoulides & Drezner, 1999). To date, many of these and other specification search algorithms have been proposed in the SEM literature along with specialized software (Leite, Haung, & Marcoulides, 2008; Leite & Marcoulides, 2009; Marcoulides, 2009; Marcoulides & Drezner, 1999; 2002; 2003; Marcoulides et al., 1998). Unfortunately, many of these implementations consisted of FORTRAN-based programs that are not user friendly and require problem- and model-specific construction and programming design.

This article introduces and demonstrates the application of programming code in the R statistical programming environment (R Development Core Team, 2011)for conducting SEM specification searches. Using data with known structure, the implementation and flexibility of the code is illustrated using Tabu search, although the underlying programming code can be readily modified to implement other search procedures like ant colony optimization, genetic algorithms, ruin-and-recreate, or simulated annealing. To implement the SEM estimation routines, we use the popular R package *lavaan* (Rosseel, 2012) for model estimation, which can handle such characteristics as continuous and ordinal data, incomplete data, and multiple group analysis. The choice of *lavaan* was based primarily on flexibility, ease of use, native integration within R, and popularity. That is, other native R-based options are possible (e.g., *OpenMx*; Boker et al., 2011) as well as packages within R that can execute code from commercial software packages (e.g., MplusAutomation – see Hallquist & Wiley, 2017).

Although the underlying programming code is able to use various fit criteria to comparatively evaluate the selection of the final model (e.g., AIC, the BIC, the chi-square/degree of freedom ratio ($\chi^2$/df), and the root mean square error of approximation (RMSEA) indices), past research has shown that the BIC is often best at finding a true model when one exists (e.g., Leite & Marcoulides, 2009; Marcoulides & Leite, 2014). For this reason, specification search results using only the BIC criterion are reported in this article. Evaluation of the examined models can of course also be supplemented using other available fit indices, for example, using the newly proposed equivalence testing approach (Marcoulides & Yuan, 2017), which has been recommended as a potentially superior way to evaluate the goodness of fit of models (Yuan, Chan, Marcoulides, & Bentler, 2016).

The code also makes specific use of converting any current model specification into a binary vector (0 and 1) as originally suggested by Marcoulides and Drezner (2003). This binary vector essentially details what parameter(s) are free (1) versus fixed (0) in a specified model. This binary vector is useful for conducting supervised, semi-supervised, or unsupervised specification searches. The example provided here details the use of such code for *lavaan* and for a single-group confirmatory factor analysis (CFA) model based on a semi-supervised approach. Further extensions (e.g., more general structural models or even multiple group models) are detailed throughout and in the discussion.

## EXAMPLE ANALYSIS

To illustrate, a simple confirmatory factor analytic model based on $p = 10$ observed variables ($y$) with $k = 2$ common factors ($\eta$) and a specified structure is tested against simulated data. That is, the true covariance matrix, $\Sigma$, has the following structure: $\Sigma = \Lambda\Phi\Lambda' + \Psi$, where $\Lambda$ is a $p \times k$ matrix of factor loadings, $\Phi$ $(k \times k)$ contains factor covariances, and $\Psi$ is a $p \times p$ diagonal matrix of residual covariances. The dataset was simulated from a multivariate normal distribution and based on $N = 500$ cases generated according to the following population model:

$$\Lambda = \begin{bmatrix} .5 & 0 \\ .7 & 0 \\ 0 & .7 \\ .8 & 0 \\ .7 & 0 \\ 0 & .3 \\ 0 & .4 \\ 0 & .7 \\ .9 & 0 \\ 0 & .5 \end{bmatrix}, \Phi = \begin{bmatrix} 1 & \\ .3 & 1 \end{bmatrix}$$

with the diagonal of $\Psi$ chosen such that the observed variables have unit variance. Based on the simulated data and true model, $\chi^2(34) = 37.27, p = .32$, $AIC = 13000.99$, $BIC = 13089.49$, CFI = .997, SRMR = .028, RMSEA = .0138, 90% CI RMSEA = [0.00, 0.036].

Although many specification errors can be made in the above factor model, for illustrative purposes suppose the following misspecified factor loading matrix ($\Lambda$) containing two incorrect loadings was hypothesized:

$$\Lambda = \begin{bmatrix} \lambda_{11} & 0 \\ \lambda_{21} & 0 \\ \lambda_{31} & 0 \\ \lambda_{41} & 0 \\ \lambda_{51} & 0 \\ 0 & \lambda_{62} \\ 0 & \lambda_{72} \\ 0 & \lambda_{82} \\ 0 & \lambda_{92} \\ 0 & \lambda_{102} \end{bmatrix}$$

Based on this misspecified model, $\chi^2(34) = 223.75, p < .001$, $AIC = 13187.47$, $BIC = 13275.98$, CFI = .797, SRMR = .09, RMSEA = .106, 90% CI RMSEA = [0.93, 0.119]. Thus, the misspecified model does not fit the data and exhibits poor fit with respect to fit indices.

To load the R code and necessary functions, use the following command within R:

```
library(lavaan)
source("functions.R")
```

where “functions.R” contains the relevant R code and functions that will be described shortly.[1] The code was written and tested using *lavaan* version 0.5–23.1097. This R file contains search.prep, add.param, rm.param, nullval.param, free.param, fix.param, and tabu.sem functions, which will be illustrated shortly.

Next, the initial provided misspecified model is declared and fit using the following programming code:

```
mod<-'
F1 =~ Y1 + Y2 + Y3 + Y4 + Y5
F2 =~ Y6 + Y7 + Y8 + Y9 + Y10
'
fit.start <- lavaan(mod, data=dat,
auto.var=TRUE, auto.fix.first=FALSE,
std.lv=TRUE,auto.cov.lv.x=TRUE)
```

## SPECIFICATION TABLE

In order to proceed with a specification search, the proposed model is first converted to a binary vector using the search.prep function. This is a specialized function intended for CFA models. The following function command (with a comment statement) performs this activity:

```
 # prepare for specification search
 spec.table<-search.prep(fit.start,
loadings=TRUE,fcov=TRUE,errors=FALSE)
```

We note that the arguments in this function (loadings, fcov, and errors) can be modified to extract different types of model parameters. In the illustrated example, we are only interested in factor loadings and the factor covariance. This function command outputs the following:

```
spec.table

#          lhs    op    rhs    block    free    label    nullval
##    1     F1    ~~     F2        1       1                   0
##    2     F1    =~     Y1        1       1                   0
##    3     F1    =~     Y2        1       1                   0
##    4     F1    =~     Y3        1       1                   0
##    5     F1    =~     Y4        1       1                   0
##    6     F1    =~     Y5        1       1                   0
##    7     F1    =~     Y6        1       0                   0
##    8     F1    =~     Y7        1       0                   0
##    9     F1    =~     Y8        1       0                   0
##    10    F1    =~     Y9        1       0                   0
##    11    F1    =~    Y10        1       0                   0
##    12    F2    =~     Y1        1       0                   0
##    13    F2    =~     Y2        1       0                   0
##    14    F2    =~     Y3        1       0                   0
##    15    F2    =~     Y4        1       0                   0
##    16    F2    =~     Y5        1       0                   0
##    17    F2    =~     Y6        1       1                   0
##    18    F2    =~     Y7        1       1                   0
##    19    F2    =~     Y8        1       1                   0
##    20    F2    =~     Y9        1       1                   0
##    21    F2    =~    Y10        1       1                   0
```

The generated results form a table that we will refer to as a *specification table* that is used for the specification search, and is similar to how *lavaan* internally represents model parameters in the form of a *parameter table*. Users can gain some intuition about this representation from documentation on the lavParTable command. The model is represented as a series of equations with a left-hand-side (lhs), operator (op), and right-hand-side (rhs) —the combination of which resembles *lavaan*'s model

[1] The code is available upon request from the authors and at the second author's website: http://www.psych.mcgill.ca/perpg/fac/falk/research.html.

syntax. For example, “F1 ~~ F2” is the covariance between latent traits, “F1 =~ X1” is a loading for X1 on factor 1, and so on. Additional columns include a “block” column that could indicate parameters that belong to different groups in a multiple group model, the binary vector “free” corresponding to whether the parameter is free (1) or fixed (0), and labels for the parameters (if these were specified by the user as part of the model syntax; here it is blank). Finally, the last column “nullval” is numeric and determines what value the parameter would be fixed to if we specify it to be fixed. In this example, fixing any parameter defaults to those parameters being fixed to 0, which is the typical case for factor loadings and factor covariances (and error covariances) in CFA models.

We note that the above table could also be further modified before embarking on a specification search and several utility functions are provided to assist with this matter. For example, if we do not want to explore the possibility of fixing or freeing a particular loading, we could remove it from the above table. That is, the table represents only those parameter(s) that could be used in the specification search; other parameter(s) in the model will remain untouched. The function rm.param accomplishes this task. The arguments are the specification table, followed by a string that resembles *lavaan* model syntax. For example, if we wanted to remove the third item loading on F1, we could invoke the following command:

```
  spec.table<-rm.param(spec.table,
"F1 =~ Y3")
```

The result is another specification table, and here we overwrite spec.table. The generated specification table is now as follows (with the loading in line ##4 from the previous specification table removed):

```
spec.table

##        lhs   op   rhs   block   free   label   nullval
##     1  F1    ~~   F2        1      1                 0
##     2  F1    =~   Y1        1      1                 0
##     3  F1    =~   Y2        1      1                 0
##     5  F1    =~   Y4        1      1                 0
##     6  F1    =~   Y5        1      1                 0
##     7  F1    =~   Y6        1      0                 0
##     8  F1    =~   Y7        1      0                 0
##     9  F1    =~   Y8        1      0                 0
##    10  F1    =~   Y9        1      0                 0
##    11  F1    =~   Y10       1      0                 0
##    12  F2    =~   Y1        1      0                 0
##    13  F2    =~   Y2        1      0                 0
##    14  F2    =~   Y3        1      0                 0
##    15  F2    =~   Y4        1      0                 0
##    16  F2    =~   Y5        1      0                 0
##    17  F2    =~   Y6        1      1                 0
##    18  F2    =~   Y7        1      1                 0
##    19  F2    =~   Y8        1      1                 0
##    20  F2    =~   Y9        1      1                 0
##    21  F2    =~   Y10       1      1                 0
```

Other parameter(s) could also be added to the specification table. For example, if we wished to add an error covariance, and wished that this error covariance had a null value of 0 when fixed, we can invoke the following command:

```
   spec.table<-add.param(fit.start,
spec.table, "Y1 ~~ 0*Y2")
```

The result is another specification table as follows (with line ##22 now added to the previous specification table):

```
spec.table

##        lhs   op   rhs   block   free   label   nullval
##     1  F1    ~~   F2        1      1                 0
##     2  F1    =~   Y1        1      1                 0
##     3  F1    =~   Y2        1      1                 0
##     5  F1    =~   Y4        1      1                 0
##     6  F1    =~   Y5        1      1                 0
##     7  F1    =~   Y6        1      0                 0
##     8  F1    =~   Y7        1      0                 0
##     9  F1    =~   Y8        1      0                 0
##    10  F1    =~   Y9        1      0                 0
##    11  F1    =~   Y10       1      0                 0
##    12  F2    =~   Y1        1      0                 0
##    13  F2    =~   Y2        1      0                 0
##    14  F2    =~   Y3        1      0                 0
##    15  F2    =~   Y4        1      0                 0
##    16  F2    =~   Y5        1      0                 0
##    17  F2    =~   Y6        1      1                 0
##    18  F2    =~   Y7        1      1                 0
##    19  F2    =~   Y8        1      1                 0
##    20  F2    =~   Y9        1      1                 0
##    21  F2    =~   Y10       1      1                 0
##    22  Y1    ~~   Y2        1      0                 0
```

Note that the initially fitted model (fit.start) is required as the first argument, followed by the specification table and *lavaan* model syntax for the added parameter. Use of the initially fitted model allows the underlying code to check whether any added parameters are involved in (in)equality constraints and will generate an appropriate warning if such parameters are used as part of a specification search. In addition, note the starting values in the model syntax used for add.param are used for the “nullval” column. If the model of interest actually contains a structural model, the programming code should also be able to handle regression paths as part of the specification search (e.g., “F1 ~ F3”). However, this latter feature of the programming code has not yet been fully tested in conjunction with a specification search.

Finally, if when fixing a parameter, we want it fixed to something other than 0, we could also accomplish this by modifying the last column “nullval.” This can be done manually, or by invoking nullval.param. An example is given for the loading of item 5 on the first factor, though fixing this value to 5 may not always make theoretical sense:

```
    spec.table<-nullval.param(spec.table,
"F1 =~ Y2", .5)
```

This results in the following specification table (with line ##3 now modified):

```
  spec.table

  ##          lhs    op    rhs    block    free    label nullval
  ##    1      F1    ~~     F2        1       1              0.0
  ##    2      F1    =~     Y1        1       1              0.0
  ##    3      F1    =~     Y2        1       1              0.5
  ##    5      F1    =~     Y4        1       1              0.0
  ##    6      F1    =~     Y5        1       1              0.0
  ##    7      F1    =~     Y6        1       0              0.0
  ##    8      F1    =~     Y7        1       0              0.0
  ##    9      F1    =~     Y8        1       0              0.0
  ##    10     F1    =~     Y9        1       0              0.0
  ##    11     F1    =~    Y10        1       0              0.0
  ##    12     F2    =~     Y1        1       0              0.0
  ##    13     F2    =~     Y2        1       0              0.0
  ##    14     F2    =~     Y3        1       0              0.0
  ##    15     F2    =~     Y4        1       0              0.0
  ##    16     F2    =~     Y5        1       0              0.0
  ##    17     F2    =~     Y6        1       1              0.0
  ##    18     F2    =~     Y7        1       1              0.0
  ##    19     F2    =~     Y8        1       1              0.0
  ##    20     F2    =~     Y9        1       1              0.0
  ##    21     F2    =~    Y10        1       1              0.0
  ##    22     Y1    ~~     Y2        1       0              0.0
```

Two additional utility functions are useful for changing the status of a parameter in the specification table from fixed to free, and vice versa. In particular, free.param and fix.param are illustrated in conjunction with a model refitting function in the following section.

## MODEL REFITTING

The function refit.model enables one to use *lavaan* and refit the model after changing the aspects of the specification table. In conjunction with the above commands, this function could be used for a supervised specification search. For example, if a particular factor loading was freed (e.g., if X6 regressed on F1), and refitting the model with *lavaan* could be accomplished using the commands below. The command free.param takes the specification table and *lavaan* syntax as an argument, and returns the specification table with the "free" column changed from 0 to 1 for the specified parameter. The command refit.model then only requires two arguments: the initially fitted model and the updated specification table as follows:

```
    # re-obtain the originally declared
specification table
    spec.table<-search.prep(fit.start,
loadings=TRUE,fcov=TRUE,errors=FALSE)
    # free a loading
    new.table<-free.param(spec.table,
"F1 =~ Y6")
    # re-fit model
    fit.1<-refit.model(fit.start,new.table)
```

Here refit.model returns a fitted *lavaan* model. The user may then inspect model fit, modification indices, Wald-based tests, and so on (e.g., "summary(fit.1, fit.measures = TRUE)") to determine whether the model is acceptable or whether additional modifications are desired.

Alternatively, if we wanted to change a value from free to fixed, it can be accomplished with the fix.param command, which essentially has the same syntax as free.param:

```
    new.table2<-new.table
    new.table2<-fix.param(new.table,
"F2 =~ Y6")
    fit.2<-refit.model(fit.start,new.table2)
# re-fit model
```

Importantly, the mere existence of a command such as refit.model allows automated model refitting based on changes to the specification table. There is no need to manually re-create model syntax and re-run the *lavaan* or cfa command. In the context of a semi-supervised search, this allows the underlying algorithm to only need to change the binary vector that represents the space for the specification search (i.e., the "free" column of the specification table), refit the model, and extract the desired information about model fit.

To perform a semi-supervised specification search, we illustrate use of a Tabu algorithm with the tabu.sem function. This search requires an objective criterion by which to judge alternative models. The "obj" argument requires a function that can be used on a fitted *lavaan* model and that returns the value of the objective function. For instance, "AIC" and "BIC" could be passed to this function because "AIC(fit.start)" and "BIC(fit.start)" will return the AIC and BIC values of "fit.start" or any other *lavaan* model passed to these functions. To perform a specification search based on the BIC fit criterion, the following commands can be used:

```
    search.result<-tabu.sem(fit.start,
spec.table,obj=BIC,niter=30,tabu.size=5)
```

This function takes as arguments the initially fitted model, the desired specification table for the search, the objective function to be minimized, the number of iterations (niter), and size of the Tabu list (tabu.size). Put succinctly, each iteration of the Tabu algorithm examines all neighboring models and selects the best-fitting model based on the objective criterion as the starting point for the next iteration. A Tabu list is maintained that contains models that have already been examined and were not a better fitting model. Moves are not allowed to neighbors that are currently on the Tabu list. Use of this list allows the algorithm to effectively search the model parameter space in areas that might be away from a currently best-fitted model as well as escaping loops. In contrast, an AIC or BIC stepwise approach that employs both backward and forward changes to the model could get stuck at a single suboptimal solution that has no better fitting neighbors. Readers are referred elsewhere for detailed advice on the choice of number of iterations and list size (e.g., Marcoulides et al., 1998). We note finally that tabu.sem currently only allows for neighbors that involve a single change of the binary vector, yet this feature could be modified in later versions of the code.

The tabu.sem function returns a list that contains the specification table corresponding to the binary vector that had the best fit (best.binvec), the value of the objective function for this best model (best.obj), and the fitted *lavaan* model (best.mod). Based on the above displayed commands, the generated output results in the following specification table (that corresponds to the original specified population model) and provides the fit criterion of BIC = 13,089.49.

```
search.result$best.binvec

##        lhs    op    rhs    block    free    label    nullval
##    1   F1     ~~    F2         1       1                   0
##    2   F1     =~    Y1         1       1                   0
##    3   F1     =~    Y2         1       1                   0
##    4   F1     =~    Y3         1       0                   0
##    5   F1     =~    Y4         1       1                   0
##    6   F1     =~    Y5         1       1                   0
##    7   F1     =~    Y6         1       0                   0
##    8   F1     =~    Y7         1       0                   0
##    9   F1     =~    Y8         1       0                   0
##   10   F1     =~    Y9         1       1                   0
##   11   F1     =~    Y10        1       0                   0
##   12   F2     =~    Y1         1       0                   0
##   13   F2     =~    Y2         1       0                   0
##   14   F2     =~    Y3         1       1                   0
##   15   F2     =~    Y4         1       0                   0
##   16   F2     =~    Y5         1       0                   0
##   17   F2     =~    Y6         1       1                   0
##   18   F2     =~    Y7         1       1                   0
##   19   F2     =~    Y8         1       1                   0
##   20   F2     =~    Y9         1       0                   0
##   21   F2     =~    Y10        1       1                   0
```

The details of the best model could also be inspected in detail by using the command:

```
summary(search.results$best.mod)
```

Alternatively, custom objective functions can also easily be written in an attempt to find a model with the smallest RMSEA value:

```
obj.func<-function(x){fitMeasures(x)
["rmsea"]}
```

Since tabu.sem seeks minimization, or the smallest values of this objective function, a search looking to maximize CFI could use the following function instead (note use of the negative sign):

```
obj.func<-function(x){-fitMeasures(x)
["cfi"]}
```

Functions of fit parameters, or any other function of quantities that could be extracted from a fitted *lavaan* model, could also be used. To then perform a search, "obj.func" would be used in place of "BIC":

```
tabu.sem(fit.start,spec.table,
obj=obj.func,niter=30,tabu.size=5)
```

We note that these examples represent a small fraction of the different options that can be included when fitting a model.

## DISCUSSION

This article introduced and demonstrated the application of newly developed R programming code for conducting SEM specification searches. The code is provided as open-source software and can be readily extended by the R community to include other search procedures like ant colony optimization, genetic algorithms, ruin-and-recreate, or simulated annealing. Although only results using the BIC as the optimization criterion were presented in this article, other fit criteria can also be easily utilized to try and find the best-fitting model. More importantly, however, it is essential to keep in mind that such computerized specification searches can capitalize on chance. As a consequence, it is highly recommended that all final generated models be cross-validated before any aspect of their validity can be claimed. At the very least, specification search algorithms can be effective as initial exploratory tools that can be used in modeling complex behavioral, educational, and social phenomena.

In the context of attempting to refit many alternative models, the underlying code may not always know what model(s) are identified and might inevitably in the process attempt to refit underidentified models. Any model that does not yield convergence and/or does not have valid standard errors is automatically not considered as a candidate model by tabu.sem. As a by-product, however, this may sometimes result in the generation of a large number of warning messages. Currently, these are not suppressed, although it is anticipated that future versions of the programming code could allow this.

Although above we demonstrated a model in which the factor variances were fixed to 1 and all loadings were estimated, researchers may alternatively wish to use a marker or reference variable to identify each factor so that factors are not inadvertently interchanged (e.g., as may similarly occur in the context of exploratory factor analysis simulations and/or bootstrapping). The current behavior of search.prep is to omit from the specification table those loadings that are fixed to nonzero values. Thus, if such a marker variable strategy is used for the initially fitted model, search.prep should automatically generate an appropriate binary vector and specification table.

The underlying code presented here can also work in the context of multiple group models and with models that have (in)equality constraints; however, a warning is appropriate for these cases. First, the specification table and Tabu search algorithm implemented here are not yet designed to handle cases where model constraints are part of the specification search. For instance, the underlying code is not yet prepared to handle cases where part of the search requires that two parameters are either (a) constrained equal or (b) freely estimated as two separate parameters. Such a case is common in multiple group models with across-group constraints, and additional coding work is required for such features to be possible. Although constraints can also be translated into a binary vector, it may be difficult to simultaneously employ a search for different constraints at the same time that involved parameter(s) that may be fixed or free. In a similar manner, search.prep and add.param will generate warnings if the code detects that any parameters in the specification table are involved in (in)equality constraints, but our exploratory tool does not yet contain an automated way of translating all fixed versus free parameters into a binary vector.

## REFERENCES

Akaike, H. (1987). Factor analysis and AIC. *Psychometrika*, *52*, 317–332. doi:10.1007/BF02294359

Arbuckle, J. L. (2012). *IBM SPSS AMOS 21 user's guide*. Chicago, IL: AMOS Development Corporation.

Bentler, P. M. (2004). *EQS structural equation program manual*. Encino, CA: Multivariate Software Inc.

Bianchi, L., Dorigo, M., Gambardella, L. M., & Gutjahr, W. J. (2009). A survey on metaheuristics for stochastic combinatorial optimization. *Natural Computing*, *8*, 239–287. doi:10.1007/s11047-008-9098-4

Boker, S., Neale, M., Maes, H., Wilde, M., Spiegel, M., Brick, T., … Fox, J. (2011). OpenMx: An open source extended structural equation modeling framework. *Psychometrika*, *76*, 306–317. doi:10.1007/s11336-010-9200-6

Černý, V. (1985). Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, *45*, 41–51. doi:10.1007/BF00940812

Doringo, M., & Stützle, T. (2004). *Ant colony optimization*. Cambridge, MA: MIT Press.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computer & Operations Research*, *13*, 533–549. doi:10.1016/0305-0548(86)90048-1

Hallquist, M., & Wiley, J. (2017). M*plus*Automation: An R package for facilitating large-scale latent variables analysis in Mplus. *Structural Equation Modeling*. Advance online publication. doi:10.1080/10705511.2017.1402334

Holland, J. H. (1975). *Adaptation in natural and artificial systems*. Ann Arbor, MI: University of Michigan Press.

Jöreskog, K. G., & Sörbom, D. (2005). *LISREL user's reference guide*. Chicago, IL: Scientific Software International.

Kaplan, D. (1988). The impact of specification error on the estimation, testing, and improvement of structural equation models. *Multivariate Behavioral Research*, *23*, 69–86. doi:10.1207/s15327906mbr2301_4

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, *220*, 671–680. doi:10.1126/science.220.4598.671

Leite, W. L., Huang, I.-C., & Marcoulides, G. A. (2008). Item selection for the development of short forms of scales using an ant colony optimization algorithm. *Multivariate Behavioral Research*, *43*(3), 411–431. doi:10.1080/00273170802285743

Leite, W. L., & Marcoulides, G. A. (2009, April). *Using ant colony optimization algorithm for specification searches: A comparison of criteria*. Paper presented at the Annual Meeting of the American Educational Research Association, San Diego, CA.

Long, J. S. (1983). *Covariance structure models: An introduction to LISREL*. Beverly Hills, CA: Sage Publications.

MacCallum, R. (1986). Specification searches in covariance structure modeling. *Psychological Bulletin*, *100*(1), 107–120. doi:10.1037/0033-2909.100.1.107

Marcoulides, G. A. (2009, June). *Conducting specification searchers in SEM using a ruin-and- recreate principle*. Paper presented at the Annual Meeting of the American Psychological Society, San Francisco, CA.

Marcoulides, G. A., & Drezner, Z. (1999). Using simulated annealing for model selection in multiple regression analysis. *Multiple Linear Regression Viewpoints*, *25*, 1–4.

Marcoulides, G. A., & Drezner, Z. (2001). Specification searches in structural equation modeling with a genetic algorithm. In G. A. Marcoulides, & R. E. Schumacker (Eds.), *Advanced structural equation modeling: New developments and techniques*. Mahwah, NJ: Lawrence Erlbaum Associates, Inc.

Marcoulides, G. A., & Drezner, Z. (2002). A model selection approach for the identification of quantitative trait loci in experimental crosses – Discussion on the paper by Broman and speed. *Journal of the Royal Statistical Society, Series B*, *64*, 754.

Marcoulides, G. A., & Drezner, Z. (2003). Model specification searches using ant colony optimization algorithms. *Structural Equation Modeling*, *10*, 154–164. doi:10.1207/S15328007SEM1001_8

Marcoulides, G. A., Drezner, Z., & Schumacker, R. E. (1998). Model specification searches in structural equation modeling using tabu search. *Structural Equation Modeling*, *5*, 365–376. doi:10.1080/10705519809540112

Marcoulides, G. A., & Ing, M. (2012). Automated structural equation modeling strategies. In R. Hoyle (Ed.), *Handbook of structural equation modeling*. New York, NY: Guilford Press.

Marcoulides, G. A., & Leite, W. (2014). Exploratory data mining algorithms for conducting searches in structural equation modeling: A comparison of some fit criteria. In J. J. McArdle, & G. Ritschard (Eds.), *Contemporary issues in exploratory data mining in the behavioral sciences*. London, England: Taylor & Francis.

Marcoulides, K. M., & Yuan, K.-H. (2017). New ways to evaluate goodness of fit: A note on using equivalence testing to assess structural equation models. *Structural Equation Modeling*, *24*, 148–153. doi:10.1080/10705511.2016.1225260

McArdle, J. J., & Ritschard, G. (2014). *Contemporary issues in exploratory data mining in the behavioral sciences*. London, England: Taylor & Francis.

Muthén, L. K., & Muthén, B. O. (2014). *Mplus user's guide*. Los Angeles, CA: Muthén & Muthén.

R Development Core Team. (2011). *R: A language and environment for statistical computing*. Vienna, Austria: R Foundation for Statistical Computing. ISBN 3-900051-07-0. Retrieved from http://www.R-project.org/

Rosseel, Y. (2012). Lavaan: An R package for structural equation modeling. *Journal of Statistical Software*, *48*, 1–36. doi:10.18637/jss.v048.i02

Scheines, R., Spirtes, P., Glymour, C., Meek, C., & Richardson, T. (1998). The tetrad project: Constraint based aids to causal model specification. *Multivariate Behavioral Research*, *33*, 65–117. doi:10.1207/s15327906mbr3301_3

Sörbom, D. (1989). Model modification. *Psychometrika*, *54*, 371–384.

Whittaker, T. A., & Marcoulides, G. A. (2007, April). *Model specification searches using cross-validation indices*. Paper presented at the Annual Meeting of the American Educational Research Association, Chicago, IL.

Yuan, K.-H., Chan, W., Marcoulides, G. A., & Bentler, P. M. (2016). Assessing structural equation models by equivalence testing with adjusted fit indexes. *Structural Equation Modeling*, *23*, 319–330. doi:10.1080/10705511.2015.1065414