Model specification search using genetic algorithm for factor analysis model

因子分析モデルにおける 遺伝的アルゴリズムを用いたモデル探索

Hiroto Murohashi

室橋 弘人

Contents

Chapter 1

The necessity of model specification search in structural equation modeling

1.1 What is structural equation modeling? Origin and features

When several objects are measured in multiple perspectives, multivariate data is obtained. Some methods to draw information from this type of data are generally known as multivariate analyses. Currently, collecting multivariate data on certain phenomena and analyzing this data using multivariate analyses constitute virtually indispensable processes to examine the nature of these phenomena and their interrelationships from a scientific and objective perspective.

1.1.1 The superiority of SEM to other multivariate analysis methods

Multivariate analysis includes a wide variety of methods. However, almost all of these methods treat the information involved in multivariate data by adopting a mathematical approach and performing some arithmetical operations to represent this information in a form that is easier to interpret. Multivariate analysis is therefore a tool that converts multivariate data based on certain mathematical models and accentuates some information that is difficult to obtain using the original data, i.e., in the form of multivariate data.

The purpose of multivariate analyses can be broadly divided into two categories. One is to describe the characteristics of phenomena in an abridged form, and the other is to describe the association among these phenomena. Factor analysis and principal component analysis are the representative methods of the former category; these methods compile the similar properties of phenomena and facilitate the summarizing of objects. Analysis of variance (ANOVA) and regression analysis are typical examples of the latter category; these methods examine the correlations among various phenomena and attempt to discover some causal relationships or laws that affect objects. Structural equation modeling (SEM)—the main topic of this study—is also one method of multivariate analysis. However, it is difficult to classify SEM into one of the two categories described above because it is essentially a flexible method and is able to accomplish the purposes served by both of the categories.

In multivariate analyses, information is extracted from multivariate data based on certain mathematical models. Each method of analysis employs a different mathematical model, but all methods involve the risk of excluding some information that cannot be captured by the model employed. For example, the classical ANOVA assumes that the influences from one object to another are linear. This assumption arises from the mathematical model employed by ANOVA. ANOVA is therefore unable to draw nonlinear correlations among objects even if multivariate data does contain such information. It is important to be aware of which mathematical model is assumed and what its limitation are when performing a multivariate analysis.

However, the mathematical model employed by SEM is considerably more flexible than the ones employed by other multivariate analyses. Thus, we can attempt to extract many types of information according to the purpose of our analysis by using only one SEM method. This is the great advantage of SEM and one of the reasons why it is known as "the second generation of multivariate analysis" (Fornell, 1982a, 1982b). Alternatively, SEM can be treated as a larger framework of analyses involving some of the traditional multivariate analyses that employ a limiting mathematical framework. Therefore, SEM does not necessarily imply a single method. Instead, it is appropriate to regard SEM as a collection of some related multivariate analysis methods (Kline, 2005). There are some terms that represent the same concept as SEM, such as "covariance structure analysis", "causal modeling", or "analysis of covariance structure". This diversity itself reflects the multifaceted nature of SEM.

1.1.2 The development history of SEM

The major cause of diversity in SEM is related to its development process. In fact, SEM was proposed as a framework to combine two different methods of multivariate analysis. Therefore, SEM has inevitably been characterized by a variety of aspects since its naissance. Factor analysis and path analysis form the two origins of SEM.

Factor analysis

Factor analysis is a multivariate analysis method proposed by Spearman (1904a). Spearman, an English psychologist, contributed considerably to psychometrics through his study on the two-factor intelligence theory, which assumes that two inner components concern the performance of people's intellectual activity. One component is general intelligence, which commonly affects all intellectual activity. The other component is a specific factor, which affects only some of the intellectual activity. If we accept this theory, we should give greater importance to general intelligence than to the specific factor in order to achieve a highly accurate estimation of people's performance of intellectual activity. This is because general intelligence is related to a greater number of intellectual activities than the specific factor. In view of this, Spearman attempted to create a method to measure one's general intelligence.

General intelligence is an abstract quality that people possess; this makes it impossible to measure it directly in defined units such as physical quantities. Spearman therefore considered estimating the underlying general intelligence through several measurable variables that concern intellectual activity. For example, he used test scores as data. Since solving a test is a type of intellectual activity, it can be inferred that general intelligence commonly affects all the test results. Then, we can determine general intelligence by examining the correlations among multiple test scores. However, test scores are not identical to general intelligence itself because they are also influenced by specific factors. Moreover, measurement errors are inevitably mixed. Consequently, the correlations among test scores fall to a lower value than those among different general intelligence levels, which we wish to measure. This phenomenon is currently known as attenuation. In the process of developing his own study aiming to explore a method of coping with attenuation (Spearman, 1904b), Spearman developed the method of factor analysis.

The factor analysis model assumes that every observed variable is affected by one common factor that commonly influences every observed variable and by a unique factor that influences only that variable. This assumption represents the mechanism of attenuation whereby each observed variable is influenced by the common and unique factors in a variety of densities, thereby disturbing the correlations among these variables. The purpose of factor analysis is to estimate the actual process behind attenuation and to extract a pure common factor. The idea behind separating the common and unique factors is identical to the theoretical framework of the two-factor intelligence theory. It is therefore obvious that Spearman developed factor analysis in order to validate his own theory.

However, because Spearman established a method to quantify unobservable concepts, the application of factor analysis extended not only to research on intelligence but also to the social sciences as a whole, inclusive of psychology. In the social sciences, it is common to deal with latent concepts that lack in physical substance. However, there are virtually no methods that can objectively capture such concepts. At this juncture, factor analysis indicated the means to materialize latent concepts by revealing the causes of the correlations among observed variables as a common factor. Spearman applied factor analysis to the observed variables that are related to intellectual ability and derived the latent concept of general intelligence, which underlies his observed variables. However, the same procedure can be applied to concepts other than intelligence. This implies that Spearman established a general mathematical method to measure latent concepts. Thus, factor analysis came to be widely applied in various research areas.

Spearman's two-factor intelligence theory was subsequently contradicted by Thurstone (1935), and the multiple-factor theory, which assumes that intelligence contains a variety of components and that general intelligence is nonexistent, became mainstream. However, factor analysis itself continued to be applied. At present, it is typical to assume a number of common factors as in Thurstone (1947) rather than a single common factor as in Spearman (1904a); however, the basic idea underlying this concept is still credited to Spearman. SEM is also based on this concept. In SEM, we use two types of variables: observed variables and latent variables. The former is a directly measured value, and the latter is a value that is not measured directly. It is common to assume that latent variables are estimated from observed variables by the model based on Spearman's factor analysis. Moreover, we refer to the equations representing observed variables—the sum of the common factors and the unique factor—as the measurement equations in the LISREL model, which will be described later. This obviously reflects the fact that latent variables are assumed to be measured by the model based on the factor analysis in SEM.

Path analysis

The other origin of SEM is path analysis, which was proposed by Wright (1918). Wright—an authority on population genetics—advocated the shifting balance theory and is famous his arguments with Fisher on the evolutionary mechanism. However, like Fisher, Wright is also famous for his contribution to statistical methods. Path analysis is one of his greatest achievements in statistics.

Path analysis aims to quantitatively identify the size of the effects from one variable to another. This may imply that there is no difference between path analysis and multiple regression analysis. However, path analysis has a definite advantage over regression analysis in that it has flexibility and a variety of configuring correlations among variables. On the other hand, multiple regression analysis can only examine the influence of one group of independent variables on a single dependent variable. In contrast, we can freely assume the relationship among variables in path analysis. We can therefore build a sufficiently complicated model to express our hypothesis; in the model, multiple variables are influenced by multiple variables and one variable is affected by some variables and also influences other variables.

However, the flexibility of path analysis leads to the diversity in the form of equations assumed among the variables. Traditional multivariate methods such as regression analysis directly solve simultaneous equations that represent the correlations among variables. Then, if the assumed correlation changes and the equations changes, the process of solving the simultaneous equations should also change. This sometimes leads to the impossibility of solving the simultaneous equations. Regression analysis deals with this problem by limiting the form of equations, which also causes the model expression to have a low power. In contrast, path analysis can solve a great variety of equations by employing only one method. This is because path analysis employs the idea of dealing with the entire covariance structure among variables and not just the simultaneous equations. The covariance structure is derived from the simultaneous equations among variables and aggregates the cross relationships. If the assumed correlations among variables change, some values of the covariance structure also change. However, the framework to solve the covariance structure continues to function. This is one of the remarkable features of path analysis.

The other feature of path analysis is the use of a path diagram to describe the covariance structure. Path diagrams provide a graphic representation of all the assumed correlations among variables. For example, figure 1.1 shows an example of the path diagram that represents the model where variable A affects variable B and variable B affects variable C.

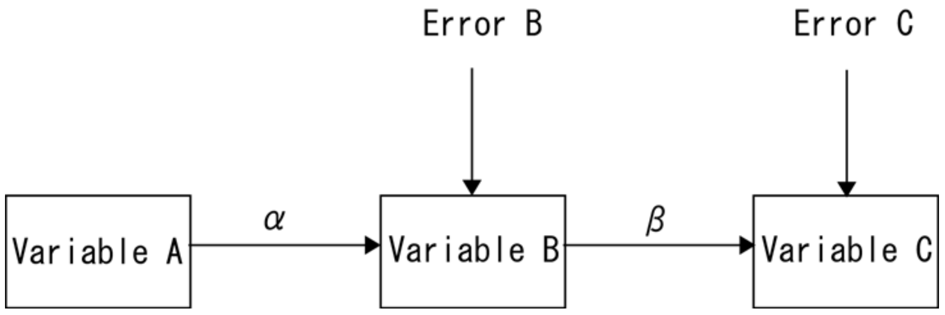


Figure 1.1: An example of a path diagram

Similar to the above figure, the influences from one variable to another are depicted by arrows in the path diagram. Figure 1.1 collectively represents the following two equations among the variables:

$$\begin{aligned}\text{variable } B &= \text{coefficient } \alpha \times \text{variable } A + \text{error of } B \\ \text{variable } C &= \text{coefficient } \beta \times \text{variable } B + \text{error of } C.\end{aligned}$$

Path diagrams provide a clear understanding of the complicated covariance structure.

The two abovementioned features of path analysis also apply to SEM. Although the notation is extended, the path diagram is still used in SEM to represent models. The approach to treat the entire covariance structure is also used in SEM and is the cornerstone of parameter estimation. Therefore, it would be accurate to state that SEM is directly derived from path analysis.

Establishment of SEM

The development of SEM from the proposal of path analysis was a lengthy process. This is because path analysis has seldom been focused on for a few decades despite its originality and availability. It was after Duncan's (1966) rediscovery of path analysis in the field of sociology that path analysis became popular. In sociology, great importance is accorded to the study devoted to developing a method to measure correlations among phenomena or concepts that are unable to be measured directly because it is imperative to examine such relationships in this field. For example, Blalock (1963) proposed the method intended to discover a causal association among latent concepts by analyzing observable variables that are considered to be concerned with the target concepts. In response to this necessity, Duncan rediscovered path analysis and achieved some significant results by applying this method to social research data; for instance, in Blau & Duncan (1967), he revealed the fact that the hierarchy of American society is passed on from parents to children mainly through education. Due to his achievements, the use of path analysis became widespread in the field of sociology in the 1960s.

However, Duncan's accomplishments are not limited to the rediscovery of path analysis. A more important accomplishment is the fact that he revealed the close relations between path analysis and other methods of analysis, including the simultaneous equations model used in econometrics and factor analysis used in psychology, and indicated the possibility of a united framework of multivariate analyses. The simultaneous equations model was used to express a theoretical economic model by representing economic phenomena in some regression equations and solving them at the same time. While this method can examine complicated causal relationships, the purpose of the analysis is limited to observable variables such price or expenditure; thus, we cannot consider latent concepts such as economic climate. In contrast, as previously mentioned, factor analysis is a method of analysis for extracting latent concepts. However, factor analysis can only describe the existence of factors; thus, we are unable to delve further into causality among factors.

Duncan and his co-worker Goldberg, an econometrician, suggested the possibility of linking the simultaneous equations model and factor analysis by using the framework of path analysis because the concept behind using the simultaneous equations model to treat certain regression equations at the same time is identical to the concept of path analysis; the factor analysis model can also be described as being the union of some simultaneous equations. If we can combine the accurate measurement of latent concepts enabled by factor analysis and the high flexibility of causal relationship expression enabled by the simultaneous equations model under the framework of path analysis, then the consequent method of analysis is expected to be very useful in all the fields of the social sciences, which mainly deal with latent concepts. Some studies were published after this suggestion was made by Duncan and Goldberg, but they only reported one example of analysis and were insufficient to be able to establish a new framework of analysis. It was in the 1970s that a breakthrough was created by a series of studies: Jöreskog (1973), Keesling (1972), and Wiley (1973). These studies extended the framework of path analysis and organized the theory of a more general method of analysis including the simultaneous equations model and factor analysis. This method—then known as the JKW model from the initials of the authors—marked the origins of SEM. Hence, SEM was originally proposed to unite some methods of analysis; this suggests that SEM includes more diverse elements than other multivariate analyses.

Presently, the JKW model is frequently known as the LISREL model after the software that implements the method proposed by Jöreskog, Keesling, and Wiley. LISREL is the oldest SEM model; it illustrates SEM's essential feature well. By summarizing the LISREL model, the major characteristics of SEM will be described in the following.

SEM is an expansion of path analysis and the simultaneous equations model; therefore, users must represent the assumed relationship among variables in the form of some equations and solve them simultaneously to arrive at a conclusion. One of the main features of the LISREL model is to classify these equations broadly into two categories, the measurement and structural equations. The measurement equation is the part of the model representing that the observed variable is measured as the sum of the common factors and the specific factor, based on Spearman's factor analysis model. We can also view this equation as the separation of the latent variables from the observed variable. By using the measurement equation, it becomes possible to incorporate unmeasurable variables into the analysis in SEM. In contrast, the structural equation is the part representing the relationships between the variables through regression equations. The structural equation can express not only the relationship between observed variables but also that between latent variables and between the observed and latent variables. This flexibility of SEM enables its users to constitute a greater variety of models than regression analysis or the simultaneous equations model, which can only express relationships among the observed variables. This is a major advantage of SEM as compared to the existing methods of multivariate analysis.

Another important feature of SEM is its ability to solve simultaneous equations collectively in the form of the covariance structure. This idea is derived from path analysis and makes it possible to solve multiple equations using a unified methodology. However, SEM is not necessarily the same as path analysis. Theoretical progress specific to SEM rather than path analysis uses linear algebra to derive the covariance structure.

The two abovementioned characteristics—the high degree of freedom of modeling and the use of the covariance structure—are not only specific to the LISREL but also commonly applicable to the general SEM. Therefore, the approach that is adopted in these methodologies in order to represent and examine a hypothesis is itself considered to be a valid definition of SEM. Each model assumed in the individual analyses is just a representation of SEM and is not equivalent to SEM as a whole. Rather, the essence of SEM is composed of the theoretical framework underlying every analysis.

1.2 Outline of the major research areas relating to SEM

The process underlying the origin of SEM has been described in the previous section. However, SEM in its current form is quite different from the original. This is because till date, it has been subject to several modifications. The establishment of SEM in the early 1970s provided a new and unified perspective for some methods of analysis that were previously regarded as distinct methods, and thereby enabled greater effectiveness and various improvements. This developmental history of SEM can be examined by roughly dividing it based on two aims. One is the objective to refine the original ingredients of SEM; and the other, to extend its framework in order to deal with new problems.

1.2.1 Basic and essential terms relating to SEM

The original ingredients of SEM are the research areas referred to in Jöreskog (1973), Keesling (1972), and Wiley (1973). In particular, the main themes are model notation, parameter estimation, model identification, and model fit.

(a). Model notation

Model notation pertains to research concerning how to aggregate individual equations among variables into a covariance structure. The JKW or LISREL model is the oldest method of model notation in SEM. As described earlier, the LISREL model distinguishes equations based on the type of variables that equation includes. However, later methods like the EQS model (Bentler & Weeks, 1980) and the RAM model (McArdle & McDonald, 1984) are focused on simplifying the method of notation. These are therefore more unified and clear as compared to the LISREL model. Another approach emphasizing the covariance structure more than each equation was also attempted, and the COSAN model (McDonald, 1985) was proposed as a result. Basically, these notation methods are different representations of the same covariance structure. However, the notation method affects the means to describe restrictions on the parameters included in the model and the means to determine the method of estimation. Each model notation represents a different preferred type of hypothesis.

(b). Parameter estimation

Parameter estimation refers to research concerning how to solve the equations and estimate the value of unknown parameters included in the model. The first method proposed was maximum likelihood estimation, which is commonly used in statistics. The maximum likelihood estimator is desirable particularly in the case of large samples, and therefore, this method is still used as the principal estimation method in SEM. Moreover, Arbuckle (1996) proposed the full information maximum likelihood estimation method in order to consider missing data.

Besides maximum likelihood estimation, methods based on least squares estimation are also used in SEM. There are different variations of least squares estimation, including unweighted, scale-free, generalized, and asymptotically distribution-free least squares. These estimation methods have different features, but their estimators are generally more robust than the maximum likelihood estimator. Moreover, it is indicated that these methods can be coherently understood from the viewpoint of weighted least squares (Browne, 1984). All the abovementioned estimation methods involve repetitions. In contrast, the two- and three-stage least squares methods do not involve any iteration in their estimation process. These methods are used for determining the initial values of iterative estimation methods (Fox, 1984).

Nowadays, there is a strong movement towards applying Bayesian statistics to parameter estimation in the field of statistics. This trend holds true in SEM, and many studies are attempting to apply the EM algorithm (Dempster & Rubin, 1977), or the Markov chain Monte Carlo method(Metropolis, Rosenbluth, Rosenbluth, Teller, & Teller, 1953; Hastings, 1970; Geman & Geman, 1984) for parameter estimation in SEM. Such methods are better than the existing ones in terms of solving complicated models. It is therefore probable that Bayesian estimation methods will become the principal estimation methods for SEM in the future.

(c). Model identification

Model identification concerns research that deals with assessing the theoretical capability of parameter estimation. Each model assumed in SEM is a set of simultaneous equations, and hence, there is a possibility that the model that the user wishes to estimate cannot be mathematically solved. In this case, it is important to derive the condition for determining when the model becomes unsolvable. However, a complete rule that can be applied universally is yet to be discovered. We are only aware of the necessary and sufficient conditions for certain limited situations; we do not know the necessary and sufficient conditions that can be applied universally in any situation.

The known necessary conditions for model identification are as follows: (a) the degree of freedom of the target model has a positive value; (b) the sample size is larger than the number of parameters to be estimated; and (c) all the scales of latent variables are defined (Toyoda, 1992; Kline, 2005). If a model does not satisfy one of these conditions, it is definitely nonidentifiable. Hence, the user should choose a model that satisfies at least the necessary conditions.

In contrast, if a model satisfies one of the sufficient conditions, it is certainly identifiable. However, there are still possibilities for different models to be identifiable. Therefore, sufficient conditions can be used as an indicator of model choice, but they do not cover all the possible models. Many sufficient conditions have been proposed thus far, including the ones that can be widely applied, such as the recursive rule and the ones that only determine an individual equation's identifiability, for example, rank condition and order condition Bollen:1989,Toyoda:1992.

Toyoda (1994) proposed one of the most widely available set of sufficient conditions. However, it is still not a complete one and requires further research.

(d). Model fit

Model fit refers to research regarding the verification of the analyzed model's validity based on data. Users can specify the model freely in SEM, and if the model is identifiable, they can arrive at some kind of solution. However, there is no assurance that the chosen model is appropriate enough to explain the structure of their data. To compensate for the inadequacy, the chi-square test is commonly used in multivariate analysis. However, this test is often unable to function normally in SEM due to certain reasons. Therefore, in SEM, it has become more common to use fit indexes instead of the chi-square test.

Many kinds of fit indexes are proposed depending on the perspective from which the goodness of model fit is defined. There exist some indexes that are commonly used in statistics such as the Akaike information criterion (AIC) (Akaike, 1974) as well as SEM-specific indexes such as the root mean square error of approximation (RMSEA) (Steiger & Lind, 1980; Browne & Cudeck, 1993). However, there is no decisive fit index, and hence, models that satisfy some fit indexes are considered to have a good fit.

A fit index expresses the validity of the entire model. By contrast, there are some methods to examine the validity of every path among variables. The representative techniques include the ones that apply the Wald test (Lee, 1985) and the Lagrange multiplier (LM) test to SEM (Lee & Bentler, 1980). These methods will be briefly described in the later sections of this article. Moreover, we can compare some models by using the method based on the likelihood ratio (LR) test (Satorra & Sarris, 1985).

1.2.2 Advanced terms relating to SEM

Another aspect of the development of SEM is extending its framework to accommodate new problems that were not treated in the earlier framework of SEM. This modification was possible because of the innate flexibility of SEM. As the applicability of SEM widened, it became more popular and frequently used in various areas. The major expansions in SEM are as follows.

(a). Moment structure

As described in the previous section, the original SEM was intended to analyze the covariance structure. This suggests that SEM only uses information on second-order moments. Sörbom (1982) made improvements to this aspect and proposed a method that also uses first-order moments. This method of analysis is sometimes known as the mean and covariance structure analysis in order to distinguish it from the normal SEM. Under the traditional SEM, which uses only the covariance structure, an implicit restriction is imposed that every latent variable included in the model has the same expected value. However, this restriction becomes unnecessary if we use the mean and covariance structure method of analysis. Moreover, this method enables us to estimate the mean value of the observed and latent variables. We can therefore introduce a more detailed consideration of the data. The mean and covariance structure analysis becomes particularly effective when we want to examine the longitudinal change in latent variables.

Moreover, there have recently been some studies that attempted to use higher-order moments (Mooijaart, 1985; Shimizu & Kano, In press). If we consider higher-order moments in the analysis, we can extract additional information from the data as compared to the case when using the normal SEM and the mean and covariance structure analysis model. In this case, some models that are unidentifiable or just identifiable under the existing methods become identifiable. This appears rather advantageous; however, nearly all the theoretical studies on SEM have assumed that data is normally distributed, and the existence of higher-order moments like skewness and kurtosis has been ignored. Hence, this kind of approach that actively considers the non-normality of data is still rather primitive, even though it is indicated as having been effective in Bentler (1983).

(b). Treatment of sample

In the original SEM, all samples are assumed to be obtained from one common parent population. If this assumption does not hold, the estimation result from the normal SEM can be inaccurate. To deal with this problem, some models for non-heterogeneous samples are proposed and employed. If we know the population from which the samples are derived, we can use the multiple group analysis model (Kano, 2001).

This model is based on the concept of Guttman (1952) and is effective in examining the difference between populations. Moreover, using this model in combination with the mean and covariance structure analysis enables us to compare the mean values of latent factors and provides us with a very rich consideration of the data (Mulaik, 1986; Kano & Miura, 2002). In contrast, if we do not know the population from which the samples are obtained, we can estimate this affiliation itself by using mixture modeling with latent variables (Muthén & Shedden, 1999; Muthén, Shedden, & Spisic, 1999). This type of mixture modeling can also help us analyze the correlations between the probability of affiliation to each population of every sample and the other variables used in the model. This makes it possible to examine the characteristic of the populations.

Moreover, if we can assume the homogeneity of the population and if the sampling method is not random sampling but stratified sampling, it is recommended that the mechanism of data sampling itself should be expressed in the model being analyzed. There are two approaches to deal with this problem: one is to improve and extend the SEM framework and to construct a new theory of analysis for multilevel sampling data (McDonald & Goldstein, 1989; Lee, 1990; Goldstein, 1995), and the other is to devise a method to express the process of multilevel sampling within the existing framework of the SEM (Muthén, 1994; Muthén & Satorra, 1995; Duncan et al., 1997). Because of its applicability, the latter approach is primarily used at present. By using these methods for multilevel data, we can distinguish the between- and within-class variations and obtain more accurate knowledge about the data.

(c). Type of variable

The principal estimation method employed in SEM is maximum likelihood estimation, which assumes that all variables are normally distributed. Therefore, typically, the type of variable used in SEM is implicitly limited to continuous variables. However, since ordinal or dichotomous variables are often used in the social sciences, the method to be used for this type of variable is needs to be determined. Ordinal variables appear to be equivalent to continuous variables, and we are even able to treat them as continuous variables in parameter estimation. However, the intervals between the scales are not consistent in the case of ordinal variables. This has adverse effects on the estimation results (Bollen, 1989). When we want to use an ordinal variable as the independent variable in the model, the widely used approach is to assume that the continuous variable underlies the ordinal variable and that continuous variable is divided at certain points and is thereby converted into an ordinal variable. Then, the underlying continuous variable is used in the estimation process instead of the ordinal variable. This idea was originally investigated in the area of factor analysis (Muthén, 1984; Bollen, 1989). In contrast, when we want to use an ordinal categorical variable as the dependent variable, the ideas of the logistic regression and probit analyses are incorporated (Muthén, 2004). In this case, it is equivalent to assume the existence of a continuous variable underlying the categorical variable.

(d). Nonlinearity

Linear equations are used to describe the correlations between the variables in SEM. We can therefore only extract a linear causal relationship by using the normal SEM. To overcome this problem, some methods have been proposed (Ping, 1996; Bollen & Paxton, 1998; Schumacker & Marcoulides, 1998). However, all these methods do not use nonlinear equations but incorporate some nonlinear terms into the linear equations. SEM is essentially a linear statistical model, and it is only possible for us to introduce nonlinear effects as a part of the model.

(e). Devising models

Some researches that attempted to extend the theoretical framework of SEM were presented. Other than those mentioned above, there are many other studies that focused on devising certain specialized models within the existing framework of the SEM. SEM involves numerous multivariate analysis methods as its subordinate models. Hence, we can execute such analyses by using SEM. Transplanting the traditional analysis methods into SEM enables us to relax some restrictions imposed in the theoretical framework of the old methods. For this purpose, many multivariate analyses have been re-represented in terms of the SEM model.

Moreover, many models aiming to represent the mode of data have also been proposed. Most of the multivariate data have two modes, sample and variable. However, some data have three or more modes.

To analyze this type of data, it is recommended that a special model that expresses its mode structure be employed. For example, semantic differential (SD) data has three modes: sample, variable, and pair of adjectives. We can use models such as the multimode model (Bentler, Poon, & Lee, 1988), direct product model (Bagozzi & Yi, 1992), and multimode direct product models (Verhees & Wansbeek, 1990) for SD data. Longitudinal data is another typical multimode data that has three modes: sample, variable, and time point. An old and relatively simple model for longitudinal data is the autoregressive model, which represents the framework of Guttman (1954) and examines the simplex structure of the ordered tests in SEM. This model is also known as the Markov simplex model. There are other models including the differential model, which expresses the change in the difference (Arminger, 1986), and the latent growth curve model, which assumes the existence of a certain function behind longitudinal data and estimates its parameters (Meredith & Tisak, 1990). In addition, some models are improved to analyze latent variables instead of observed variables (Duncan & Duncan, 1996; McArdle & Hamagami, 1996).

1.2.3 Computer software programs for SEM

The present popularity of SEM cannot be separated from the development of its computer software programs. One of the features of SEM is that researchers supplied software programs that implemented their proposed methods or models and improved them based on user feedback. Thus, we conclude this section by examining SEM software programs.

The first software program for SEM was LISREL, which implemented the JKW model and is regarded as being synonymous with this model notation. At present, LISREL (Jöreskog & Sörbom, 2006) continues to be one of the representative computer software programs for SEM. Apart from LISREL, there are some software programs that were developed to achieve certain model notations. EQS (Bentler, 2003) employs the EQS model and Mx (Neale, Boker, Xie, & Maes, 2004) focuses on the matrix notation based on the RAM model. Moreover, at present, most of the comprehensive statistical software packages involve SEM functionality. SAS/STAT (SAS Institute Inc., 2004) implements the CALIS procedure; SYSTAT (Systat Software Inc., 2004), the RAMONA module; and STATISTICA (StatSoft Inc., 2004), the SEPATH module.

A current trend in SEM software programs is employing GUI; this enables users to specify their model by drawing a path diagram on the screen. This type of interface is very easy to use and has contributed significantly to the diffusion of SEM. AMOS (Arbuckle, 2006) is particularly famous for its user-friendly GUI. In contrast, another trend is the enhancement of functionality. SEM is still developing as a method, and new research continues to be published. Therefore, implementing these achievements is also important. Mplus (Muthén & Muthén, 2006) does not employ an intuitive GUI, but it can analyze a wide variety of models including the features unique to Mplus, such as finite mixture modeling and latent categorical variables. Hence, it serves a certain purpose in SEM software programs.

1.3 How to define and find a "good" model in SEM?

When we use SEM, we have to determine the entire specification of a path diagram, including how many latent variables are involved and how all the variables are interconnected, prior to executing the analysis. SEM is therefore sometimes referred to as a confirmatory method of analysis (Kline, 2005). However, researchers do not always have an explicit hypothesis and are eager to use their data in order to find an appropriate model in application studies Sörborm (1989). Additionally, even if researchers hold certain hypotheses, the models generated via researchers' hypotheses rarely fit the collected data perfectly (Henderson & Denison, 1989). In light of this, is it impossible to use SEM in these cases? The answer is "No". There are some researches dealing with how to find an appropriate model in SEM. These are generally known as model specification searches.

Jöreskog (1993) classified the purposes of SEM into the following three categories: (a) strictly confirmatory purpose where the researcher wishes to verify that his hypothesized model is appropriate, (b) a slightly confirmatory purpose where the researcher wants to compare some alternative models and find the best one among them, and (c) an exploratory purpose where the researcher wants to generate a good-fitting model based on the data. Model specification search deals with the third purpose of this classification. However, for model specification search, we need to first define the characteristics of the "good" model. How we can assess the goodness of models in SEM?

1.3.1 Evaluation of the model's goodness-of-fit to the data

(a). Using a residual matrix

Every model assumed in SEM can be represented in a covariance matrix form derived from simultaneous equations corresponding to the causal relationships among the variables. Meanwhile, the collected data has its own covariance structure; the former covariance form is a theoretically hypothesized correlation, and the latter is an actually existent one. If both of these are alike, the assumed causal relation is regarded as being probable in that it is supported by facts. This similarity forms the basic concept of the goodness of the model in SEM.

In mathematical terms, the covariance structure of data is expressed as Σ and the covariance structure of the model is expressed as $\Sigma(\boldsymbol{\theta})$, which means that the covariance structure Σ is parameterized by some parameters $\boldsymbol{\theta}$. Therefore, if the difference between Σ and $\Sigma(\boldsymbol{\theta})$ is small, this model can be considered to be a good one. However, complete data on a certain population is rarely available and we cannot know the true value of the parameters $\boldsymbol{\theta}$. Then, the only information we can obtain is the sampled data $\boldsymbol{S}$ and the estimates of the parameters $\hat{\boldsymbol{\theta}}$. The difference between these, given by $\boldsymbol{S} - \Sigma(\hat{\boldsymbol{\theta}})$, is termed the residual or the residual matrix and is used as an index to represent the analyzed model's fitness of the data (Bollen, 1989). However, the magnitude of the elements in a residual matrix is known to be greatly affected by the scale of the observed variables and the sample size. In view of this, Jöreskog & Sörborm (1986) proposed a standardized residual that eliminates these influences; at present, it is common to use the standardized residual instead of the normal one.

The smaller the magnitude of the standardized residual matrix, the better is the assumed model. However, even if $\Sigma(\hat{\boldsymbol{\theta}})$ represents the definitely true structure $\Sigma(\boldsymbol{\theta})$, the residual almost never becomes zero because the sampled data $\boldsymbol{S}$ includes the measurement error or the fluctuation to some extent. Consequently, the problem concerns how small the magnitude of the residual matrix should be in order to indicate that the analyzed model fits the data. In SEM, the chi-square test is used as a guide to determine this validity. Under some popular estimation methods such as the maximum likelihood estimation, it is indicated that the statistic based on the value of the target function used in the estimation process, which is known as the discrepancy and is essentially the same as the residual, asymptotically follows a chi-square distribution (Browne, 1974, 1982, 1984; Bollen, 1989). By using abovementioned feature, we can test the smallness of the residual.

This chi-square test is however deemed as an unreliable indicator from various perspectives (Murohashi, 2003). The statistic based on discrepancy is not always chi-square distributed but is asymptotically chi-square distributed under some conditions. Therefore, if one or more of these conditions are not met, the result of the chi-square test will be biased. The first condition is that the data lacks kurtosis. A finite sample from a normally distributed population could have kurtosis to some degree, and the observed variables from the nonnormal population certainly has kurtosis (Browne, 1974, 1982). The chi-square test is not effective for these types of data. The second condition pertains to whether the covariance matrix or correlation matrix is analyzed. If the correlation matrix is analyzed, larger correlations generally lead to a higher chi-square value (Jöreskog & Sörborm, 1986). The third condition concerns the sample size. An asymptotic chi-square value is very sensitive for sample size; samples that are too small as well as and too large distort its estimates (Boomsma, 1983; Kano & Miura, 2002). For the first problem, it is effective to use estimation methods that are not affected by data distribution (Browne, 1984), and for the third problem, adding some modification to the chi-square value has been proposed (Bollen, 1989). However, no methods that can deal with all these problems have yet been discovered.

(b). Using fit index

Because of the reasons described above, the fit index indicator is presently used more often than the chi-square test to evaluate the goodness-of-fit of a model in SEM. "Fit indexes" is a generic term for numerical values computed by performing some corrections to certain statistics that represent the model's goodness-of-fit based on the elements that affect the statistics. Many fit indexes are proposed from various perspectives, but they are commonly more sophisticated and robust than the chi-square test described above (Kline, 2005).

There are two major types of fit indexes: one is based on the residual matrix, and the other on the discrepancy. The representative examples of the former type are AGFI and RMR (Jöreskog & Sörborm, 1986), and those of the latter type are CFI (Bentler, 1990) and NFI (Bentler & Bonnet, 1980). In addition, there is another type of fit indexes that employs the concept of information criteria, which assess the model fit in a hypothetical replication of data sampling based on one set of sampled data. These are sometimes known as predictive fit indexes; examples of this type of index are the AIC (Akaike, 1974) and the Bayesian information criterion (BIC) (Raftery, 1993).

Moreover, there is another important viewpoint in assessing the model fit: parsimony. If the degree of freedom of the model becomes small, these statistics become smaller unconditionally. Further, when the degree of freedom of the analyzed model is zero, the simultaneous equations of the model are just identified and the residual and the discrepancy always become zero (Toyoda, 1998). Therefore, a low value of the residual and discrepancy do not necessarily imply the high adequacy of the model. Instead, we must treat a model with a low degree of freedom as a potentially poor-fitting one because of overfitting (Raykov & marcoulides, 2000). The concept of parsimony is effective in dealing with this problem. Parsimony rate is defined as the proportion of the degree of freedom of the analyzed model and the maximum number of estimatable parameters according to the data. Models that are more parsimonious and that consume fewer parameters can be regarded as relatively good models because they are simpler than complicated models, which employ many parameters and expect to be robust to the data. RMSEA (Steiger & Lind, 1980; Browne & Cudeck, 1993) is a representative index that applies parsimony.

(c). Using the likelihood ratio test

All fit indexes are defined to represent the goodness-of-fit of a model from certain different aspects. Therefore, the model that has good value in only one fit index is not treated as a good one, but that which has good values in some fit indexes is regarded as a good-fitting model (Maruyama, 1998). This methodology has become popular in SEM since it was proposed by Akaike (1987).

However, most of the statistical properties of fit indexes have not been revealed, and there exists virtually no precise criterion to determine what value of a particular index is sufficiently appropriate (Thomarken & Waller, 2003). There are some rule-of-thumb standards, but they often depend on the research area (McDonald & Ho, 2002). Therefore, if one wants to show that there is an exact statistically significant difference between the goodness-of-fit of two models, it is recommended that the difference between the chi-square values of the target models be tested according to the chi-square distribution.

This test employs the feature that the difference between chi-square random variables is also a chi-square random variable; the test is sometimes known as the LR test (Bollen, 1989).

1.3.2 Automated model specification search

By using the chi-square test of discrepancy, fit indexes, and the LR test, we can achieve the first and second purposes of SEM as categorized by Jöreskog (1993). However, a comparison of two models is insufficient to meet the third purpose. Some methods indicate the modification alternative are required. To deal with this problem, we can typically use the following two methods.

(a). Lagrange multiplier test

Specifying the model to be analyzed in SEM implies not only determining which parameters are to be estimated but also which are not to be estimated; this is because “unestimated” parameters are constrained to be zero as per the estimation process. Hence, specifying a model is equivalent to imposing restrictions on the entire set of simultaneous equations representing the relationship among variables. Then, there exists a different value of the LM corresponding to differently restricted equations, and the difference in the value of the LM represents the difference in the model.

The LM test (Rao, 1948; Aitchison & Silvey, 1958) is a statistical test that examines the difference between Lagrange multipliers; Lee & Bentler (1980) applied this idea in the comparison of two models in the SEM framework. The LM test in SEM is used to examine the significance of adding some freely estimated parameters, which is equivalent to relaxing the restriction that the relevant parameters are estimated to be zero.

(b). Wald test

The Wald test (Wald, 1943) is a multivariate generalization of the square of the normal z-test and is proposed as a statistical test to assess the hypothesis that constrains the estimated parameters to be unestimated. This test is used to examine the significance of removing some freely estimated parameters in SEM (Lee, 1985; Bentler & Dijkstra, 1985). Therefore, the LM test is also used to examine hypotheses that are completely opposite.

The LM and Wald tests have been distinctly described above. However, Bentler & Dijkstra (1985) indicated that both of them are essentially equivalent because they can be regarded as conducting LR test by examining the difference between the chi-square value of two different models. Despite their homogeneity, the LM and Wald tests are still known by different names in SEM. This is because imposing and relaxing a restriction have opposite implications; it is therefore convenient to distinguish the two tests by using different names.

If the starting model is determined, we can achieve the third purpose of Jöreskog (1993) by using the two abovementioned tests to improve the model fit. This approach is proposed by Long (1983) as an idea toward the realization of model specification search and has been implemented in a particular software program upon his suggestion (Bentler, 1986). However, despite its availability, model specification searches are far from widespread nowadays. Moreover, in Bentler (1995), it is cautioned that care must be taken in model specification search when using the LM and Wald tests and that its results should not be unconditionally considered as the final conclusion.

This is because based on a large-scale simulation study, MacCallum (1986) confirmed that if the LM and Wald tests are used in model specification search, the true model is seldom arrived at. MacCallum (1986) cited five reasons for this inaccuracy: (a) The estimators of the parameters are correlated with each other, and hence, the order of adding or removing the restriction affects the search result. (b) One trial of the model specification search involves many repetitions of statistical tests; hence, the cumulative error rate cannot be ignored. (c) The LM and Wald tests find the modification alternative that is expected to improve the chi-square value of the model to a great extent; however, it does not directly imply improvement in the interpretation of the model. (d) The LM and Wald tests assume that certain statistics are asymptotically chi-square distributed, but this assumption holds true only when the estimated model is equal to the true model. Therefore, the result of these tests will be biased under a misspecified model, where model specification search is mainly performed. (e) If the starting model differs greatly from the true model, it is highly possible that the LM and Wald tests will not be able to find the true model. For example, they cannot determine how to modify parameters in terms of adding or removing latent variables.

With regard to the third problem, the method that can estimate not only how the model fit will be improved but also how the estimates of the parameters will change by adding or removing a certain path is proposed in the case of the LM test (Sarris, Satorra, & Sörbom, 1987). This statistic is called the expected parameter change and represents the prospective size of the change in parameter estimates. Sarris et al. (1987) indicated that preferentially modifying the parameters that have high expected change estimates leads to greater improvement in the model fit. However, with regard to the other problems, an effective solution has not yet been discovered; therefore, model specification search has thus far been regarded as unreliable method. However, model specification search is crucial per se and plays an important role particularly in application studies. Therefore, determining more accurate methods for model specification search is one of the critical research issues in SEM.

Chapter 2

Model specification search as a combinational optimization problem

2.1 Combinational optimization problem and its solution method

Another approach to model specification search that does not use the LM and Wald tests has been attempted. For example, Glymour, Scheines, Spirtes, & Kelly (1987) proposed an automated model specification search procedure named TETRAD. It is based on the theory of the covariance selection used in path analysis. It is more powerful than the method using the LM and Wald tests but is still far from being sufficiently precise.

Therefore, nowadays, a different approach has been focused on. As mentioned earlier, numerical values such as the chi-square value and fit indexes are used to evaluate the goodness-of-fit of models in SEM. Then, if a certain model is determined, we can obtain quantitative indexes based on this model. Consequently, we can regard model specification to be searched by using a function into which a certain model is substituted to derive its goodness in numerical form. From this viewpoint, the model specification search in SEM can be treated as a combinational optimization problem (Marcoulides & Drezner, 2001).

2.1.1 Basics of the combinational optimization problem

The optimization problem refers to the task of optimizing an objective function under certain constraint conditions. The condition can usually be represented as a set of feasible solutions, and the purpose of solving the optimization problem is to find the solution from the set of feasible solutions that minimizes the objective function. This solution is called the optimal solution. In addition, if the feasible solutions have a combinational structure, such an optimization problem is called a combinational optimization problem or a discrete optimization problem. A combinational structure implies that the set of feasible solutions can be expressed as a discrete set of a finite number of elements or a countable infinite number of elements. Therefore, the purpose of the combinational optimization problem is to find an appropriate combination or order of elements included in a set of feasible solutions (Lawler, 1976).

Assessing the algorithm

The use of an algorithm is one technique to solve a combinational optimization problem, and algorithm performance is mainly assessed by two different viewpoints—space complexity and time complexity. Space complexity is a concept that measures the memory size required by an algorithm. Time complexity is the amount of time consumed in solving a problem by using this algorithm. Both these concepts are not only specific to the algorithm but also dependent on the size of the target problem. The size of the problem is usually defined as the length of data required to represent the target problem. If the size of the problem is large, the memory and time complexities tend to increase as well. However, the size of the problem is not an essential element in determining the goodness of the algorithm because the algorithm concerns the structure of the problem; further, one algorithm can solve numerous problems having a variety of sizes but with the same structure. Therefore, when evaluating the efficacy of an algorithm, the space and time complexities are usually expressed as functions of the size of the problem (Sait & Youssef, 2000).

Of the two types of complexities, space complexity is mainly dependent on the size of the problem. Thus, it is not considered as much as time complexity when evaluating the goodness of algorithms. Moreover, the developments in computers have resulted in the increased availability of memory storage devices. Therefore, even if the size of the problem increases, the time complexity usually poses a more serious problem than the space complexity. Currently, time complexity is a major issue in the field of combinational optimization problems.

Types of problems

From the viewpoint of time complexity, the algorithms for combinational optimization problems are divided into two categories—tractable and intractable problems (Horowitz & Sahni, 1984). If an algorithm for a certain problem has a time complexity that can be represented as a polynomial function of the size of the problem, such a problem is known as a tractable or a polynomial problem. Polynomial problems are considered to be easily solvable since their time complexity is proportional to the size of the problem. In contrast, if a problem only has algorithms in which the time complexities are expressed as functions of an order higher than that of a polynomial function, such as exponential or logarithmic functions, the problem is categorized as an intractable problem. The time complexity of intractable problems increases at a faster rate according to the size of the problem; hence, such algorithms are regarded as impractical methods (Foulds, 1984).

Many types of problems fall under the category of intractable problems. Some of them have time complexities expressed in the form of a logarithmic function; however, these do not require much calculation time, provided the size of the problem is small. These types of problems are known as pseudo-polynomial problems and can be solved in most cases. However, there exist far more intractable problems that are very difficult to solve. One important category of intractable problems is a class of nondeterministic polynomial problems, which are usually known as class NP problems. The distinctive feature of class NP problems is that the algorithm for the problem can determine whether or not a certain solution is feasible in polynomial order time; however, the algorithm cannot perform this determination for all the possible solutions in polynomial order time. This type of problems can be solved in polynomial order time by assuming the existence of a computer that can execute nondeterministic computation—an ability to simultaneously evaluate multiple branches in the algorithm—because a nondeterministic computer can enumerate all the possible solutions in polynomial order time. The problems that are strictly involved in class NP are known as NP-hard or NP-complete problems (Garey & Johnson, 1979). In contrast, problems having polynomial order time algorithms are regarded as class P problems, named after the initial letter of the word "polynomial".

In fact, it has not been proved that class NP problems always do not have polynomial time algorithms and that they are definitely more difficult than class P problems. Moreover, it is extremely difficult to prove whether or not a certain problem is NP-hard. However, in reality, there are many computational optimization problems with complicated restrictions and undetermined polynomial order time algorithms. Such problems are basically regarded as NP-hard problems, and dealing with them has become a major issue in the field of computational optimization problems (Yanagiura & Ibaraki, 2001).

2.1.2 Algorithms for the combinational optimization problem

The most simple and certain algorithm to solve a combinational optimization problem enumerates all the possible combinations of parameters. This algorithm is known as the enumeration method. However, the time complexity of the enumeration method seldom becomes a polynomial function in the case of most combinational optimization problems. Therefore, the basic purpose of combinational optimization study is to find an algorithm that is more efficient than that obtained through the enumeration method. This type of algorithm is roughly divided into two categories (Horowitz & Sahni, 1984): an exact algorithm and an approximation algorithm.

(a). Exact algorithms

Exact algorithms can literally find the best solution of a problem exactly. The detailed descriptions of exact algorithms are as follows.

Branch and bound method

All the possible combinations of parameters can be described systematically by using a branching tree diagram where the value of the first parameter is fixed at a certain value at the first bifurcation point, the value of the second parameter is fixed at the second bifurcation point, and so on. The enumeration algorithm evaluates all the branches in the diagram, while the branch and bound method does not. In the branch and bound method, we determine whether all the solutions in lower parts from a certain branching point can involve the best solution and if they cannot, we do not evaluate such solutions individually. This operation is called a bounding operation. By using the bounding operation and eliminating some branches, we can save some time as compared to the case of the enumeration method.

Dynamic programming

Dynamic programming appears similar to the branch and bound method in the case of fixing some of the parameters and searching for the remaining ones. However, dynamic programming enumerates solutions in a more accomplished and efficient way than the branch and bound method. In dynamic programming, the solutions that have the same condition are merged and treated as a single solution. This concept is known as the principle of optimality, and it enables us to save more time than that in the case of the branch and bound method.

Linear programming

In linear programming, the target problem is transformed into the task of minimizing linear target functions under some linear constraints. This set of constraints is known to constitute a polyhedron, and each vertex of the polyhedron corresponds to the feasible solution. Some algorithms to solve linear programming are proposed, and it is confirmed that we can solve linear programming in polynomial order time even in the worst case. In most cases, the solution can be retrieved faster.

(b). Approximation algorithms

Among the three exact algorithms described above, linear programming is the most efficient. However, based on the definition of NP-hard problems, linear programming cannot be applied to obtain the solution of these problems; only the branch and bound method and dynamic programming can be used. However, it is known that the efficiency of these algorithms approaches that in the case of the enumeration method if we apply them to NP-hard problems (Sait & Youssef, 2000). Therefore, the exact methods are not sufficiently powerful to solve complicated combinational optimization problems. To deal with this problem, approximation algorithms have been investigated in the field of the combinational optimization problems.

A typical feature of approximation algorithms is that they are not always able to arrive at the globally optimal solution. Alternatively, they focus on searching for a sufficiently good solution in practical time. This type of algorithms is sometimes known as heuristics. Detailed descriptions of these algorithms are provided below.

Greedy method

In the combinational optimization problem, one solution usually comprises a certain number of elements. The greedy method does not address all these elements but deals with some elements at a time. For example, when minimizing a function, the same number of elements exists as the parameter of the function. The greedy method first considers some selected parameters and determines their values that will minimize the value of the target function. Once a subset of parameters is determined, some of the other parameters are considered. By repeating this process, the entire solution is generated. Thus, the greedy method is also known as a constructive method. Undoubtedly, this method may not necessarily be appropriate because some parameters are not considered. Therefore, the solution process of this method is sometimes considered as being based on the local contribution to the objective function.

Local search method

In the local search method, a concept known as neighborhood plays an important role. The neighborhood of one solution implies a set of some solutions that is similar to the original solution; further, the process of generating the neighborhood of a certain solution by introducing some transformation is called a neighborhood operation. When using the local search method, we have to first determine the initial solution. The neighborhood operation is then applied to the initial solution and the best-fitting solution included in the neighborhood is chosen as the destination. By repeating this procedure, we can improve the solution and find a better fitting one. This method is also known as the iterative improvement method, the hill climbing method, and the neighborhood search algorithm. How to define the neighborhood of a certain solution and how to choose the improved solution involved in the neighborhood will be important in the local search method.

Metaheuristics

Metaheuristics is an extension of the local search method. Basically, in the local search method, the best-fitting solution in the neighborhood is blindly chosen, and therefore, its result depends completely on the initial value. However, the search space of an NP-hard problem usually has a complicated shape, and several locally optimal solutions are possible. Therefore, we must choose the initial solution that results in a global optimal solution and not a local one. However, we are unable to ascertain whether or not the result of the local search method is a global optimal solution. Metaheuristics deals with this problem by using two completely different approaches. One is the diversification of the search and the other is the intensification of the search.

Diversification implies searching not merely the neighborhood of the base solution but a wider area. The diversification effect is usually attained by introducing some random process to the neighborhood operation. This effect disrupts the dependency of the result obtained by the local search method on the initial solution and enables a more robust search. However, the diversification effect basically serves to disturb the search process. Therefore, merely adding a diversification effect results in the inefficacy of convergence and creates the need for the intensification of the search. Intensification refers to the concentration of calculation effort according to the proximate optimality principle, which is an assumption that good solutions have similar structures. Based on this principle, metaheuristics focuses on searching not only the simple neighborhood but also a more promising set of candidate solutions. Combining these two effects, it is possible to obtain more accurate search results using metaheuristics rather than the local search method. However, metaheuristics typically consumes greater calculation time than the local search methods.

The method for solving the combinational optimization problem can be classified based on certain different viewpoints. One approach is using a deterministic or probabilistic algorithm. In deterministic algorithms, the result is perfectly determined. The three exact algorithms described above are classified as deterministic algorithms because their results are always identical to the global optimal solution. The greedy method is also a deterministic algorithm because its result is determined according to the problem to be solved. The result of the local search method can vary according to the initial solution; however, if the initial solution is fixed, its result is invariant. Therefore, the local search method is usually regarded as a deterministic algorithm. Only metaheuristics is a probabilistic algorithm, and the result obtained using this algorithm can differ every time it is applied to the same problem with the same setting.

Another viewpoint is using a constructive or repetition algorithm. This classification is usually applied to approximation algorithms. The greedy method is a constructive algorithm because its result is derived by combining small pieces of a solution for some limited parameters. In contrast, the local search method and metaheuristics are repetition algorithms because they repeatedly improve the initial solution. A constructive algorithm is generally quicker than a repetition algorithm. However, it only considers the local contribution of each parameter; thus, its result could be a local optimal solution. A repetition algorithm is more robust than a constructive algorithm because it considers all the parameters at once.

Metaheuristics is particularly superior to the local search method from the viewpoint of robustness because its result does not depend on the initial solution. However, a repetition algorithm consumes a considerably greater amount of computational time; thus, the time at which the search should be terminated becomes important. Approximation algorithms aim to find a sufficiently good solution in practical time and not the globally optimal solution. Hence, the acceptable range for a "good" solution will define the search speed of the algorithm; in addition, the range depends on the problem to be solved and the purpose of the analyst. The following are the major terminating criteria: (a) the solution that fulfills certain conditions is found; (b) the cycle is repeated a certain number of times; and (c) the pace of improvement in the solution decelerates. When using the repetition algorithm, we must specify the probable terminating criterion that is sufficient to obtain a good solution but does not consume much time.

2.2 Metaheuristics and its applicability to the model searching task

The framework to classify the method of solving the combinational optimization problem was presented in the last section. In this section, we restructure the existing algorithms for the model specification search in SEM from the viewpoint of the combinational optimization problem study and examine the type of solution method that is deemed promising.

2.2.1 Automated model specification search method for SEM as a combinational optimization problem

First, we consider the enumeration method. For the model specification search in SEM, the approach wherein all the possible subsets are selected (Bentler, 1995; Jöreskog & Sörborm, 1990), which considers all the possible models, corresponds to the enumeration method. However, it is evident that the enumeration method is not practical in most cases for the model specification search in SEM. It has not been proved whether the model specification search in SEM is definitely an NP-hard problem. However, the structure of the model specification search that requires us to find a good-fitting combination of constraints for the simultaneous equation within the confines of the degrees of freedom is similar to the knapsack problem, one of the representative problems in the study of combinational optimization problems (Papadimitriou & Steiglitz, 1982). Moreover, if there is an SEM model that estimates N parameters, the possible pattern of the model specification increases in the order of N^2 by adding one more parameter. This increasing rate is exponential, and hence, the time complexity of the model specification search in SEM is expected to grow at a basically exponential rate. Therefore, it is not an inappropriate assumption to treat the model specification search in SEM as an NP-hard problem. Then, the approach wherein all the possible subsets are selected is considered to work effectively only in situations where there are very few observed variables. For the same reason, it is difficult to apply the exact method to the model specification search in SEM. By its definition, linear programming cannot be used as has been described, and the branch and bound method and dynamic programming become inefficient.

Hence, the approximation method is considered to be suitable for the model specification search in SEM. Of the three types of approximation methods, the greedy method is not commonly used. This is because the parameters in SEM are correlated with each other, due to adding or removing a parameter affects the estimates of the existing parameters (Jöreskog & Sörborm, 1990). This makes it difficult to independently assess the goodness of a specific parameter; the greedy method is therefore unfit for the model specification search in SEM. In addition, we can express the correlation between two variables in several ways in an SEM model. The correlation can be formulated not only by assuming the direct relationship between variables by a unidirectional or bidirectional edge but also by constructing an indirect relationship through other variables. Hence, it is difficult to even determine the number of parameters to be used for the model specification search in SEM. This modeling flexibility is usually regarded as a benefit of SEM, but in this case, it is rendered unsuitable for the greedy method.

In reality, the most widely used combinational optimization method in the model specification search is the local search method. The automated model search algorithm for SEM model proposed by (Long, 1983) first determines the initial model and then improves this model by using the LM and Wald tests. However, we can change only one parameter at a time because changing one parameter's restriction will affect the estimates of all the other parameters. Thus, the search space is limited around the proposed model. This procedure is identical to the framework of the local search method, and the LM and Wald tests correspond to the neighborhood operation. Therefore, it has the same shortcoming as the local search method in that it is highly likely to arrive at the local minima (Harwood & Scheines, 2002). This independence on the initial solution has already been pointed out by MacCallum (1986) and is presently recognized as being a major difficulty in the automated model specification search using the LM and Wald tests.

Hence, applying metaheuristics, acknowledged as being considerably more robust than the local search methods (Sait & Youssef, 2000), is deemed an effective approach to establish a more reliable and accurate model search algorithm. In fact, not only does metaheuristics constitute a theoretical framework, it has already been widely applied to many practical problems and has shown successful results (e.g., Nonobe & Ibaraki (1999); Caprara, Fischetti, & Toth (1999)). Therefore, applying metaheuristics to the model specification search in SEM is expected to yield good results.

2.2.2 Some algorithms to achieve metaheuristics

Metaheuristics is essentially a generic term for non-deterministic repetition algorithms that are used to solve combinational optimization problems, and hence, it includes many types of algorithms. Therefore, when using metaheuristics, we have to choose the algorithm to be applied. This section describes various methods classified as metaheuristics. Some of these algorithms can be combined and used.

(a). Algorithms based on the greedy method

Randomized greedy method

One approach to achieve metaheuristics is by improving the greedy method. The typical greedy method is a deterministic algorithm that constructs a solution based on each parameter's local contribution to the target function and then chooses the best value of the parameters. The result is therefore invariant. The randomized greedy method is a modified version of the greedy method; this method chooses not only the best-fitting value of parameters but also the value of the parameters in proportion with each value's local contribution to the target function (Feo & Resende, 1995). This randomness achieves the diversification of the search; hence, the randomized greedy method is no longer a deterministic algorithm, but a probabilistic one.

Ant colony optimization

The search area of the randomized greedy method is not very large because the algorithm is still based on the local contribution of each parameter. To overcome this limitation, Dorigo (1992) proposed the ant colony optimization. In this case, some solutions are first generated by using the randomized greedy method and their goodness-of-fit is evaluated; each parameter's local contribution is then adjusted based on the evaluated goodness. By repeating this process, ant colony optimization can yield more accurate values of each parameter's local contribution and hence derive a more robust solution.

The adaptive multi-start technique (Boes, Kahng, & Muddu, 1994) also uses a similar approach. This method assigns a high score to the elements included in the good-fitting solution and attempts to combine the high-scored elements in order to derive the solution. However, the operational procedure of this technique is more similar to the genetic algorithm, which will be described below.

Genetic algorithm

The genetic algorithm (GA) is an optimization technique proposed by Holland (1975). The unique feature of this algorithm is that it is based on the mechanism of natural selection and is therefore also known as evolutionary computation (Bäck, Fogel, & Michalewicz, 1997). The underlying mechanism of the GA can be interpreted as an extension of the greedy method, but it is proposed independently of the greedy method.

In the GA, the solution of the target problem is treated as a set of certain parameters (Man, Tang, & Kwong, 1999). Each parameter is then regarded as a gene, and the set of all genes is referred to as a chromosome. With this analogy, the GA operates on the chromosomes instead of the solution itself, and based on the mechanism of natural selection, it attempts to identify the chromosome that represents the optimal solution. The details of this algorithm will be described in the next section.

The theoretical framework of the GA is more abstractive than that of other algorithms because it deals with chromosomes. Therefore, the GA is often used as a basis when constructing a generalized metaheuristics algorithm. Simulated evolution (Kling & Banerjee, 1991) and stochastic evolution (Saab & Rao, 1991) are versatile and powerful randomization methods that are based on the concept of the GA.

(b). Algorithms based on the local search method

Multi-start local search algorithm

The multi-start local search algorithm (MLS) is a generic term for the methods that prepare several different initial solutions and then execute the local search method using them. Subsequently, the best solution obtained through multiple repetitions of the local search method will be used as the final solution. This is a simple and old approach to implement metaheuristics (Reiter & Rice, 1966; Kernighan & Lin, 1970).

If the initial solutions are generated at random, the algorithm is known as a random multi-start local search. However, such a primitive approach is seldom used at present. The greedy randomized adaptive search procedure (GRASP) is one of the representative algorithms of the MLS (Feo & Resende, 1989; Feo, Venkartraman, & Bard, 1991). The GRASP uses the randomized greedy method to generate initial solutions.

The iterated local search (ILS) method is another approach to improve the MLS (Johnson, 1990; Martin, Otto, & Felten, 1991). ILS uses the result of one local search method as the initial solution for the next execution of the local search method. Occasionally, a random modification is added to the solution. This type of ILS is called the chained local optimization method (Martin & Otto, 1996). The randomness is reduced according to the progression of the search cycle in the variable neighborhood search method (Mladenović & Hansen, 1997), similar to the simulated annealing method described below.

Simulated annealing method

The simulated annealing method is an algorithm based on the physical process of annealing. Annealing is a technique used in metallurgy in which a material is first heated and then cooled carefully in a controlled manner to increase the size of its crystals and reduce their defects. In the annealing process, the heat causes the atoms to wander randomly from their initial positions, and the slow cooling allows them greater chances of finding configurations with lower internal energy than the initial one. Kirkpatrick, Gelatt Jr., & Vecchi (1983) and Cerny (1985) separately focused on this feature and applied the annealing mechanism to solve the combinational optimization problem. Therefore, their proposed methods were named the simulated annealing method.

When improving the current solution in the search cycle, the local search method always chooses the best solution in the neighborhood. In contrast, the simulated annealing method chooses the solution with a probability that depends on the goodness of each candidate. This allows the algorithm to choose a bad solution at a certain rate and prevent it from being trapped in the local minima. However, this randomness prevents searching from convergence. Therefore, the probability of choosing a bad solution is

gradually lowered during the search process. This adjustment is usually done by modifying the temperature parameter. Using this parameter, the simulated annealing method achieves the diversification and intensification of the search.
The simulated annealing method is very simple and has great versatility in that it can be applied to virtually all types of combinational optimization problems. However, it generally consumes a large amount of computational time, thus necessitating some improvements in the search speed. The most widely studied approach is one where the search process is parallelized (Darema-Rogers, Kirkpatrick, & Norton, 1987); in another approach, a more efficient algorithm is designed. Some algorithms including the threshold accepting method (Dueck & Scheuer, 1990) and the great deluge algorithm (Dueck, 1993) have been proposed based on the latter approach. However, the promotion of search efficiency depends largely on the specific problem, and therefore, making improvements to the general framework of the algorithm has only a limited effect. Hence, the simple simulated annealing method presently continues to be used as a significant algorithm in metaheuristics.

Tabu search

Tabu search is a generalization of the local search method proposed by Glover (1989, 1990). In the local search method, the search is terminated if no solution has a fit better than the current solution. In contrast, the tabu search chooses the best-fitting solution in the neighborhood of the current solution even if the candidates are inferior to those in the current solution. By adding this modification, the tabu search aims to avoid being trapped in the local minima. However, if the current solution is a local optimal solution, it is highly probable that after some repetition of the search cycle, the first local minima will be chosen again. This process is termed cycling. The tabu search employs another modification to avoid cycling, namely, a tabu list. The tabu list contains the solutions that have been visited in the recent previous cycles, and the algorithm will not choose the solutions recorded in this list as a destination when improving the current solution. If there is no tabu list, the algorithm will get trapped in the local minima. However, if all the solutions visited in the cycle are recorded in the tabu list, the algorithm will be unable to find a new solution to move to. Therefore, the length of lists memorizing the solution is very important in a tabu search. It is common to simultaneously use tabu lists of different lengths. Short term memory is said to be primarily concerned about the intensification of the search; and long term memory, about the diversification of the search (Sun & Mckeown, 1993). Moreover, the solutions themselves are not usually recorded in the tabu list. Instead, information on some attributes that summarize the solution is used to save computational time and working memory. These attributes are also known as aspiration criteria; further, some tabu lists that have a completely different term of memories and aspiration criteria should be combined in order to design an effective tabu search algorithm. Therefore, a tabu search is sometimes known as adaptive memory programming.

Search space smoothing method

The MLS, simulated annealing method, and the tabu search method are algorithms that introduce some fluctuation in the solutions generated in the estimation cycle. In contrast, the search space smoothing method (Gu & Huang, 1994) modifies the target function that is used to evaluate the goodness of the solution. The modification is made to smooth the difference in the target function; this implies that all the solutions obtain similar values of the target function. By this modification, the search space smoothing method aims to prevent being trapped in the local minima.
As in the simulated annealing method, the amount of modification is usually required to decrease according to the search cycle, but the orientation of the modification is always maintain throughout the estimation in the search space smoothing method. Charon & Hudry (1993) proposed the noising method that adds some randomness to the orientation of modification to the target function. This method is also known as perturbation and can be interpreted as a variation of ILS (Codenotti, Manzini, Margara, & Resta, 1996).

2.2.3 Which algorithm sould be applied for the model specification search in SEM?

As mentioned earlier, there exist many algorithms in metaheuristics and there are numerous previous studies that attempted to apply metaheuristics to the model fitting task. For example, Harwood & Scheines (2002) employed the GA to determine a direct acyclic graph. In this study, metaheuristics yielded excellent results in simulation studies. Other studies have applied the reversible jump Markov chain Monte Carlo method for a multiple change-point analysis(Green, 1995), and employed the tabu search for variable selection in a multiple regression analysis (Mills, Olejnik, & Marcoulides, 2005). These results were also good.

We now decide which algorithm is considered suitable for the model specification search in SEM. Marcoulides, Drezner, & Schumacker (1998) used the tabu search method for the model specification search in SEM. However, we decided to focus on the GA in the present study. While the theory of the GA is still in the development stage, many studies have already been conducted and it has already been used in applications with some success due its simplicity and robustness (e.g., Gabbert, Brown, Huntley, Markowicz, & Sappingston (1991); Syswerda & Palmucci (1991); Chambers (1995)). Therefore, applying the GA to the model specification search in SEM is expected to yield good results. Moreover, Marcoulides & Drezner (2001) indicated that the GA could be applied to the model specification search in SEM; however, they did not test this. The basic aim of our study is to test whether the GA can be applied to the model specification search in SEM. Undoubtedly, it is possible that other algorithms would work better than the GA for this purpose; however, as a first step, it is important to verify whether GA, which is one of the most widely applied and popular methods in metaheuristics, is capable of application.

2.3 Basic mechanism of the genetic algorithm

2.3.1 Chromosome representation

As described in the earlier section, the GA is based on the genetic phenomena in the real world. The gene is the fundamental unit of information in living systems; however, more than a one gene governs the characteristics of living things. Usually, multiple genes comprise a structure called a chromosome, and this chromosome determines the features of our body, behavior, and personality. The combination of genes constituting the chromosome is called the genotype, and the visible property defined by the chromosome is called the phenotype. Therefore, the same character is represented in two different ways in our reproduction process. This duality also holds true in the GA.

When using the GA, we must express the solution of the target problem as a set of certain parameters (Man et al., 1999). The GA treats each parameter as a gene and refers to the set of all genes as a chromosome. For example, if the target problem is to minimize the function $f(x_1, x_2)$, the value of the parameters x_1 and x_2 could be the genes and the compilation of these genes is regarded as the chromosome. With this analogy, the GA operates on the chromosomes instead of the solution itself and attempts to identify the chromosome that represents the optimal solution based on the mechanism of natural selection. Hence, determining the coding rule in order to encode a solution to a chromosome will be of importance in GAs.

The oldest encoding method is bit string encoding (Holland, 1975) which represents the information in combination of 0s and 1s. In bit string encoding, a number is expressed in binary form. For example, if the value of $x_1 = 5$, the value of the corresponding gene would be 101. Bit string encoding is also intended for representing the logical value. Hollstien (1971) investigated the use of gray codes, which are also a combination of binary numbers, but in this case, only one bit changes from one entry to the next. Bit string encoding can handle gray codes, and it works effectively for some types of problems.

Recently, real-value encoding has been introduced (Wright, 1991). It is a natural and straightforward representation of problems having real-value parameters. In this encoding rule, the gene for $x_1 = 5$ would directly be 5. However, the theory and operation of the conventional GA are developed based on bit string encoding, and there is no consensus on the use of real-value encoding. Some researches have indicated that it yields good results (Michalewicz, 1996), while some have indicated it would not be effective (Goldberg, 1990a). Hence, we must be careful in using this type of encoding rule.

Another viewpoint with respect to the coding rule is how to connect individual genes and construct a chromosome. The most classic method is to simply connect genes by treating them as character strings. For example, the gene for $x_1 = 5$ and that for $x_2 = 34$; then, the corresponding chromosome would be $[x_1\ x_2] = [5\ 3\ 4]$. However, if the problem has a particular structure, a problem-oriented representation of the chromosome is expected to work effectively. For example, a matrix-based representation (Cohoon & Paris, 1987) and an order-based representation (Davis, 1991) are proposed for certain problems.

2.3.2 Search cycle of the GA

After deciding the encoding rule, we perform the search cycle of the GA. The basic procedure of the GA is as follows.

1. **Initialization:** Generate individuals at random to form the initial population.
2. **Reproduction:** Select some individuals from the population and apply a crossover operator and mutation operator to create new individuals.
3. **Replacement:** Merge the existing individuals included in the population and newborn individuals created through the reproduction process. Then, select individuals with good chromosomes and reform the population.
4. **Repetition:** Return to the reproduction process if a stopping criterion is not satisfied.

Initialization process

The GA treats not a single chromosome but a pool of certain different chromosomes. An entire set of chromosomes is regarded as a population and each chromosome in the pool is regarded as an individual living in the world. The initial step of the GA involves generating individuals and forming the initial population.

Generally, the chromosomes of these individuals are randomly decided, although the specific procedure depends on the chromosome format. Thus, it is important to decide the encoding rule with regard to how information about the solution of the target problem is encoded into the chromosome. Sometimes, the solutions obtained from different optimization algorithms are used as the initial population of the GA. This method is known as seeding (Davis, 1991).

The size of the population is also an important consideration. The population size is equal to the variety of chromosomes maintained in the population and is directly linked to the accuracy of the search. However, it is also directly linked to the calculation amount (Youssef, Sadiq, Nassar, & Benten, 1995); hence, we must choose an appropriate population size to solve the target problem.

Reproduction process

The reproduction process constitutes the core of the optimization conducted by the GA. This process should be designed to evoke the survival of the fittest mechanism, which suggests that the individual with a better chromosome is more likely to produce a greater number of offspring. This process is thus expected to generate a better solution.

To complete this process, we must initially determine the degree of goodness of each chromosome. However, this is entirely dependent on the target problem, and there is no general rule that can be applied in this determination. Usually, some positive-valued index is used for the goodness. Goodness is also known as fitness because a good chromosome would imply good adaptation to the environment.

(a). Parent selection

First, we have to select individuals for mating. Of the many selection methods proposed, virtually all are based on the goodness of the chromosome and often stochastically select the better individuals. However, simply using the value of fitness as a selection ratio leads to the lack of diversity when there is one outstanding individual in the population. This is because this superior chromosome tends to be selected too often. To avoid this, an operation known as scaling is sometimes applied to the each individual's value of fitness before the parents are selected.

Linear scaling (Goldberg, 1989) adjusts the fitness by using a linear function, and power law scaling (Gillies, 1985) uses a power function. Each method aims to bring the values of goodness close. In addition, linear scaling can produce a negative value of fitness. Fitness values are however not expected to be negative. In such cases, a negative value of fitness is usually truncated to zero. This operation is known as sigma truncation.

The simplest method for parent selection is the roulette wheel selection algorithm. This algorithm first makes a virtual roulette that represents the proportion of the degree of fitness of all the population members such that good chromosomes encompass a wide area on the roulette wheel and bad chromosomes cover a narrow area. Then, the wheel is spun and two individuals are selected to be parents. Roulette wheel selection is considered to be a method of stochastic sampling with replacement. Therefore, the sampling result obtained would be biased if there are only a few trials. To deal with this problem, some modifications were proposed. Stochastic sampling with partial replacement is an algorithm that reduces a chromosome's segment on the roulette wheel if the chromosome is selected. Stochastic universal sampling is another algorithm with zero bias (Man et al., 1999).

Further, ranking selection (Baker, 1985) is an algorithm for parent selection. It selects individuals by the rank of their fitness instead of the values of fitness. The purpose of this algorithm is similar to scaling. In addition, tournament selection (Goldberg, 1990b) was proposed as an algorithm that simultaneously uses roulette wheel selection and ranking selection. In this algorithm, a subset of the population is first extracted and the individuals to be parents are then sampled from this subpopulation. The methods described above are expected to be more unbiased than roulette wheel selection.

(b). Crossover

After selecting individuals for breeding, the next step is generating new chromosomes from the chromosomes of the selected parents. This is equivalent to crossover in the natural world. As is the case with selection, many methods for crossover are proposed, but these typically split the parents' chromosomes into small parts and combine them to create the offspring's chromosome.

The simplest algorithm for crossover is the one-point crossover. The one-point crossover divides the chromosome of the parents into two and recrosses them with each other. For example, if the parents' chromosomes are

$$\text{Parent 1} : [1\ 1\ 2\ 3\ 2\ 3]$$
$$\text{Parent 2} : [1\ 2\ 2\ 3\ 3\ 2],$$

and if the cutting point occurs after the second gene, then the offspring's chromosomes will be

$$\text{Child 1} : [1\ 1\ 2\ 3\ 3\ 2]$$
$$\text{Child 2} : [1\ 2\ 2\ 3\ 2\ 3].$$

Child 1 inherits genes from the first half of parent 1 and the second half of parent 2. In contrast, child 2 inherits genes from the second half of parent 1 and the first half of parent 2.

The position of the cutting point is not necessarily fixed. Sometimes, it is randomly determined each time the crossover operation is executed. Moreover, we can also use more than one cutting point. This type of algorithm is generally called the multi-point crossover. The most extreme version of the multi-point crossover is the uniform crossover. This algorithm produces offspring from the parents' chromosomes, based on a randomly generated binary mask. The mask should be of the same length as the parents' chromosomes, and the value of the genes is randomly determined based on the Bernoulli distribution, which has a success rate of 0.50. For example, if the parents' chromosomes are

$$\text{Parent 1} : [1\ 1\ 2\ 3\ 2\ 3]$$
$$\text{Parent 2} : [1\ 2\ 2\ 3\ 3\ 2],$$

then the mask might be determined as follows.

$$[1\ 1\ 0\ 0\ 1\ 0]$$

Certainly, each value constituting the mask can vary from case to case, but with the mask indicated above, the offspring's chromosomes will be

$$\text{Child 1} : [1\ 2\ 2\ 3\ 3\ 3]$$
$$\text{Child 2} : [1\ 1\ 2\ 3\ 2\ 2].$$

This means that child 1 inherits genes from parent 1 where the mask value is 0 and from parent 2 where the mask value is 1. Conversely, child 2 inherits genes from parent 1 where the mask value is 1 and from parent 2 where the mask value is 0. The mask is randomly determined each time the crossover operator is executed; hence, the combination pattern of the parents' chromosomes varies at every crossover.

Determining which algorithm should be used is a difficult problem and basically depends on the task to be solved. The investigation conducted by Spears & DeJong (1991) revealed that the two-point crossover could perform poorly as compared to the multi-point crossover. In contrast, DeJong (1975) reported that the two-point crossover is the most optimal number for the multi-point crossover.

If the chromosome is encoded to be a permutation of a finite number of elements, the crossover algorithms described above sometimes generate offspring that have certain elements redundantly or do not contain certain elements at all. These chromosomes are inappropriate, and therefore, we need to employ a special algorithm in such a situation. The partially mapped crossover (Goldberg & Lingle, 1985), order crossover (Davis, 1985), and cycle crossover (Oliver, Smith, & Holland, 1987) are the crossover algorithms to be used in the case of a permutational chromosome. We now provide details on the partially mapped crossover.

In the partially mapped crossover algorithm, the genes underlying the cutting points are exchanged, similar to the case of the two-point crossover that was performed initially. For example, if the parents' chromosomes are

$$\begin{aligned}\text{Parent 1} &: [a\ b\ c\ e\ f\ d]\\ \text{Parent 2} &: [e\ d\ f\ b\ c\ a],\end{aligned}$$

and if the cutting point occurs after the third gene, then the second part will be changed to

$$\begin{aligned}\text{Child 1} &: [a\ b\ c\ b\ c\ a]\\ \text{Child 2} &: [e\ d\ f\ e\ f\ d].\end{aligned}$$

Then, the genes in the same position included in the second half are treated as a pair. Here, (b, e), (c, f), and (a, d) are the pairs of genes. If the first half of the children's chromosomes contains genes in those pairs, they will be changed to the corresponding value. For example, child 1 contains a in the first half of its chromosome that will be changed to d because a and d form a pair. After repeating the same procedure in the case of the all the genes in the first half of the children's chromosomes, their chromosomes will be

$$\begin{aligned}\text{Child 1} &: [d\ e\ f\ b\ c\ a]\\ \text{Child 2} &: [b\ a\ c\ e\ f\ d].\end{aligned}$$

This is the result of the partially mapped crossover. We can generate offspring with chromosomes that are the permutations of elements a to f.

(c). Mutation

The final process of reproduction is mutation. Mutation changes chromosomes randomly with a certain probability. Usually, the mutation rate is not very high, but it is an essential process in the GA (Sait & Youssef, 2000). The determination of the mutation rate therefore affects the accuracy and speed of the GA.

Replacement process

After reproduction, we must replace the old population with newborn offspring. This step is known as replacement. The major problem associated with replacement is how many individuals should be replaced by offspring at a particular time. This replacement rate is known as the generation gap, and the target problem determines which generation gap is the most effective. Basically, replacing the current population completely is considered to be effective (Grefenstette, 1986). However, a steady-state GA that replaces the current population after each reproduction reduced computing workload (Sait & Youssef, 2000).

Another problem pertains to the criterion to determine the individuals to be replaced by offspring. There are two main types of replacement methods: deciding randomly and deciding based on the goodness of the chromosome. However, in either case, if all individuals were chosen to survive stochastically, a good chromosome in a certain population might fail to be retained, leading to the optimization of inefficiency. To avoid this, Jong (1975) proposed the algorithm known as elitism. Elitism is a strategy to retain some individuals with good chromosomes and not permit them to be replaced preferentially. This strategy is believed to improve the performance of optimization.

Repetition process

A single execution of the reproduction and replacement processes is termed one generation; this is the basic unit of iteration in the optimization cycle of the GA. Therefore, this cycle will be repeated until certain stopping criteria are fulfilled. The possible criteria are (a) finding an individual with a sufficiently good chromosome, (b) repeating the cycle a certain number of times, and (c) stopping the estimation when the pace of chromosome improvement slows. This is not a problem specific to the GA but is relevant to all the iterative optimization algorithms.

2.3.3 Some topics related to the convergence of the algorithm

The most remarkable feature of the GA is that individuals with relatively bad chromosomes are intentionally retained and merged randomly into the reproduction process. This widens the search area outside the neighborhood of the initial population. This is also supported by mutation. As a result, the GA can search a wider area than the local search method, which increases the possibility of finding a global optimal solution independent of the initial population. This feature is known as internal parallelism (Holland, 1975).

The convergence process of the GA can be treated as the Markov chain; numerous researches have been conducted on this process (Fogel, 1992). Many studies arrived at the result that by using bit string encoding with parent selection, crossover, and the mutation operator, the GA can find the global optimal solution if the search cycle is repeated a sufficient number of times. However, if the algorithm lacks at least one of these three operators, convergence is not assured. The effects of each operator are investigated from the perspective of the schema theory (Holland, 1975), which examines how the operator preserves and destroys the meaningful components in the chromosome.

However, because it is more likely to expand the search area using the simple GA, it is not suitable for a careful examination of a narrow area. The GA can be considered as an algorithm that places greater emphasis on the diversification than the intensification of the search. Therefore, a simple GA suffers from the disadvantages of slow convergence in practical situations. To deal with this, the local search method is generally combined with the GA at present (Sait & Youssef, 2000). These hybrid methods are sometimes known as genetic local search (GLS) methods to distinguish them from the simple GA.

In practice, parameter setting is also of significance. One of the attributes of metaheuristics is that its performance changes according to the parameters (Nonobe & Ibaraki, 1998). In the GA, the population size and the mutation rate are the most important parameters. However, other parameters such as the generation gap also affect the convergence speed. Therefore, seeking effective parameter settings is a critical process in constructing the GA for specific problems. Some studies that have attempted to adjust parameters automatically in the estimation process itself may be helpful in tackling this issue (e.g., Beasley (1993); Nonobe & Ibaraki (1998)), but these are still in the development stage.

Chapter 3

Model specification search using a genetic algorithm with factor reordering for a simple structure factor analysis model

3.1 Basic idea to apply the GA to the model specification search in SEM

As described in the previous section, when using the GA, we must encode the solution of the target problem in chromosome format. In this section, we wish to apply the GA to the model specification search in SEM. Therefore, encoding information on model specification into the chromosome is required.

Marcoulides & Drezner (2001) demonstrated an important method of overcoming this problem and indicated the capability of applying the GA to model specification search in SEM. Their basic idea is to use a dummy matrix that represents whether or not the parameter is freely estimated. For example, we consider the situation that we wish to search the factor pattern for the factor analysis model. The confirmatory factor analysis model can be represented as

$$\boldsymbol{x} = \boldsymbol{\mu} + \boldsymbol{A}\boldsymbol{f} + \boldsymbol{e},$$

where $\boldsymbol{x}$ is the observed variable vector; $\boldsymbol{\mu}$, the population mean vector; $\boldsymbol{A}$, the factor loading matrix; $\boldsymbol{f}$, the factor vector; and $\boldsymbol{e}$, the error variable vector. Then, with the assumptions $E[\boldsymbol{f}] = \boldsymbol{o}$, $E[\boldsymbol{e}] = \boldsymbol{o}$, and $E[\boldsymbol{f}\boldsymbol{e}'] = \boldsymbol{O}$, the covariance structure of the factor analysis model is represented as follows:

$$\boldsymbol{\Sigma}(\boldsymbol{\theta}) = \boldsymbol{A}\boldsymbol{\Sigma}_{\boldsymbol{f}}\boldsymbol{A}' + \boldsymbol{\Sigma}_{\boldsymbol{e}}, \tag{3.1}$$

where $\boldsymbol{f}$ is the factor vector, $\boldsymbol{A}$ is the factor loading matrix, $\boldsymbol{e}$ is the error variable vector, $\boldsymbol{\theta}$ is the parameter vector, $\boldsymbol{\Sigma}_{\boldsymbol{f}}$ is the factor covariance matrix, and $\boldsymbol{\Sigma}_{\boldsymbol{e}}$ is the error covariance matrix.

Then, we generate matrix $\boldsymbol{A}^*$, which is identical in size to $\boldsymbol{A}$. Each element of $\boldsymbol{A}^*$ represents whether its corresponding elements are to be fixed or freely estimated. If an element of $\boldsymbol{A}$ is to be estimated freely, the value of the element in an identical position $\boldsymbol{A}^*$ will be “one”. In contrast, if this is not the case, the value of the element of $\boldsymbol{A}^*$ will be “zero”. Finally, vec$[\boldsymbol{A}^*]$ becomes a character string that represents model specification. Hence, we can treat this vec$[\boldsymbol{A}^*]$ as a chromosome.

For example, if the true model is a factor analysis model with two factors and four variables as depicted in Figure 3.1, we have to estimate factor loadings from f_1 to v_1, f_1 to v_3, f_2 to v_2, and f_2 to v_4. The other factor loadings should be fixed at zero.

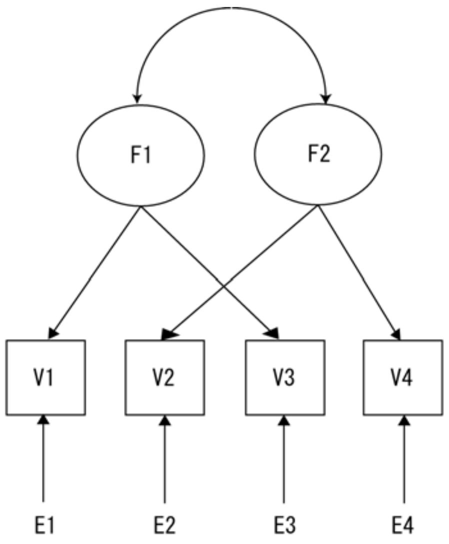


Figure 3.1: Factor analysis model with two factors and four variables

Therefore, if we express factor loading with λ, the factor loading matrix $\boldsymbol{A}$ can be represented as follows.

$$\boldsymbol{A} = \begin{bmatrix} \lambda_{11} & 0 \\ 0 & \lambda_{22} \\ \lambda_{31} & 0 \\ 0 & \lambda_{42} \end{bmatrix}.$$

Then, the dummy matrix $\boldsymbol{A}^*$ will be

$$\boldsymbol{A}^* = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix},$$

and corresponding chromosome will be

$$\text{vec}[\boldsymbol{A}^*] = [1\ 0\ 1\ 0\ 0\ 1\ 0\ 1].$$

We can use the same approach for other part of the model. For example, if we want to search model specifications with regard to factor covariance or error covariance, we only have to generate similar matrices such as $\boldsymbol{\Sigma}^*_{\boldsymbol{f}}$ and $\boldsymbol{\Sigma}^*_{\boldsymbol{e}}$, and establish a relation of vec$[\boldsymbol{\Sigma}^*_{\boldsymbol{f}}]$ and vec$[\boldsymbol{\Sigma}^*_{\boldsymbol{e}}]$ with vec$[\boldsymbol{A}^*]$. Any model in SEM can be represented in the matrixical form, and thus, this can be considered as the most general encoding method that converts information on model specification in SEM in the chromosome format.

However, the specific algorithm itself indicated by Marcoulides & Drezner (2001) was rather primitive, and therefore, applying it without any modifications to the real data analysis may create problems with regard to the accuracy and efficiency of the model search. The GA is a comparatively simple algorithm and is easy to apply to certain problem; however, this also is the reason why it is necessary to tune the implemented algorithm in accordance with the nature of the target problem in order to perform quick and accurate search (Man et al., 1999).

The first problem with the algorithm by Marcoulides & Drezner (2001) is that it does not consider model complexity. This creates a tendency to choose models with a low degree of freedom. In light of this, we usually do not attempt to employ such a complicated model in a practical situation; instead, we employ a simpler, easier to interpret one. The second problem is that the authors do not account for equivalent models. Equivalent models are different in path diagrams and represent different interpretations of the data but have the same estimated covariance matrix. Therefore, we cannot distinguish them from the value of the fit indexes. The third problem relates to the rotational indefiniteness of factors. Many models assumed in SEM contain factors, particularly in the field of psychology. Further, there could be models that have different factor loading patterns but substantially represent the same structure. For example, Figure 3.2 illustrates the factor analysis model in which v_2 and v_4 consist of one factor and v_1 and v_3 consist of another. This structure is essentially the same as the factor structure of the model presented in 3.1. However, the two models have a different factor loading matrix. Performing a model specification search without considering the abovementioned problems may disturb the search.

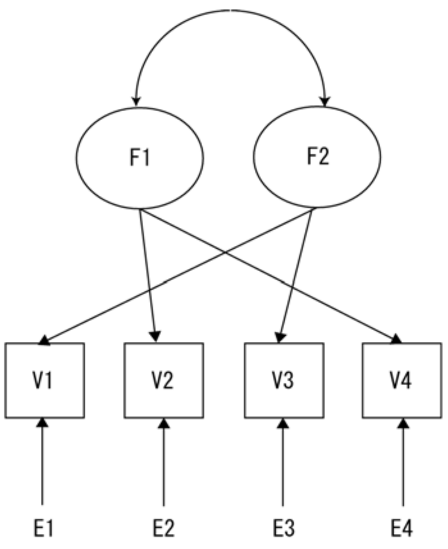


Figure 3.2: Another factor analysis model with two factors and four variables

Moreover, we must consider the calculation amount when using the GA because it generally requires a considerably greater amount of processing time than other algorithms categorized as metaheuristics (Youssef et al., 1995). Thus, the implemented algorithm should be carefully designed to demand less computational effort; otherwise, the task-elapsed time of the search will increase, making it more difficult to put the GA-based model specification search into practical use.

In view of these problems, to confirm the availability and performance of the GA for the model specification search in SEM from a practical standpoint, we confined the search space to a factor analysis model in this study. This restriction should decrease the computational effort in the search and solve problems that are unique to the model specification search in SEM but those which are not considered in the algorithm of Marcoulides & Drezner (2001). A more detailed explanation of these advantages will be provided later. Although reducing the search space damages the generality of the discussion to some extent and makes it impossible to refer to the model specification search in the general SEM, the factor analysis model is very popular in the field of psychology. Therefore, we expect our study to be valuable particularly from the standpoint of application.

I also focus attention on parameter setting in the GA. As far as it has been described, the performance of the GA is influenced greatly by its parameters. Despite its importance, very few earlier studies have explored the effect of parameter setting for model specification search in a factor analysis model or a general SEM. Therefore, we examined the parameter setting in a simulation study and then considered the real data analysis application.

3.2 Search algorithm for a simple structure factor analysis model with factor reordering

In this section, we will provide a detailed description of the GA that we constructed. The purpose of this study is to propose an algorithm for the model specification search of a factor analysis model. However, the algorithm described below is actually a search algorithm to be used only for a simple structure factor analysis model. This is because we intend to address the problems explained in the previous section.

Researchers tend to use the model search method when they usually have limited knowledge about their data. In such a situation, confining the search space to a simple structure appears extremely restrictive and is inadequate. Nevertheless, we employed this restriction because we were able to find a sufficiently practical and good-fitting factor analysis model by using an existing automated model search method, which begins from a simple structure factor analysis model found by the GA. It is evident that real data rarely has a completely simple structure; however, in a practical situation, we often strive to obtain a simple structure because a simple structure model is easier to interpret. Thus, a procedure that first finds a good-fitting simple structure factor analysis model and subsequently explores around this model is not very different from the real data analysis situation and also meets the purpose. Accepting this procedure, we confined the search space of our algorithm to a simple structure factor analysis in this chapter. The algorithm for the general factor analysis model will be described in the next chapter.

3.2.1 Encoding method

First, we have to determine the rules for encoding information about model specification into the chromosome. Here, we introduce the following five setups.

1. Limit the search space to a simple factor, first-order factor analysis model.

2. Fix the number of observed variables and factors prior to starting the search.

3. Fix the factor's variance at one to identify the confirmatory factor analysis model.

4. Always estimate all the covariances among factors because it is highly unlikely that there are completely no correlations among factors.

5. Do not estimate the error covariance among observed variables.

With these assumptions, whether elements are to be fixed or free is determined for $\boldsymbol{\Sigma_f}$ and $\boldsymbol{\Sigma_e}$ in Equation 3.1. From assumptions 3 and 4, the diagonal elements of $\boldsymbol{\Sigma_f}$ are to be fixed at one, and other elements are to be freely estimated. From assumption 5, the diagonal elements of $\boldsymbol{\Sigma_e}$ are to be free, and other elements are to be fixed at zero.

Consequently, we have to only consider the elements of matrix $\boldsymbol{A}$. Since $\boldsymbol{A}$ is a factor loading matrix, whether or not its parameter is estimated relates to whether or not each observed variable measures the corresponding factor. Further, from assumption 1, we only need to deal with the simple structure, implying that every observed variable measures only one factor. Thus, we can use the information on which factor is measured by the observed variables as the value of the chromosome. For example, if $\boldsymbol{A}$ is

$$\boldsymbol{A} = \begin{bmatrix} \lambda_{11} & 0 & 0 \\ \lambda_{21} & 0 & 0 \\ 0 & \lambda_{32} & 0 \\ 0 & \lambda_{42} & 0 \\ 0 & 0 & \lambda_{53} \end{bmatrix}, \tag{3.2}$$

then the corresponding chromosome is derived as follows.

$$[1\ 1\ 2\ 2\ 3]. \tag{3.3}$$

This is the coding rule employed in this study. By limiting the search target to a simple structure model, the search area is drastically reduced as compared to the search for a general factor analysis model. Hence, we can expect to reduce the calculation amount. This can also deal with the problem of model complexity because all the candidate models eventually have the same degree of freedom.

However, there continue to be problems with regard to the equivalent model and the rotational indefiniteness of factors. To deal with these problems, we introduced a factor reordering operator that reorders factors in ascending order corresponding to the observed variables assigned to the youngest number among the observed variables that measure the same factor. For example,

$$\boldsymbol{A} = \begin{bmatrix} 0 & 0 & \lambda_{13} \\ 0 & 0 & \lambda_{23} \\ 0 & \lambda_{32} & 0 \\ 0 & \lambda_{42} & 0 \\ \lambda_{51} & 0 & 0 \end{bmatrix}$$

will be reordered in the same manner as Equation 3.2; thus, the corresponding chromosome will be the same as given in Equation 3.3. This reordering operator was always applied when the algorithm generated a new chromosome.

3.2.2 Search cycle

Initialization process

According to the coding rule, the chromosome should be a character string consisting of integer numbers up to the number of factors. Thus, the chromosomes of individuals in the initial population can be generated based on the random variables sampled from the multinomial distribution.

However, we must be aware of the possibility of generating individuals with chromosomes representing an inestimable model specification. This can occur not only in the initialization process but also in the reproduction process. In this study, such individuals were removed immediately from the population. Alternatively, initialization or reproduction was re-executed, if necessary. Individuals with fewer factors than the number assumed at the beginning of the search were also eliminated.

Reproduction process

To complete this process, we must employ some method to determine each individual's degree of goodness, and fortunately, there are many fit indexes in SEM that can be used. In this study, we chose RMSEA as an index to denote the goodness of the chromosome. However, as the RMSEA value becomes smaller with the improving model fit and the optimum value of RMSEA becomes zero, its magnitude represents the "badness" of chromosome rather than the goodness. Hence, we actually used the inverse of RMSEA; when RMSEA $= 0$, we regard the value of the goodness as $1e + 99$.

As a method of selection, we employed the roulette wheel selection algorithm and as a crossover method, we employed the uniform crossover algorithm. These algorithms have been described in the previous chapter. In addition, the mutation operator was applied. Each gene of the children's chromosomes is checked individually based on a certain mutation rate, and if mutation occurs, the value of that gene will be changed. For example, if the number of factors is three and the present value of the gene is two, the value will change to one or three.

Replacement process

In this study, it was decided that individuals would be replaced stochastically based on the goodness of the chromosome. We first generated as many offspring as the original population size and then randomly selected individuals who would survive into the next generation from the existing individuals in the population and the newborn offspring, in proportion to the goodness of their chromosome. However, since the probabilistic sampling was without replacement, each individual surviving into the next population had a different chromosome and there were no overlaps. Moreover, elitism was employed to ensure that a good chromosome in a certain generation does not fail to pass on information to the succeeding generation. Only one individual with the best chromosome was retained as elite in every generation.

Repetition process

In practical situations, we do not know the optimum value of the RMSEA for the target data. Therefore, we stopped the estimation when the pace of improvement in the goodness of a chromosome decelerated. In particular, if the RMSEA of the best fitting individuals in a population did not improve for three consecutive generations, we terminated the search and adopted the best chromosome as a result of the search. In addition, we also stopped the search upon finding a chromosome for which the RMSEA was zero or when the iteration number of the search reached 50.

3.2.3 Local search method

To accelerate the convergence, we also adopted a local search method. In the implemented procedure, we searched the 1-flip neighborhood of the existing model such that the model could be better explored by changing only one gene of the base model. This method is indicated in Marcoulides & Drezner (2001), but with one difference: in the present study, the search was executed in order from the first gene, and if a better-fitting model than the base model was found, the search was stopped and that model was adopted as the result of the local search. For example, if the number of factors is assumed to be three and the chromosome of the base model is

$$[1\ 2\ 2\ 3\ 3\ 1],$$

we then examined two models, the chromosomes of which are

$$\begin{gathered}[2\ 2\ 2\ 3\ 3\ 1]\\ [3\ 2\ 2\ 3\ 3\ 1].\end{gathered}$$

If either of these models had a better RMSEA than the first model, it was adopted and the search was complete. However, if the RMSEA did not improve, then we attempted to change the second gene and examined the two models as follows.

$$[1\ 1\ 2\ 3\ 3\ 1]$$
$$[1\ 3\ 2\ 3\ 3\ 1]$$

Because this method does not search the 1-flip neighborhood completely, the possibility of overlooking the best-fitting model could not be eliminated. However, to examine all the possible models in the 1-flip neighborhood and choose the best one as in Marcoulides & Drezner (2001) requires a great deal of time; further, the loading increases according to the number of observed variables and factors. Therefore, for the application of the real data analysis, we employed a simpler but faster method.

3.3 Simulation study

3.3.1 Method

We conducted a simulation study to verify the proposed algorithm. We obtained simulation-generated data from a known true structure and examined whether or not it is possible for the GA to find the true model from that data. For the true structure, we prepared three different types of setup: (a) simple factor analysis model on 3 factors with 12 observed variables where each factor is measured by 4 observed variables and all the factors are correlated, (b) one more factor loading is added to the first model such that only one factor is then measured by 5 observed variables, and (c) in addition to the first model, each factor is again measured by an additional observed variable, thus three paths for factor loading are added.

At each replication in the simulation study, we first generated a true covariance matrix from a randomly determined true structure following the abovementioned setups. We randomly decided which variables would measure which factor and stochastically determined all the parameters according to continuous uniform distributions, the ranges of which were as follows: (a) the factor loadings were to lie within 0.6 to 0.9; (b) the factor correlations, within 0.2 to 0.6; and (c) the residual variances, within 0.6 to 0.9. Subsequently, we generated 1,000 samples based on a multivariate normal distribution where the covariance matrix was equal to the true covariance matrix determined; these samples were used as data for the replication.

When the true structure was a simple one, we measured the success or failure of the GA search according to whether or not the true model and the model that was found had an identical factor pattern. However, when the true structure was not a simple one, we judged the search to be successful if the estimated factor pattern was included in the true factor pattern, i.e., if the observed variable 1 measures factors 1 and 3 in the true pattern, the estimated pattern in which variable 1 measures factors 1 or 3 is fine, but where it measures factor 2 is regarded as a failure. In this study, we considered not only the success and failure of the search but also the elapsed generations and CPU time.

Further, we ran the model specification search under certain different settings of the GA parameters and compared the results. In particular, we changed the population size and the mutation rate. The number of populations was set at 5, 10, and 20, and the mutation rates were set to 0.01, 0.05, and 0.10. All the other details of the algorithm were left unchanged and were the same as described in the previous section. The chromosomes of the individuals in the initial population were generated based on the trinomial distribution for which each result is equally probable. The mask for the crossover was generated from the binomial distribution, which has a success rate of 0.50, and the number of trials was 12.

We used a total of 9 patterns of parameter settings for GA and 3 types of true structures for data generation. Thus, there were a total of 27 conditions in the simulation study. We repeated data generation and the model specification search 50 times for each condition. All the calculations were performed on Windows-based R (R Development Core Team, 2006), and the SEM package (Fox, 2006) was used to compute the RMSEA.

3.3.2 Results

Table 3.1 shows the success rate of the model specification search. When the true data approaches a simple structure, the exploration algorithm is effective. The success rate increases as the population size increases and exceeds 0.90. When the true structure is not a simple one, the success rate becomes comparatively lower, but it still remains near 0.50. Therefore, it is probable that the proposed algorithm can find the best-fitting model. Figure 3.3 presents an example of the search process when the true data is a simple structure; the population size, 20; and the mutation rate, 0.10. A good-fitting model suddenly appears in the search process, and other individuals' fitness is improved in later generations due to the best model.

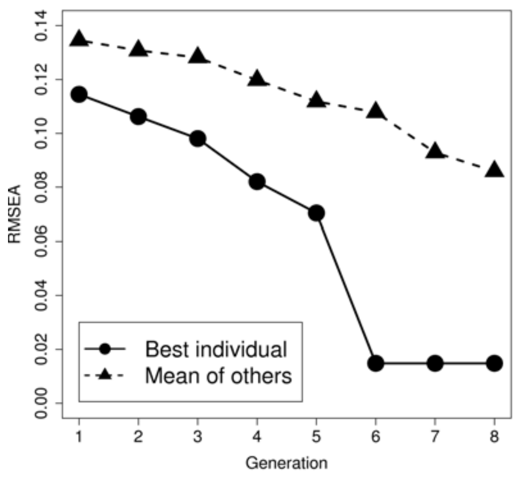


Figure 3.3: One example of the historical trend in RMSEA during the GA search process

Table 3.1: Success rate in finding the true model

Mutation rate	Population size		
	5	10	20
Simple Structure			
0.01	0.68	0.84	0.96
0.05	0.80	0.80	0.92
0.10	0.74	0.88	0.94
Adding One Path to the Simple Structure			
0.01	0.82	0.88	0.86
0.05	0.76	0.76	0.92
0.10	0.60	0.80	0.94
Adding Three Paths to the Simple Structure			
0.01	0.36	0.50	0.46
0.05	0.44	0.42	0.48
0.10	0.38	0.44	0.48

Table 3.2 shows the means and standard deviations (SDs) of the number of generations for which the algorithm was able to find the true model. From this table, we can state that with a large population size, there was a reduction in the number of elapsed generations of the GA. However, this does not imply that large the population size improves the search speed. Table 3.3 shows the means and SDs of CPU time for which the algorithm was able to find the true model. This table contains only the results for when true data was a simple structure. The Athlon 64 3200+ computer that we used had a 2GB memory. Table 3.3 shows that the calculation time increases with an increase in the population size. This is because most of the computational effort was devoted to calculating the RMSEA in this algorithm. Thus, the search is quicker with a larger number of generations and a smaller population size.

Table 3.2: Means and SDs of the genetic cycles involved in finding the true structure

Mutation rate	Population size		
	5	10	20
	Simple Structure		
0.01	6.74(2.08)	6.62(1.81)	5.16(1.39)
0.05	6.28(1.68)	6.33(1.56)	5.48(1.80)
0.10	6.35(1.93)	6.11(1.99)	5.68(1.84)
	Adding One Path to the Simple Structure		
0.01	8.46(1.61)	7.77(1.51)	7.02(1.54)
0.05	8.66(2.13)	8.42(1.98)	6.54(1.17)
0.10	8.30(1.59)	7.45(1.57)	6.89(1.34)
	Adding Three Paths to the Simple Structure		
0.01	8.06(2.31)	8.20(1.66)	7.43(1.85)
0.05	8.59(1.99)	7.48(1.57)	6.92(1.32)
0.10	8.11(1.70)	8.09(1.41)	7.79(1.64)

Note. The replications that failed to find the true structure are not included. SDs are given in parentheses.

Given the above results, we can state that with a large population size, it is highly likely to find the best-fitting model after a few generations. However, in reality, this does not necessarily reflect a better speed. Metaheuristics cannot be relied upon to find a global optimal solution, and it is recommended that the search be repeated more than once. A repeat search that balances

Table 3.3: Means and SDs of CPU time spent on finding the true structure when it is a Simple one

Mutation rate	Population size		
	5	10	20
0.01	583.61(250.61)	1135.70(428.70)	1590.73(721.70)
0.05	524.02(224.92)	1021.48(393.76)	1600.06(785.75)
0.10	539.53(234.95)	946.60(406.82)	1654.49(795.31)

Note. The measurement unit is seconds. The replications that failed to find the true structure are not included. SDs are given in parentheses.

speed and accuracy is therefore desirable. Finally, it is noteworthy that the effect of the mutation rate is complicated and difficult to interpret.

3.4 Example analysis 1: Analysis of the data from psychological tests

3.4.1 Method

In the previous section, the proposed algorithm is confirmed to work effectively in the case of artificial data. However, this does not necessarily imply that it also performs well for real data. I confirm this by using data with a well-known structure.

The data used here is obtained from four psychological tests in Murohashi & Toyoda (2004). This data contains items that measure four different psychological latent concepts: depression, anxiety, extroversion, and inferiority. Each item is carefully chosen to measure a relevant attribute based on the item response theory (Lord, 1980). In this paper, I revalidate the structure in Murohashi & Toyoda (2004) by using the GA.

The original data in Murohashi & Toyoda (2004) comprised four subtests: 37 items for depression, 54 items for anxiety, 43 items for extroversion, and 34 items for inferiority. All these items were rated using a three-point Likert scale. However, in this study, I chose 5 items that measure the corresponding latent trait from each subtest with high accuracy. Then, the selected 20 items are expected to have a four-factor structure. The details of the items are presented in Table 3.4. Items 1 to 5 pertain to depression; items 6 to 10, to anxiety; items 11 to 15, to extroversion; and items 16 to 20, to inferiority. The covariance matrix of these variables is shown in Table 3.5, and the factor pattern matrix derived from the exploratory factor analysis (EFA) with promax rotation applied to the data is shown in Table 3.6. Both the tables indicate that the 20 items have a relatively explicit four-factor structure. Hence, the GA is expected to find this structure.

The GA algorithm employed was described in the same manner as in the previous section; the chromosomes of the individuals in the initial population were generated based on the quadnomial distribution where each result is equally probable, the mask for the crossover was generated from the binomial distribution, which has a success rate of 0.50, and the number of trials was 20. The population size and the mutation rate were set to 10 and 0.05, respectively. With these parameters, I ran the GA 20 times and obtained 20 models. I determined these parameters arbitrarily and empirically based on a preliminary analysis.

Table 3.4: 20 items selected from Murohashi & Toyoda's (2004) psychological test data

Target Trait	Number	Item
Depression	1	I feel low and am depressed
	2	I get depressed
	3	I am pessimistic
	4	I suffer from fatigue
	5	I am feeling down
Anxiety	6	I am almost always obsessing about something/someone
	7	I am worried that a misfortune will happen
	8	I sometimes worry excessively about what is actually an insignificant thing
	9	I often feel tense or become nervous
	10	I often worry that something will go wrong
Extroversion	11	I can freely talk to anybody
	12	I do not make friends easily
	13	I like to associate widely with people
	14	I can easily talk to the people I meet for the first time
	15	I feel free to address people who are not so friendly
Inferiority	16	I am fraught with feelings of inferiority
	17	I am easily confused
	18	I am not very confident of everything
	19	I feel inferior to many people
	20	I have a very low opinion of myself

Table 3.5: Covariance matrix of four psychological tests data

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	v_{13}	v_{14}	v_{15}	v_{16}	v_{17}	v_{18}	v_{19}	v_{20}
v_1	0.499																			*sym*
v_2	0.322	0.475																		
v_3	0.274	0.294	0.569																	
v_4	0.245	0.251	0.223	0.467																
v_5	0.248	0.267	0.210	0.230	0.417															
v_6	0.262	0.261	0.315	0.217	0.194	0.739														
v_7	0.229	0.185	0.222	0.145	0.157	0.313	0.780													
v_8	0.231	0.244	0.251	0.194	0.167	0.351	0.345	0.787												
v_9	0.235	0.250	0.282	0.162	0.179	0.353	0.273	0.349	0.751											
v_{10}	0.266	0.270	0.301	0.228	0.196	0.422	0.382	0.363	0.326	0.730										
v_{11}	0.048	0.061	0.109	0.050	0.091	0.084	0.038	0.069	0.081	0.080	0.697									
v_{12}	0.087	0.097	0.129	0.088	0.128	0.128	0.079	0.083	0.077	0.123	0.333	0.674								
v_{13}	0.038	0.083	0.060	0.050	0.116	0.043	0.026	0.065	0.022	0.063	0.338	0.308	0.686							
v_{14}	0.063	0.070	0.128	0.056	0.106	0.056	0.071	0.088	0.066	0.088	0.492	0.353	0.341	0.699						
v_{15}	0.053	0.073	0.122	0.056	0.117	0.089	0.065	0.089	0.081	0.109	0.484	0.347	0.339	0.616	0.705					
v_{16}	0.256	0.239	0.312	0.212	0.198	0.327	0.258	0.306	0.282	0.350	0.143	0.197	0.117	0.150	0.198	0.777				
v_{17}	0.195	0.178	0.233	0.174	0.142	0.225	0.181	0.223	0.251	0.246	0.118	0.141	0.113	0.138	0.138	0.307	0.727			
v_{18}	0.232	0.224	0.267	0.187	0.187	0.254	0.222	0.238	0.235	0.312	0.095	0.167	0.136	0.144	0.153	0.398	0.289	0.689		
v_{19}	0.193	0.199	0.257	0.159	0.184	0.282	0.254	0.231	0.264	0.325	0.133	0.144	0.093	0.114	0.152	0.469	0.214	0.365	0.691	
v_{20}	0.178	0.179	0.234	0.133	0.176	0.220	0.122	0.109	0.176	0.188	0.119	0.168	0.097	0.108	0.135	0.294	0.180	0.300	0.304	0.612

Table 3.6: Factor pattern matrix of the EFA for four psychological tests data

	Factor1	Factor2	Factor3	Factor4
v_1	−0.048	0.654	0.033	0.216
v_2	−0.024	0.755	0.003	0.175
v_3	0.052	0.386	0.188	0.296
v_4	−0.026	0.570	0.047	0.139
v_5	0.072	0.659	0.083	0.028
v_6	−0.017	0.145	0.133	0.568
v_7	0.020	0.010	0.041	0.606
v_8	0.051	0.089	−0.016	0.634
v_9	0.001	0.139	0.096	0.507
v_{10}	0.014	0.096	0.159	0.631
v_{11}	−0.750	0.015	−0.016	−0.008
v_{12}	−0.501	−0.073	−0.156	0.020
v_{13}	−0.514	−0.088	−0.070	0.096
v_{14}	−0.970	0.008	0.085	−0.044
v_{15}	−0.927	0.053	−0.016	−0.044
v_{16}	0.023	0.018	0.640	0.249
v_{17}	0.088	0.115	0.250	0.239
v_{18}	0.020	0.114	0.550	0.163
v_{19}	−0.029	−0.054	0.718	0.207
v_{20}	0.002	0.174	0.560	−0.050

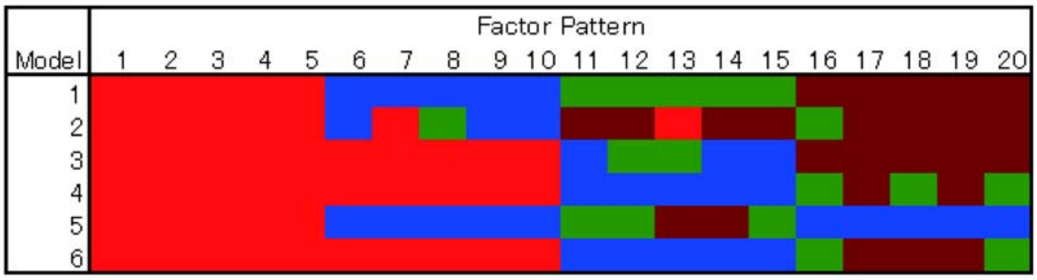
3.4.2 Results

Through 20 replications, 6 models were found by the proposed algorithm. Table 3.7 contains the fit indexes and the frequency of appearance of the models found, and Figure 3.4 contains the factor pattern of these models. Table 3.7 and Figure 3.4 present the same models. Therefore, model 1 in Table 3.7 is the same as model 1 in Figure 3.4. From these tables and figures, the model that was most frequently found by the GA had the four-factor structure, in which items from the same subtest contained one factor. Thus, the GA is considered to be capable of finding a good-fitting model even for real data.

Table 3.7: Fit indexes and frequency of appearance of the models found

Model	RMSEA	Number of times[a]
1	0.047	14
2	0.061	1
3	0.063	1
4	0.065	1
5	0.065	2
6	0.066	1

Note. [a] indicates the number of times the proposed algorithm found that model from 20 replications.



	Factor Pattern																			
Model	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	🟥	🟥	🟥	🟥	🟥	🟦	🟦	🟦	🟦	🟦	🟩	🟩	🟩	🟩	🟩	🟫	🟫	🟫	🟫	🟫
2	🟥	🟥	🟥	🟥	🟥	🟦	🟥	🟩	🟦	🟦	🟫	🟫	🟥	🟫	🟫	🟩	🟫	🟫	🟫	🟫
3	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟦	🟩	🟩	🟦	🟦	🟫	🟫	🟫	🟫	🟫
4	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟦	🟦	🟦	🟦	🟦	🟩	🟫	🟩	🟫	🟩
5	🟥	🟥	🟥	🟥	🟥	🟦	🟦	🟦	🟦	🟦	🟩	🟩	🟫	🟫	🟩	🟦	🟦	🟦	🟦	🟦
6	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟦	🟦	🟦	🟦	🟦	🟩	🟫	🟫	🟫	🟩

Figure 3.4: Factor Pattern of the models found

Note. Items with the same color in the factor pattern measure the same factor.

Of all the models other than model 1, models 3, 4, 5, and 6 have a similar structure. All of them merge two subtests into one factor and split another subtest into two factors. For example, model 3 treats depression and anxiety as single factor and divides extroversion into two. This is because the GA focused on the substructure of a certain subtest. The real data analysis confirms the importance of the fact that it is desirable to perform a repeat search.

However, the instability of the GA is not necessarily a disadvantage. When we want to explore the structure underlying the data, the variety of the models found helps us to consider the data. From this viewpoint, model 2 is worthy of attention. This model has a relatively unique structure as compared to the other models; depression still constitutes one factor, but the other factors are somewhat a combination of items from different subtests. This insight is consistent with the result of Murohashi & Toyoda (2004) that the four factors have relatively high correlations. It can be stated that it is possible for a model specification search using GA to find a variety of structures, but each result yields good-fit indexes.

3.5 Example analysis 2: Analysis of the data from a cognitive ability test

3.5.1 Method

In the analysis of the previous example, the proposed algorithm is confirmed to work effectively for real data with a well-known structure. Further, the availability of our proposed method's instability is indicated. Here, we analyze real data with a more complicated structure and examine the performance of the GA. This is an example that assumes the situation of real data analysis. Since it is not possible to know the real structure in natural data, we compared the results of the GA and the EFA.

In this study, we used the test results of Holzinger & Swineford (1939) as data. The data included the result of a cognitive ability test administered at Grant-White Elementary School; the number of examinees was 145. The test comprised 26 items. However, because items 3 and 25 and items 4 and 26 were designed to measure the same abilities with the only difference between them being related to the degree of difficulty, we used the easier items (items 25 and 26) and excluded the more difficult ones (items 3 and 4); thus, we used data for a total of 24 items in the analysis.

These items were chosen from five different areas: (a) spatial (items 1, 2, 25, and 26), (b) verbal (items 5 to 9), (c) speed (items 10 to 13), (d) memory (items 14 to 19), and (e) mathematical ability (items 20 to 24). The details of these items are presented in Table 3.8. After analyzing the collected data, Holzinger & Swineford (1939) concluded that the four-factor structure fits this data better. These data have been reanalyzed many times in the context of factor analysis (Carroll, 1993), and the number of factors employed by researchers has differed; however, in the present study, we employed the assumed four-factor structure model so as to follow the original study.

Table 3.8: Item contents of Holzinger & Swineford's (1939) test data

Area	Number	Content
Spatial	1	Visual perception
	2	Cubes
	25	Paper form board
	26	Flags
Verbal	5	General information
	6	Paragraph comprehension
	7	Sentence completion
	8	Word classification
	9	Word meaning
Speed	10	Add
	11	Code
	12	Counting groups of dots
	13	Straight and curved capitals
Memory	14	Word Recognition
	15	Number recognition
	16	Figure recognition
	17	Object-Number
	18	Number-Figure
	19	Figure-Word
Mathematical ability	20	Deduction
	21	Numerical puzzles
	22	Problem reasoning
	23	Series completion
	24	Woody-AcCall mixed fundamentals, Form I

The GA was employed in the same manner as described in the previous section; the chromosomes of the individuals in the initial population were generated based on the quadnomial distribution where each result is equally probable, the mask for the crossover was generated from the binomial distribution, which has a success rate of 0.50, and the number of trials was 24. The population size and the mutation rate were set to 10 and 0.05, respectively. With these parameters, we ran the GA 20 times and obtained 20 models. We decided these parameters arbitrarily and empirically based on a preliminary analysis.

At the same time, as a target for comparison, we carried out the EFA using the same data. First, we estimated the factor loadings using the maximum likelihood estimation and rotated it using three different rotation methods: promax, direct oblimin, and Harris-Kaiser. Then, based on the results, we assumed that each observed variable measures the factor with the highest factor loading and derived a simple structure from the EFA result. This is the standard procedure to derive a simple structure in exploratory data analysis.

3.5.2 Results

Table 3.9 contains the fit indexes and the frequency of appearance of the models found by the EFA and GA, and Figure 3.5 contains the factor pattern of these models. The most frequently found model by the GA was the one with the best RMSEA; it was found by promax rotation. Thus, for these data, both the GA and EFA were able to find the best-fitting model. Because of the arbitrariness of the parameter setting in the GA, it is possible to obtain different search results. Thus, there are advantages and disadvantages in using the GA; however, at the very least, the fact that the GA can perform as effectively as the EFA has been confirmed.

From model 1, the items selected from mathematical ability seem to be divided into other factors. This is the common feature of models 1, 3, and 13, which are chosen by the EFA. However, the factors that these items are classified into differ across models. Therefore, mathematical ability is considered as being a complex of many elements; it should therefore be treated as a more integrated trait than the other factors.

This is, however, not the only interpretation for this data because the model found by the GA has a somewhat different structure from the three EFA models, but each has an almost identical good-fit index. Models 2, 8, and 9 indicate the possibility that the items from memory are divided into other categories, as in the case of spatial ability. We can interpret these models such that the items for memory might measure the spatial memory, or the examinee can use some spatial strategy to solve these items. Moreover, models 4, 5, and 12 suggest that the items for memory can be combined with those for speed. Therefore, the speed and memory abilities are also correlated. The result of the GA indicates the diverse aspects of the items for memory that the EFA fails to show. This feature is the advantage of the proposed algorithm over the existing method.

Table 3.9: Fit indexes and frequency of appearance of the models found

Model	RMSEA	Searching Method	
		GA[a]	EFA[b]
1	0.06376	5	Promax
2	0.06379	4	-
3	0.06420	-	Harris-Kaiser
4	0.06518	2	-
5	0.06670	1	-
6	0.06792	1	-
7	0.06885	1	-
8	0.06890	1	-
9	0.06896	1	-
10	0.06931	1	-
11	0.07021	1	-
12	0.07025	1	-
13	0.07060	-	Direct Oblimin
14	0.07083	1	-

Note. The Dashes indicates the model for which a corresponding factor pattern was not found. [a] indicates the number of times the proposed algorithm found the factor pattern from 20 replications. [b] indicates that the corresponding factor pattern was obtained by the EFA using the rotation methods described earlier.

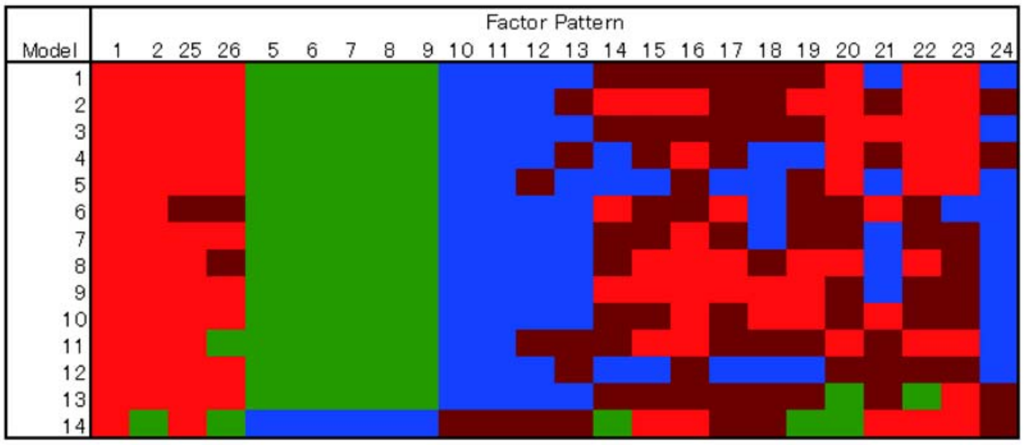


Figure 3.5: Factor pattern of the models found

Note. Items with the same color in the factor pattern measure the same factor.

Chapter 4

Model specification search using a genetic algorithm with factor reordering for a general factor analysis model

In the previous chapter, a search algorithm for a simple structure factor analysis model was proposed. We confined the search space to a simple structure in order to deal with some problems that disturb the search and decrease efficiency. However, as previously discussed, this restriction is too strong and impractical. Hence, we will propose an algorithm for searching a general factor analysis model in this chapter. A substantial portion of the algorithm proposed in this chapter is common to that described in the previous chapter. Therefore, we will only discuss the differences between the two.

4.1 Search algorithm for a general factor analysis model with factor reordering

4.1.1 Encoding method

First, the four assumptions described below are assumed.

1. Fix the number of observed variables and factors prior to starting

search.

2. Fix the factor's variance at one to identify the confirmatory factor analysis model.

3. Always estimate all the covariances among factors because it is highly unlikely that there are completely no correlations among factors.

4. Do not estimate the error covariance among observed variables.

The above assumptions are basically the same as those introduced in the previous section. The difference is that the search space is not limited to the simple structure. Hence, in order to search the model specification of the general factor analysis model, whole elements in the factor loading matrix $\boldsymbol{A}$ in Equation 3.1 should be the target of the search. Therefore, we can use the encoding algorithm by Marcoulides & Drezner (2001).

However, if we perform a search that covers all the possible patterns for the factor loadings, the GA will generate numerous individuals whose chromosomes represent an unidentifiable model and will therefore exert a harmful influence on the accuracy and efficacy of the search. To avoid this, we introduce the restriction for representing the exploratory factor analysis model in SEM on the factor loading matrix.

The variances of the factors have been already fixed at one by assumption 2. Hence, an additional assumption needs to be employed. It is as follows:

5. Every factor must be measured by at least one observed variable that only measures that corresponding factor.

If there are n observed variables and m factors, this assumption forces the m observed variables to measure only one factor. Hence, not all of the factor patterns are to be searched under this restriction. However, all the possible factor patterns for $n - m$ variables become identifiable. In addition, it is extremely rare that all the variables measure multiple factors, and the analyst usually tends to prefer a simple model, which, as described, is easy to interpret. The characteristic of this assumption that induces the search algorithm to find simpler model is therefore considered rather desirable.

Based on the above assumptions, the encoding method for the search of the general factor analysis model using the GA will be proposed as below:

- First, m genes of the chromosome represent the observed variables, which are constrained to measure only one factor. Each gene can have an integer value from 1 to m and the value of the Mth gene expresses the number of the observed variable, which measures the Mth factor. These m genes define m rows in the factor loading matrix $\boldsymbol{A}$.

- The remaining $n-m$ rows of $\boldsymbol{A}$ are encoded by the method proposed by Marcoulides & Drezner (2001). These elements are replaced by binary strings such that 0 represents the element to be fixed at zero and 1 represents the element to be freely estimated. Then, the elements are vectorized to become a character string.

Hence, the chromosome for this algorithm is composed of two different parts, and the total length will be $m + m(n-m)$.

If there are 6 observed variables and 2 factors, a chromosome will have 10 genes. One example of a chromosome would be

$$[3\ 5\ 1\ 1\ 0\ 1\ 1\ 0\ 1\ 1]. \tag{4.1}$$

The first two genes of this chromosome suggest that the third observed variable only measures the first factor and that the fifth observed variable only measures the second factor. The remainder of the genes can be reconstructed as a matrix

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}.$$

This is equal to the binary coded factor loading matrix $\boldsymbol{A}^*$, except for the rows for third and fifth observed variables, which are expressed in the first part. Therefore, the complete elements in $\boldsymbol{A}^*$ are as follows:

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}. \tag{4.2}$$

Finally, the chromosome in Equation 4.1 is decoded to express the factor analysis model illustrated in Figure 4.1. This is the encoding rule used for the algorithm dealing with the general factor analysis model in this study.

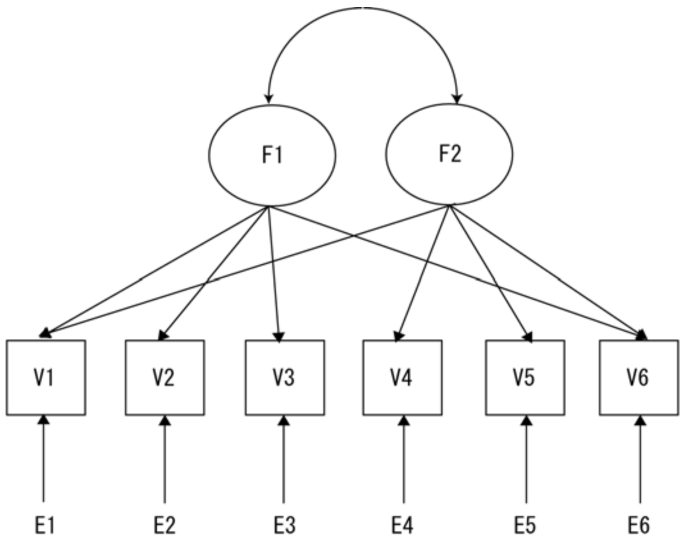


Figure 4.1: Factor analysis model represented by Equation 4.1

Moreover, the factor reordering operator was also introduced. The algorithm of this operator was as same as the one described in section 3.2.1. Factor reordering operator was executed everytime when new chromosome is generated in the algorithm.

4.1.2 Search cycle

Initialization process

The first part of the chromosome should be a character string consisting of integer numbers up to the number of observed variables. Hence, this part can be generated from a multinomial distribution that has an appropriate number of categories. In contrast, the second part of the chromosome is a binary string. Therefore, this part can be generated from the binomial distribution.

Reproduction process

Similar to the case of the algorithm used in the previous chapter, the

goodness of each chromosome was evaluated using the RMSEA. The method to select parents was also same, i.e., the roulette wheel selection algorithm. For the details, see the description given in section 3.2.2.
In contrast, the crossover algorithm was different. For the first part of the chromosome, the partially mapped crossover algorithm described in section 2.3.2 was used. This is because of the constraint that one variable can measure only one factor in this part of the chromosome; hence, the duplication of the gene value was not permitted. Using this algorithm would ensure that all the genes in the first part of the chromosome would always have different values. For the second part of the chromosome, we employed a one-point crossover algorithm, the cutting point of which is randomly determined each time a crossover operator is executed.
Finally, each gene of the children's chromosomes is checked individually based on a certain mutation rate, and if mutation occurs, the value of that gene would be changed. If mutation occurred in the first part of the chromosome, its value was changed to the integer value up to the number of observed variables except for the current value. For the second part of the chromosome, its value was changed to 1 if the current value was 0, and changed to 0 if the current value was 1.

Replacement process
The replacement process was completely identical to the algorithm used in chapter 3. Individuals who would to survive were selected based on their fitness and elitism. The number of elite individuals was one.

Repetetion process
The algorithm used for the repetition process was also identical to that used for the simple structure model.

4.1.3 Local search method

The local search method was also adopted in the algorithm for the general factor analysis model. The basic idea was similar to the method described in section 3.2.3 in that the 1-flip neighborhood of the existing model was to be searched. However, the 1-flip neighborhood includes entirely different individuals.
In this case, the 1-flip neighborhood was defined for the binary coded factor

loading matrix restored from a chromosome. For example, the chromosome of Equation 4.1 represents $\boldsymbol{A}^*$, which is indicated in Equation 4.2. In this case, the 1-flip neighborhood expressed in Equation 4.1 includes individuals with chromosomes generated by changing only one element of Equation 4.2. Therefore, a greater number of individuals were considered as part of the 1-flip neighborhood in the algorithm proposed here than the one in section 3.2.3.
However, as in the case of the algorithm for the simple structure model, not all of the 1-flip neighborhood individuals were evaluated. First, all the possible chromosomes in the 1-flip neighborhood were listed and their RMSEA were calculated in random order. Then, if a model with a fit better than that of the base model was found, the local search was terminated and the chromosome of that model found was adopted as the result of the local search.

4.2 Example analysis 1: Analysis of the data from a cognitive ability test

4.2.1 Method

It has been already confirmed that the GA can find a good-fitting model when it is applied to a model specification search in SEM. Hence, the algorithm proposed in this chapter is also basically expected to work well. However, the algorithm for the general factor analysis model is different from the one for the simple structure factor analysis model in terms of the impossibility of dealing with the model complexity and the equivalent model. Therefore, the problem lies in how this algorithm works for real data. In this section, we analyze the data of Holzinger & Swineford (1939) used in the previous chapter and examine the behavior of the proposed algorithm by comparing their results.

The items employed are shown in Table 3.8, and as in the previous analysis, we assumed four factors. The GA was employed in the same manner as described in section 4.1; the first part of the chromosomes for individuals in the initial population was generated based on the multinomial distribution with 24 categories, and the second part was generated based on the binomial distribution. In both of these distributions, each result is assumed to be equally probable. The population size and the mutation rate were set to 50 and 0.05, respectively. With these parameters, the search was completed after 50 iterations of the search cycle.

4.2.2 Results

Figure 4.2 depicts the simple structure factor pattern obtained in section 3.5 and the factor patterns derived by the proposed algorithm for the general factor analysis model. Here, the three good-fitting models are indicated; the RMSEA of models 1, 2 and 3 were 0.045, 0.047, and 0.049, respectively. All these models have better RMSEA values than the simple structure model with an RMSEA of 0.064.

This result indicates that the proposed algorithm for the general factor analysis model found a structure similar to the model obtained from the algorithm for the simple factor analysis model. Items for spatial, verbal, speed and memory constitute corresponding factors and items for mathematical ability are devided into those four factors. This suggests that the algorithm for the general factor model works in the same manner as that for the simple structure model.

Moreover, by comparing multiple models, the proposed algorithm can provide more information than the algorithm for the simple structure model. This is because the result of the proposed algorithm contains information about the whole factor loading matrix. Of the three models indicated, the structures of the factors for the spatial and verbal abilities are considerably alike. Therefore, these factors are considered to have a robust structure. In contrast, speed and memory seem to be unstable. Memory may be divided into two factors: one is the general factor and the other is the residual or specific factor. Speed appears to have a more complicated structure because of the items for mathematical ability. In this manner, we can extract some information from the elements which have relatively low value of factor loading.

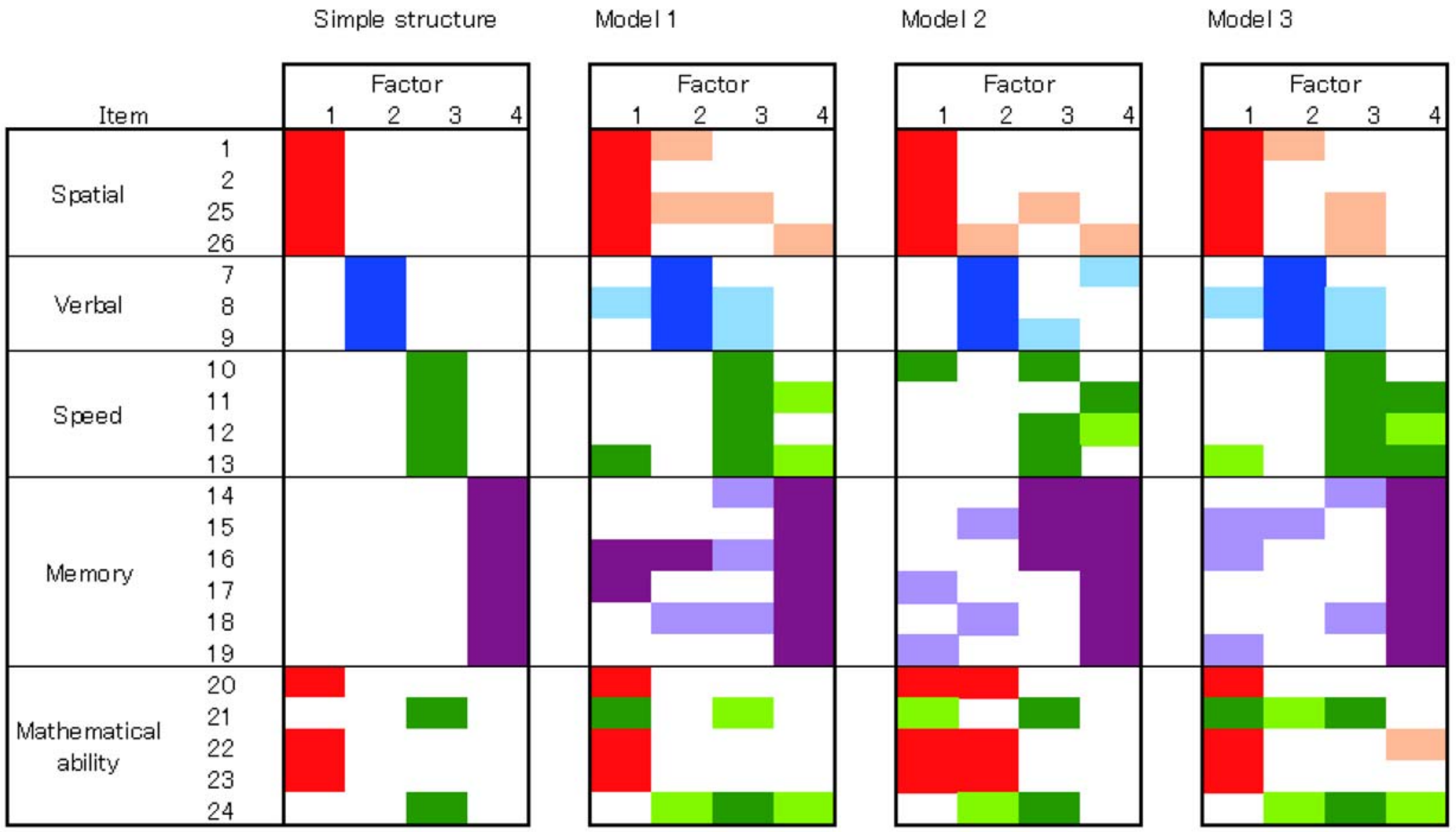


Figure 4.2: Comparison of the factor pattern obtained from the GA with the simple structure factor pattern

Note. The colored cells indicate an elements to be freely estimated in the factor loading matrix; the deep colors represent the factor loading with absolute values greater than 0.4, and the light colors represent the factor loading with absolute values greater than 0.2 and less than 0.4.

4.3 Example analysis 2: Analysis of the data from an intelligence test

4.3.1 Method

It is conirmed that the proposed algorithm for the general factor analysis model can find a good-fitting model as same as the algorithm for the simple factor analysis model in the analysis of the previous example. Here, we examine whether the algorithm for the general factor analysis model can find a huge variety of models including the models which are difficult to be found by the EFA. In the first chapter, we described Thurstone's multiple-factor theory of intelligence. In this section, his data from Thurstone & Thurstone (1941) will be reanalyzed. Thurstone proposed the following seven factors of intelligence: (a) perceptual speed, (b) associative memory, (c) verbal comprehension, (d) word fluency, (e) spatial visualization, (f) number facility, and (g) reasoning. This structure was confirmed by an exploratory factor analysis of 21 items, which are listed in Table 4.1.

First, as a target for comparison, we performed an EFA with the data. We estimated factor loadings by using maximum likelihood estimation and rotated it using promax rotation. Then, we employed the GA to analyze this data with a manner identical to that described in section 4.1; the first part of the chromosomes for individuals in the initial population was generated based on the multinomial distribution with 21 categories, and the second part was generated based on the binomial distribution. In both the distributions, each result is assumed to be equally probable. The population size and the mutation rate were set to 50 and 0.05, respectively. In addition, by considering that each observed variable measures the factor for which the absolute value of the factor loading is greater than 0.4, we add a model obtained through the EFA to the initial population through the seeding method. With the above settings, the search was completed after 50 iterations of the search cycle.

Table 4.1: Items from Thurstone & Thurstone's (1941) data

Factor	Number	Item
Perceptual speed	1	Identical numbers
	2	Faces
	3	Mirror reading
Associative memory	4	First names
	5	Figure recognition
	6	Word-number
Verbal comprehension	7	Sentences
	8	Vocabulary
	9	Completion
Word fluency	10	First-letters
	11	Four-letter words
	12	Suffixes
Spatial visualization	13	Flags
	14	Figures
	15	Cards
Number facility	16	Addition
	17	Multiplication
	18	Three-highter
Reasoning	19	Letter series
	20	Pedigrees
	21	Letter grouping

4.3.2 Results

Figure 4.3 presents the EFA-based factor pattern used for seeding and the factor patterns derived by the proposed algorithm for the general factor analysis model. The figure shows the best-fitting model with RMSEA= 0.090 and the model with a unique structure with RMSEA= 0.118.

Both the simple structure model and the best-fitting model obtained by the GA have clear seven-factor structures. Therefore, the proposed algorithm is confirmed to have the ability to find a good-fitting factor pattern from data. In addition, as discussed a number of times, the GA can extract additional information from data. The unique model presented in Figure 4.3 is one with a slightly different factor pattern obtained by the GA. This model ignores the factor for associative memory due to which its fit is not very good. Therefore, we cannot employ this model as the final result. However, it suggests the possibility that item 12 for word fluency and item 19 for reasoning are also correlated with perceptual speed. The perceptual speed may constitute the foundation of intellectual activity and should be treated as more general component than other factors. This another viewpoint for the structure of intelligence is hard to be derived by the EFA, but the proposed algorithm can provide this information.

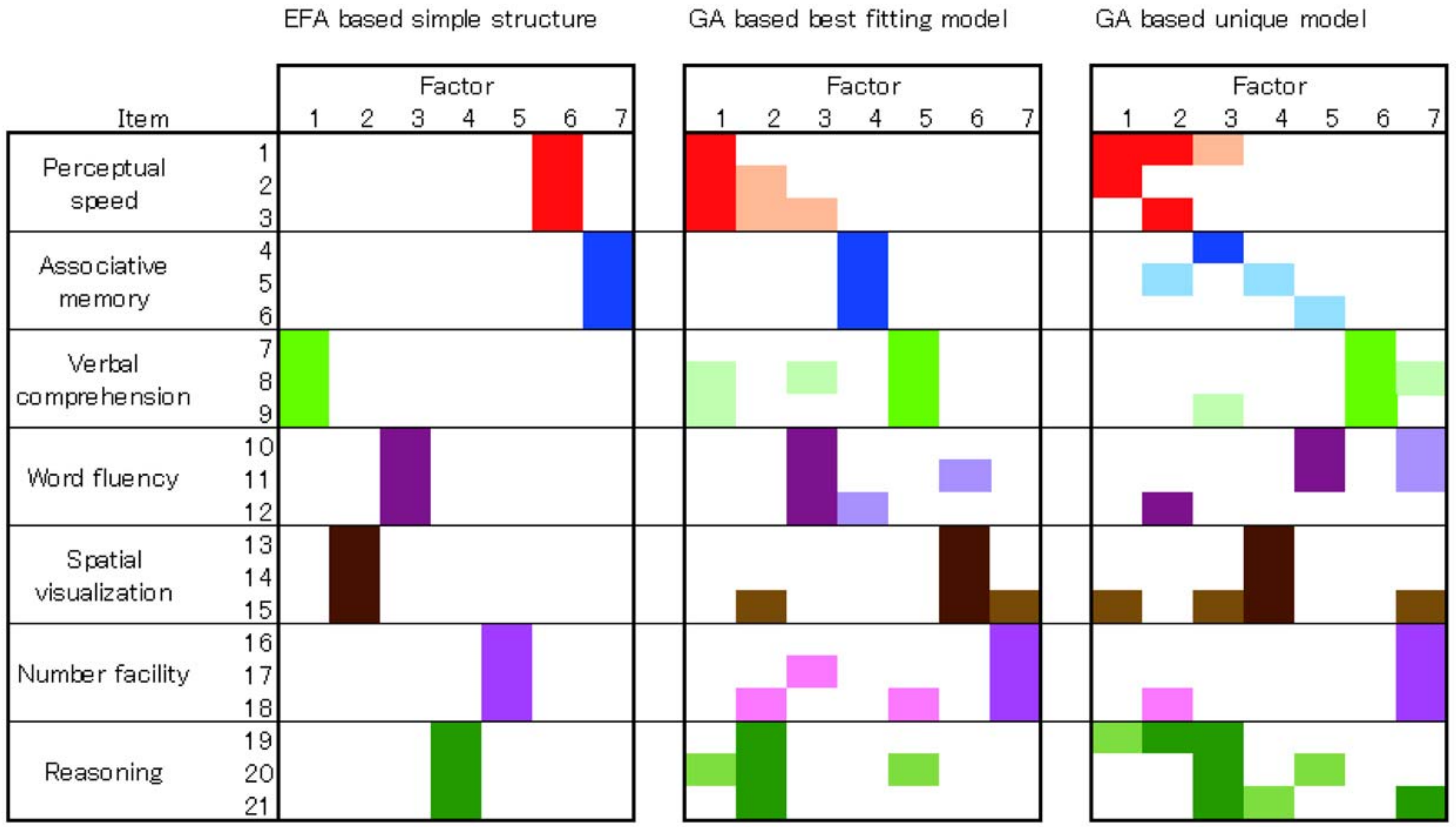


Figure 4.3: Comparison of the factor pattern obtained from the GA with the simple structure factor pattern

Note. The colored cells indicate an elements to be freely estimated in the factor loading matrix; the deep colors represent the factor loading with absolute values greater than 0.4, and the light colors represent the factor loading with absolute values greater than 0.2 and less than 0.4.

4.4 Example analysis 3: Analysis of the data from a personality test

4.4.1 Method

Finally, we analyze real data with a more complicated structure and examine the performance of the proposed algorithm. The data is from a personality test based on the big-five theory. Recently, the big-five theory was proposed as a powerful framework to perceive the human personality (Wiggins, 1996). The origin of the big-five is that the five-factor structure has often been found through the lexical approach in which words representing personality are collected and classified (Tupes & Christal, 1961; Norman, 1963). Goldberg (1990, 1993) focused on these previous studies and developed the framework of the big-five theory.

Meanwhile, the big-five theory has received severe criticism. One major criticism is that the human personality does not have a five-factor structure. For example, Eysenck (1990) advocates the three-factor model that personality can be understood in terms of extraversion, neuroticism, and psychoticism. Cattell (1990) puts forth a 16-factor model. Another criticism is that the content of the big-five varies across studies. The five factors proposed by Goldberg (1993) were surgency, agreeableness, conscientiousness, emotional stability, and culture. However, Peabody & Goldberg (1989) interpreted them as power, love, work, affect, and intellect. The revised NEO personality inventory (NEO-PI-R) is one of the most widely used inventories to measure personality based on the big-five model (Costa Jr. & McCrae, 1992). In this questionnaire, the five factors of personality are named: neuroticism, extraversion, openness, agreeableness, and conscientiousness.

For the reasons described above, the big-five theory continues to be controversial. However, it is certain that the five-factor structure is often observed in personality assessment data (McCrae & Costa Jr., 1997) irrespective of its detailed contents. Therefore, assuming the existence of the five factors constitutes a general theoretical framework to understand the human personality.

In this section, we reanalyzed the result of the NEO-PI-R conducted in Timothy & Burke (1994) and examined the structure of personality by using the GA. NEO-PI-R comprises 20 items as shown in Table 4.2. Using the GA, a search was performed assuming the five factors. The algorithm was identical to that described in section 4.1; the first part of the chromosomes for individuals in the initial population were generated based on the multinomial distribution with 20 categories, and the second part was generated based on the binomial distribution. In both the distributions, each result is assumed to be equally probable. The population size and the mutation rate were set to 50 and 0.05, respectively. In addition, we added a model obtained from EFA by considering that each observed variable measures the factor which the absolute value of the factor loading is greater than 0.4 to the initial population as seeding. With these settings, the search was completed after 50 iterations of the search cycle.

Table 4.2: Items from Church & Burke's (1994) data

Factor	Number	Item
Neuroticism	1	Anxiety
	2	Hostility
	3	Depression
	4	Self-consciousness
	5	Impulsiveness
	6	Vulnerability
Extraversion	7	Warmth
	8	Gregariousness
	9	Assertiveness
	10	Activity
	11	Excitement-seeking
	12	Positive emotions
Openness	13	Fantasy
	14	Aesthetics
	15	Feelings
	16	Actions
	17	Ideas
	18	Values
Agreeableness	19	Agreeableness
Conscientiousness	20	Conscientiousness

4.4.2 Results

Figure 4.4 illustrates the EFA-based factor pattern used for seeding and the factor patterns derived by the proposed algorithm for the general factor analysis model. Here, models 1 to 3 are the three good-fitting models obtained by the GA. The RMSEA of models 1, 2, and 3 were 0.087, 0.089, and 0.093, respectively. In addition, an additional model with a unique factor pattern and a relatively good fitness is shown. The RMSEA of this unique model was 0.100.

In the EFA-based model, the five-factor structure is slightly unclear. In particular, agreeableness and conscientiousness are measured by several unexpected items. However, we can extract more detailed information from the data with the proposed algorithm. From models 1 to 3 obtained by the GA for general factor analysis model, even extraversion and openness appeared to be unstable. These items should be reinterpreted by considering the items for agreeableness and conscientiousness. In particular, openness can be devided into two components.

The unique model provides a richer interpretation of this data. Neuroticism is the only factor that is relatively well organized in the EFA-based and the good-fitting models. However, the unique model attempts to expand neuroticism and conscientiousness and results in a unique factor pattern. This model is useful from the viewpoint of psychopathology because it focuses on neuroticism. The big-five theory is still uncertain, and its constituent factors are not determined. The GA for the general factor analysis model was able to propose a new viewpoint to this problem.

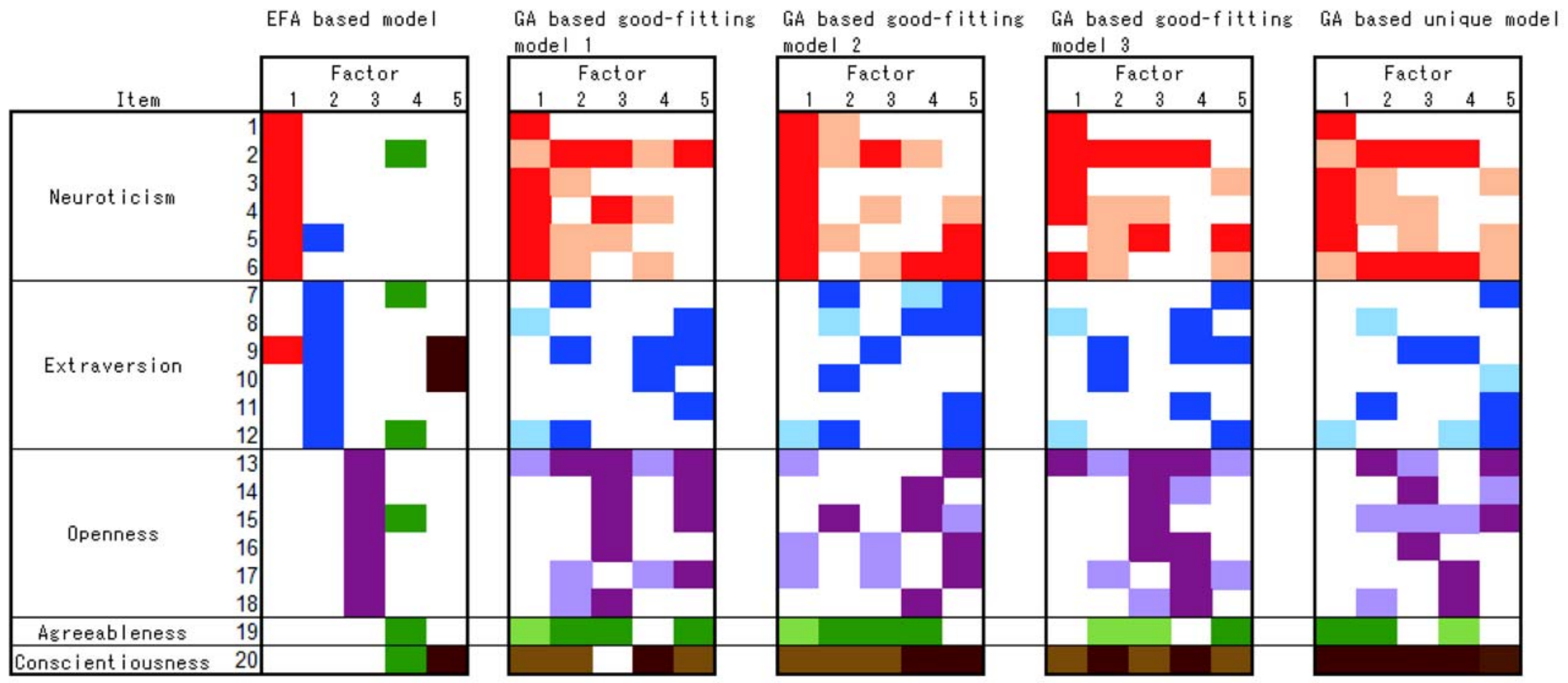


Figure 4.4: Comparison of the factor pattern obtained from the GA with the EFA-based factor pattern

Note. The colored cells indicate an elements to be freely estimated in the factor loading matrix; the deep colors represent the factor loading with absolute values greater than 0.4, and the light colors represent the factor loading with absolute values greater than 0.2 and less than 0.4.

Chapter 5

Conclusion

Through the simulation study and the real data analysis, the paper demonstrates the performance of the model specification search using the GA. The algorithm we propose here is able to find the true model when analyzing artificial data in certain a probability and within a reasonable time. It is also able to find a good-fitting model when analyzing real data. Thus, we can state that a model specification search using the GA has sufficient applicability.

Because the GA is a metaheuristic searching method, it cannot always find the best model. This feature is evident in the simulation study results. Consequently, it is highly recommended that researchers repeat model specification search more than two times and choose their own best model not only by considering their fit indexes but also by their interpretability and consistency with each field's theory. This instability of search result is usually treated as a disadvantage of metaheuristics; however not in this case. Through example analyses, it was confirmed that the proposed algorithm can find a great variety of models that are difficult to find using the EFA because of this instability. Threfore, this feature of our algorithm will contribute to model specification search where researchers do not have an explicit hypothesis and want to construct data-driven models.

However, we should take care with regard to the parameters of the GA. Population size is particularly related to the speed and accuracy of the search. Researchers must therefore set a sufficiently large population size for their data; however, an excessively large population size may slacken the search. Therefore, we should determine the appropriate population size without relying on a simple rule. It is probable that some studies that attempted to adjust parameters automatically in the estimation process itself may be helpful in tackling this issue (e.g., Beasley (1993); Nonobe & Ibaraki (1998)).

We recognize the need to improve the speed and accuracy of our algorithm; hence, we can approach these problems primarily from two different perspectives. One is to apply the latest advances in metaheuristics. Metaheuristics is an ongoing research area, and many types of algorithms are currently being proposed. These will prove to be helpful in improving our algorithm. To parallelize search process or combining more powerful local search method are deemed to be effective. The other approach is to rewrite and optimize the program code. In this study, we used the R system. R has high flexibility and is easy to use; however, it is a script language, and the processing speed is therefore not very high. Using a high-level language certainly accelerates the search.

A final consideration to be noted is that this study handled model specification search only for a factor analysis model. Nevertheless, the GA was found to be helpful for model specification search for a factor analysis model. Therefore, the application of metaheuristics for model specification search in general SEM appears to be effective, powerful, and easy to apply. For these reasons, this topic merits further research.

Appendix A

R code of the algorithm for a simple structure factor analysis model

```
# scan の出力を抑制するバージョン
specify.model2 <- function (file = "")
{
    ram <- scan(file = file, what = list(path = "", par = "",
        start = 1, dump = ""), sep = ",", strip.white = TRUE,
        comment.char = "#", fill = TRUE, quiet = TRUE)
    ram <- cbind(ram$path, ram$par, ram$start)
    class(ram) <- "mod"
    ram
}

# 推定結果が「正しいモデル」の範疇にあるかどうかを判定
final.check <- function(true.pattern, estimated.pattern){
  nof <- ncol(true.pattern)
  nov <- nrow(true.pattern)
  est.list <- true.list <- as.list( rep(NA, nof) )
  for(i in 1:nof){
    true.list[[i]] <- which(true.pattern[,i]==1)
    est.list[[i]] <- which(estimated.pattern[,i]==1)
  }
  check.result <- matrix(NA, nrow=nof, ncol=nof)
  for(i in 1:nof){
    for(j in 1:nof){
      check.result[j,i] <- all( est.list[[i]] %in% true.list[[j]] )
    }
  }
  result <- all( apply( check.result, 2, any ) )
  return(result)
}

############################################################
##  因子の分散共分散行列を生成する関数                      ##
##  gen.fact.cov(n.fac, corr.range)                         ##
##       n.fac <- 因子数 (整数)                              ##
##       corr.range <- 因子間相関の範囲                     ##
##                     (サイズ 2 のベクトル, c(下限, 上限))  ##
##    因子間相関は一様分布によりランダム発生               ##
##    因子の分散は 1.0 に固定                               ##
############################################################
gen.fact.cov <- function(n.fac, corr.range){
  fact.cov.mat <- diag(1, nrow=n.fac)
  fact.cov.mat[ lower.tri(fact.cov.mat, diag=F) ] <- runif( choose(n.fac, 2),
   min=corr.range[1], max=corr.range[2] )
  fact.cov.mat[ upper.tri(fact.cov.mat, diag=F) ] <- fact.cov.mat[ lower.tri(fact.cov.mat,
   diag=F) ]
  return(fact.cov.mat)
}

############################################################
##  誤差の分散共分散行列を生成する関数                     ##
##  gen.err.cov(n.var, var.range)                           ##
##       n.var <- 観測変数の数 (整数)                       ##
##       var.range <- 誤差分散の範囲                        ##
##                    (サイズ 2 のベクトル, c(下限, 上限))   ##
##    誤差分散は一様分布によりランダム発生                 ##
##    誤差間相関は 0.0 に固定                               ##
############################################################
gen.err.cov <- function(n.var, var.range){
  err.cov.mat <- diag( runif(n.var, min=var.range[1], max=var.range[2]), nrow=n.var )
  return(err.cov.mat)
}

###########################################################################
##  因子負荷行列を生成する関数                                          ##
##  gen.fact.load(n.fac, n.var, load.range, overlap)                      ##
##       n.fac <- 因子数 (整数)                                           ##
##       n.var <- 観測変数の数 (整数)                                     ##
##       load.range <- 因子負荷の範囲                                     ##
##                     (サイズ 2 のベクトル, c(下限, 上限))                ##
##       overlap <- 単純構造から, さらに何本のパスを増やすか (整数)     ##
##    因子負荷は一様分布によりランダム発生                              ##
##    overlap=0 の時は, 単純構造を発生                                    ##
##    overlap>0 の時は, 単純構造を起点にして,                            ##
##            同じ因子を測定する観測変数のブロックごとに,              ##
##            overlap 本のパスを追加する                                  ##
###########################################################################
gen.fact.load <- function(n.fac, n.var, load.range, overlap){
  fact.load.mat <- matrix(0, nrow=n.var, ncol=n.fac)
  simple.patr.flag <- cbind( c(1:n.var), cut( sample(1:n.var), breaks=n.fac, labels=F) )
  fact.load.mat[simple.patr.flag] <- runif( n.var, min=load.range[1], max=load.range[2] )
  ifelse( overlap==0, return(fact.load.mat), NA )
  for(i in 1:n.fac) fact.load.mat[,i][ sample( which(fact.load.mat[,i]==0),
   overlap ) ] <- runif( overlap, min=load.range[1], max=load.range[2] )
  return(fact.load.mat)
}

##################################################################################################
# 因子分析モデルに基づくデータを発生させる関数                                                 #
# gen.sample.factor(n.fac, n.var.each, fact.corr.range, err.var.range, fact.load.range, overlap, obs)     #
#      n.fac <- 因子数 (整数)     n.var.each <- 因子ごとの観測変数の数 (整数)                           #
#      fact.corr.range <- 因子間相関の範囲 (サイズ 2 のベクトル, c(下限, 上限))                      #
#      err.var.range <- 誤差分散の範囲 (サイズ 2 のベクトル, c(下限, 上限))                          #
#      fact.load.range <- 因子負荷の範囲 (サイズ 2 のベクトル, c(下限, 上限))                        #
#      overlap <- 単純構造から, さらに何本のパスを増やすか (整数)                                    #
#      obs <- サンプル数 (整数)                                                                          #
# 出力リスト: true.alpha -- 因子負荷行列の真値     true.pattern -- 因子パタンの 0/1 フラグ (0 がパス
有り)      #
#             true.sigma.f -- 因子間共分散行列の真値     true.sigma.e -- 誤差間共分散行列の真値          #
#             generated.data -- 発生されたデータ     generated.cov -- 発生されたデータの共分散行
列         #
###############################################################################################################
gen.sample.factor <- function(n.fac, n.var.each,
 fact.corr.range, err.var.range, fact.load.range, overlap, obs){
  sig.f <- gen.fact.cov( n.fac, fact.corr.range )
  sig.e <- gen.err.cov( n.fac*n.var.each, err.var.range )
  alpha <- gen.fact.load( n.fac, n.fac*n.var.each, fact.load.range, overlap )
  alpha.flag <- matrix(0, nrow=n.fac*n.var.each, ncol=n.fac)
  alpha.flag[ which( alpha>0 ) ] <- 1
  true.cov <- alpha %*% sig.f %*% t(alpha) + sig.e
  data <- mvrnorm( n=obs, rep( 0, n.fac*n.var.each ), true.cov )
  colnames(data) <- paste("v", 1:(n.fac*n.var.each), sep="")
  gen.cov <- cov(data)
  out <- list(true.alpha=alpha, true.pattern=alpha.flag, true.sigma.f=sig.f,
   true.sigma.e=sig.e, generated.data=data, generated.cov=gen.cov)
  return(out)
}

####################################################
##  因子数が規定通りかどうかチェックする関数      ##
##  check.dna(dna, nfac)                          ##
##       dna <- 1 個体分の染色体 (1 × n の行列)       ##
##       n.fac <- 正しい因子数 (整数)              ##
##    返り値は TRUE/FALSE                          ##
####################################################
check.dna <- function(dna, n.fac){
  ifelse( length( unique( c(dna) ) )==n.fac, return(TRUE), return(FALSE) )
}

####################################################
##  因子の並び替えを行う関数                      ##
##  reorder.dna(dna)                              ##
##       dna <- 1 個体分の染色体 (1 × n の行列)       ##
##    返り値も 1 × n の行列                         ##
####################################################
reorder.dna <- function(dna){
  nof <- max(dna)
  nov <- length(dna)
  flag.new <- flag.old <- cbind( 1:nov, t(dna) )
  temp.work <- matrix( c( 1:nof, rep(0,nof) ), nrow=nof, ncol=2 )
  for(i in 1:nof) temp.work[i,2] <- min( flag.old[which(flag.old[,2]==i), , drop=F][,1] )
  temp.work <- temp.work[ order(temp.work[,2]), ]
  temp.work <- cbind(1:nof, temp.work)
  for(i in 1:nof) flag.new[,2][which(flag.old[,2]==temp.work[i,2])] <- temp.work[i,1]
  dna.new <- matrix( flag.new[,2], nrow=1, ncol=nov)
  return(dna.new)
}
```

```
####################################################
##  遺伝子の発生を行う関数                      ##
##  generate.dna(n.fac, n.var)                    ##
##       n.fac <- 因子数 (整数)                   ##
##       n.var <- 観測変数の数 (整数)             ##
##    返り値は 1 × n の行列                       ##
####################################################
generate.dna <- function(n.fac, n.var){
  dna.proposed <- matrix( 0, nrow=1, ncol=n.var )
  while( check.dna( dna.proposed, n.fac )==F ) dna.proposed <- c(1:n.fac) %*%
   rmultinom( n.var, size=1, prob=rep(1/n.fac, n.fac) )
  dna.proposed <- reorder.dna(dna.proposed)
  return(dna.proposed)
}

###########################################################
##  既存個体との遺伝子の重複をチェックする関数         ##
##  check.duplication(i, group)                          ##
##       i <- 集団中の何番目の個体について調べるか (整数)  ##
##       group <- 調べたい集団 (行列)                    ##
##   重複が有れば TRUE, 無ければ FALSE                   ##
###########################################################
check.duplication <- function(i,group,nof,nov){
  for(j in 1:(i-1)){
    ifelse( all(group[j,1:nov]-group[i,1:nov]==0),
            return(TRUE), NA)
  }
  return(FALSE)
}

##########################################################
##  遺伝子からモデル指定文を生成する関数               ##
##  decode.dna(dna)                                     ##
##       dna <- 1 個体分の染色体 (1 × n の行列)         ##
##    ワークディレクトリに specification_temp.txt を生成   ##
##########################################################
decode.dna <- function(dna){
  nof <- max(dna)
  nov <- length(dna)
  factor.loading <- paste( "F", dna[1:nov], " ", "->", " ", "V", 1:nov,
   ", alpha_", 1:nov, dna[1:nov], ", NA", sep="")
  error.observed <- paste( "V", 1:nov, " <-> ", "V", 1:nov, ", delta_",
   1:nov, ", NA", sep="")
  factor.var <- paste( "F", 1:nof, " <-> ", "F", 1:nof, ", NA", ", ", 1, sep="")
  dummy.count2 <- which( lower.tri(diag(nof)), arr.ind=T )
  dummy.count <- matrix(0, nrow=((nof^2-nof)/2), ncol=2)
  dummy.count[,1] <- dummy.count2[,2]
  dummy.count[,2] <- dummy.count2[,1]
  factor.corr <- paste( "F", dummy.count[1:((nof^2-nof)/2),1], " <-> ",
   "F", dummy.count[1:((nof^2-nof)/2),2], ", corr_", dummy.count[1:((nof^2-nof)/2),1],
    dummy.count[1:((nof^2-nof)/2),2], ", NA", sep="")
  specification <- matrix(c(factor.loading, error.observed,
   factor.var, factor.corr), nrow=nov*2+nof+((nof^2-nof)/2), ncol=1)
  write(specification, "specification_temp.txt")
}
```

```
# 因子パタン行列からのモデル指定文の生成
decode.pattern <- function(pattern){
  nof <- ncol(pattern)
  nov <- nrow(pattern)
  factor.loading <- rep(0, nof*nov)
  for(i in 1:(nof*nov)) ifelse(pattern[i]==1,
   factor.loading[i] <- paste("f", (i-1)%/%nov+1, " -> v",
   ifelse(i%%nov==0, nov, i%%nov), ", alpha_",
    (i-1)%/%nov+1, ifelse(i%%nov==0, nov, i%%nov), ", NA", sep="" ), NA)
  factor.loading <- factor.loading[which(factor.loading!="0")]
  error.observed <- paste( "v", 1:nov, " <-> ", "v", 1:nov, ", delta_", 1:nov, ", NA", sep="")
  factor.var <- paste( "f", 1:nof, " <-> ", "f", 1:nof, ", NA", ", ", 1, sep="")
  dummy.count <- which( upper.tri(diag(nof)), arr.ind=T )
  factor.corr <- paste( "f", dummy.count[1:nof,1], " <-> ", "f", dummy.count[1:nof,2], ", corr_",
   dummy.count[1:nof,1], dummy.count[1:nof,2], ", NA", sep="")
  specification <- matrix(c(factor.loading, error.observed, factor.var, factor.corr), ncol=1)
  write(specification, "specification_temp.txt")
}

############################################################
##  分析を行い RMSEA を求める関数                          ##
##  calc.fit(dna)                                          ##
##       dna <- 1 個体分の染色体 (1 × n の行列)             ##
##     推定が収束しなかった場合は 99 を返す                ##
############################################################
calc.fit <- function(dna, data, nos){
  decode.dna(dna)
  temp.model <- specify.model2(file="specification_temp.txt")
  est.result <- try(sem(temp.model, data, nos))
  rmsea <- 999
  ifelse( est.result$convergence==1, rmsea <- sqrt( max( est.result$criterion / ( 0.5 * est.result$n *
   ( est.result$n + 1 ) - est.result$t ) - 1 / (est.result$N - 1), 0) ), rmsea <- 99)
  return(rmsea)
}

# 因子パタン行列からの適合度の計算
calc.fit2 <- function(pattern, data, nos){
  decode.pattern(pattern)
  temp.model <- specify.model2(file="specification_temp.txt")
  est.result <- try(sem(temp.model, data, nos))
  rmsea <- 999
  ifelse( est.result$convergence==1, rmsea <- sqrt( max( est.result$criterion / ( 0.5 * est.result$n *
   ( est.result$n + 1 ) - est.result$t ) - 1 / (est.result$N - 1), 0) ), rmsea <- 99)
  return(rmsea)
}


##################################################################
##  子供の発生を行う関数                                        ##
##  reproduction(group, mutate.prob, population, nof, nov)     ##
##       group <- 集団 (行列)                                   ##
##       mutate.prob <- 突然変異確率 (整数)                     ##
##       population <- 個体数 (整数)                            ##
##       nof <- 因子数 (整数)                                   ##
##       nov <- 観測変数の数 (整数)                             ##
##     子供 2 体分の遺伝子を 2 × n の行列で返す                 ##
##################################################################
```

```
reproduction <- function(group, mutate.prob, population, nof, nov){
  # 選択
  parents.select <- c(1:population) %*% rmultinom( 2, size=1, prob=replace(1/group[,(nov+1)],
   is.infinite(1/group[,(nov+1)]), 1e-99) )
  while( parents.select[1]==parents.select[2] ) parents.select <- c(1:population) %*% rmultinom( 2,
   size=1, prob=replace(1/group[,(nov+1)], is.infinite(1/group[,(nov+1)]), 1e-99) )
  # 交叉
  mask <- matrix( rbinom( nov, 1, 0.5), nrow=1)
  while( length( unique( c(mask) ) )==1 ) mask <- matrix( rbinom( nov, 1, 0.5), nrow=1)
  child.dna <- matrix( 0, nrow=2, ncol=nov )
  child.dna[1,] <- (mask * group[ parents.select[1], 1:nov, drop=F])+(-(mask-1) * group[ parents.select[2],
   1:nov, drop=F])
  child.dna[2,] <- (mask * group[ parents.select[2], 1:nov, drop=F])+(-(mask-1) * group[ parents.select[1],
   1:nov, drop=F])
  # 突然変異
  mutation.flag <- matrix( rbinom( 2*nov, 1, mutate.prob), nrow=2)
  if( sum(mutation.flag)>0) for(i in 1:sum(mutation.flag))
   child.dna[which(mutation.flag==1)[i]] <- setdiff( 1:nof,
    child.dna[which(mutation.flag==1)[i]]) %*% rmultinom( 1, size=1, prob=rep( 1/(nof-1), (nof-1) ) )
  return(child.dna)
}

# メインルーチン
ga.search <- function(){
# シミュレーションのパラメータ指定
nof <- 4
nov <- 24
nos <- 145

# GA のパラメータ設定
population <- 10
mutate.prob <- 0.05
criterion.fit <- 0.0
criterion.generation <- 50
criterion.change <- 0.0001
criterion.stop <- 3

# H&S データの読み込み
inp <- read.table("HandSdata.dat")
data.cov <- as.matrix(inp)
rownames(data.cov) <- colnames(data.cov)

# 初期個体群の生成
group <- matrix( c( rep(0, population*nov),rep(99, population) ), nrow=population, ncol=nov+1 )
group[1,1:nov] <- generate.dna(nof,nov)
group[1, nov+1] <- calc.fit( group[1,1:nov,drop=F], data.cov, nos )
for(i in 2:population){
  group[i,1:nov] <- generate.dna(nof,nov)
  while( check.duplication(i,group,nof,nov)==TRUE ) group[i,1:nov] <- generate.dna(nof,nov)
  group[i, nov+1] <- calc.fit( group[i,1:nov,drop=F], data.cov, nos )
}
group <- group[ order( group[,nov+1] ), ]
replace( group, is.na(group), 999 )
replace( group, is.null(group), 999 )
generation <- 1
counter <- 1
break.flag <- 0
```

```
evolution.histry <- as.list(NA)
evolution.histry[[generation]] <- list("Generation"=generation, "Best DNA"=group[1,1:nov],
 "Fit"=group[1,nov+1], "Population Detail"=group )

# 遺伝的アルゴリズムループ
while(break.flag==0){
# 生殖
offspring <- matrix( c( rep(0, population*nov),rep(99, population) ), nrow=population, ncol=nov+1 )
counter.child <- 1
repeat{
  children.proposed <- reproduction(group, mutate.prob, population, nof, nov)
  ifelse( check.dna(children.proposed[1, , drop=F],nof)==T,
         {offspring[counter.child, 1:nov] <- reorder.dna( children.proposed[1,
          , drop=F] ); counter.child <- counter.child+1},
         NA)
  ifelse(counter.child>population, break, NA)
  ifelse( check.dna(children.proposed[2, , drop=F],nof)==T,
         {offspring[counter.child, 1:nov] <- reorder.dna( children.proposed[2,
          , drop=F] ); counter.child <- counter.child+1},
         NA)
  ifelse(counter.child>population, break, NA)
}
for(i in 1:population) offspring[i, nov+1] <- calc.fit( offspring[i, 1:nov, drop=F], data.cov, nos )

# 淘汰
temp.group <- rbind( group, offspring)
## 重複個体を削除
dup.flag <- duplicated( c( temp.group[, 1:nov, drop=F] %*% matrix( rev( 1*10^( 0:(nov-1) ) ), ncol=1 ) ) )
ifelse( any(dup.flag)==T, temp.group <- temp.group[-which(dup.flag==T),], NA)
## 多様性が低すぎる場合は補充
ifelse( nrow(temp.group)<population, shortage <- population-nrow(temp.group), shortage <- 0)
ifelse( shortage>=1,
       {supl.group <- matrix( c( rep(0, shortage*nov),rep(99, shortage) ), nrow=shortage, ncol=nov+1 );
        supl.group[1, 1:nov] <- generate.dna(nof,nov);
        supl.group[1, nov+1] <- calc.fit( supl.group[1,1:nov,drop=F], data.cov, nos )},
       NA
       )
ifelse( shortage>=2,
       for(i in 2:shortage){
         supl.group[i,1:nov] <- generate.dna(nof,nov)
         while( check.duplication(i,supl.group,nof,nov)==TRUE ) supl.group[i,1:nov] <- generate.dna(nof,nov)
         supl.group[i, nov+1] <- calc.fit( supl.group[i,1:nov,drop=F], data.cov, nos )
       },
       NA
       )
ifelse( shortage>=1, temp.group <- rbind(temp.group, supl.group), NA)
## 候補が多い場合には生存競争が発生
temp.group <- temp.group[ order( temp.group[,nov+1] ), ]
ifelse(nrow(temp.group)>population,
       {elite <- temp.group[1, ,drop=F];
        temp.group <- temp.group[-1,];
        next.group <- rbind(elite, temp.group[ sample( nrow(temp.group),
         population-1, prob=replace(1/temp.group[,nov+1], is.infinite(1/temp.group[,nov+1]), 1e-99) ),] )},
       next.group <- temp.group
       )

# 局所探索
```

```
  for(z in 1:population){
  for( i in 1:nov ){
  change.list <- setdiff(1:nof, next.group[z,1:nov][i])
  new.dna <- as.list(NA)
  for(j in 1:(nof-1)) {new.dna[[j]] <- matrix( next.group[z,1:nov],
   nrow=1) ; new.dna[[j]][i] <- change.list[j]}
  new.rmsea <- rep( 99, (nof-1) )
  for(j in 1:(nof-1)) ifelse( check.dna(new.dna[[j]], nof)==T,
   new.rmsea[j] <- calc.fit( reorder.dna(new.dna[[j]]), data.cov, nos), new.rmsea <- 99)
  ifelse( min(new.rmsea)<next.group[z, nov+1], {next.group[z,
  1:nov] <- reorder.dna( new.dna[[which(new.rmsea==min(new.rmsea))]] ); next.group[z,
  (nov+1)] <- min(new.rmsea); break;}, NA)
  }}
# }

# 結果の書き出しおよび収束判定
generation <- generation + 1
next.group <- next.group[ order( next.group[,nov+1] ), ]
replace( next.group, is.na(next.group), 999 )
replace( next.group, is.null(next.group), 999 )
group <- next.group
evolution.histry[[generation]] <- list("Generation"=generation,
 "Best DNA"=group[1,1:nov], "Fit"=group[1,(nov+1)], "Population Detail"=group )
ifelse(evolution.histry[[generation]][["Fit"]]<=criterion.fit,
 {end.condition <- "Enough Fit"; break.flag <- 1}, NA)
ifelse(generation==criterion.generation,
 {end.condition <- "Max Iteration"; break.flag <- 1}, NA)
ifelse( abs(evolution.histry[[generation]][["Fit"]]-evolution.histry[[(generation-1)]][["Fit"]])<=
  criterion.change, counter <- counter+1, counter <- 1)
ifelse(counter>=criterion.stop, {end.condition <- "Evolution Stasis"; break.flag <- 1}, NA)
}

final.out <- evolution.histry
}

exe.sim <- function(rep.times){
a <- c(1:rep.times)
output.list <- formatC(a, width=3, flag="0")
speed.result <- matrix(0, nrow=rep.times, ncol=3)
for(i in 1:rep.times){
  fname <- output.list[i]
  time.out <- system.time( out <-  try(ga.search()) )
    out.hist <- capture.output(out)
    write(out.hist, file=paste(fname, "histry.txt", sep="") )
    speed.result[i,] <- time.out[1:3]
    write.table(speed.result, file="SpeedResult_10-05.txt")
  }
}

# ライブラリの読み込み
library('MASS')
library('sem')
# 作業ディレクトリの設定
setwd('H:\\Temp')

exe.sim(50)
```

Appendix B

R code of the algorithm for a general factor analysis model

```
setwd("D:\\Temp")
library(MASS)
library(sem)
set.seed(runif(n=1,min=-65535,max=65535))

searching <- function(){
#いじる設定類
データタイプ<<-1
集団サイズ<<-30
突然変異率<<-0.05

#いじらない設定類
因子 <<- 5
観測変数 <<- 20
サンプル数 <<- 647
子供生産数 <<- 集団サイズ
エリート数<<-2
世代数上限 <<- 100
適合度基準 <<- -2
停滞世代数 <<- 50

#自動計算
染色体形式<<-c(rep("m", 因子), rep("b", 因子*(観測変数-因子)), rep("b", (因子^2-因子)
/2))
遺伝子下限<<-c(rep(1, 因子), rep(0, 因子*(観測変数-因子)), rep(0, (因子^2-因子)/2))
遺伝子上限<<-c(rep(観測変数, 因子), rep(1, 因子*(観測変数-因子)), rep(1, (因子^2-因
子)/2))

#データ発生
inp <- read.table("BF.txt")
data.cov <- as.matrix(inp)
rownames(data.cov) <- colnames(data.cov)

dataFormat <- initRamSpec(ObsVar=観測変数, LatVar=因子)
dataFormat <- genSimpleFact(dataFormat, factLoad="runif(n=1, min=0.7, max=0.9)", fact
Cov="runif(n=1, min=0.4, max=0.6)", obsErr="runif(n=1, min=0.2, max=0.5)", add="only1
")
mydata <<- genGSRam(dataFormat, サンプル数)

mydata@GenCovMatVar<<-data.cov


#初期個体発生
chromFormat <- initChromFormat( type=染色体形式, min=遺伝子下限, max=遺伝子上限 )
main <- initGAEval(object=chromFormat, data=mydata, population=集団サイズ, genFunc="g
enIndCELFA", evalFunc="evalFitCELFA", evalFuncOp=c("dataobject=mydata"))

continue<<-0
repeat{
backupMain <<- main
backupState <<- 1
save.image()
    #繁殖
    MatingPool_inp <- matrix( 0, nrow=子供生産数, ncol=ncol(main@Population) )
    mateCount <- 1
    while(mateCount<=子供生産数){
        parents <- selectCELFA(main)
        children <- crossCELFA(parents, 因子)
        children <- mutateCELFA(children, rate=突然変異率, nof=因子, nov=観測変数)
        children[1,] <- reorderCELFA(children[1,], nof=因子, nov=観測変数)
        children[2,] <- reorderCELFA(children[2,], nof=因子, nov=観測変数)
        if( chromValidCELFA(children[1,], nof=因子, nov=観測変数)==TRUE && mateCount<
=子供生産数){
            MatingPool_inp[mateCount,] <- children[1,]
            mateCount <- mateCount+1
        }
        if( chromValidCELFA(children[2,], nof=因子, nov=観測変数)==TRUE && mateCount<
=子供生産数){
            MatingPool_inp[mateCount,] <- children[2,]
            mateCount <- mateCount+1
        }
    }
    main@MatingPool <- MatingPool_inp
    MatingFitness_inp <- numeric( nrow(main@MatingPool) )
    for( i in 1:nrow(main@MatingPool) ){
        gc()
        MatingFitness_inp[i] <- evalFitCELFA( main@MatingPool[i,],mydata )
    }
    main@MatingFitness <- MatingFitness_inp
    #局所探索
backupMain <<- main
backupState <<- 2
save.image()
    for(i in 1:1:nrow(main@MatingPool)) main <- localCELFA(main, mydata, i)
backupMain <<- main
backupState <<- 3
save.image()
    for(i in 1:1:nrow(main@Population)) main <- localCELFA2(main, mydata, i)
    #置き換えと世代更新
    main <- replaceCELFA(main, elite=エリート数)
    #終了判定
    if(main@Generation==世代数上限) break
    if(main@Fitness[1]<=適合度基準) break
    if( main@History[[main@Generation]]$Fitness[1]==main@History[[main@Generation-1]]
$Fitness[1] ){
        continue <<- continue+1
    } else{
        continue <<- 0
    }
    if(continue==停滞世代数) break
}

resultChrom <- main@Population[1,]
resultFit <- main@Fitness[1]
resultPattern <- decodeCELFA(resultChrom, nof=因子, nov=観測変数)

trueCoef <- matrix(0,nrow=観測変数, ncol=因子)
trueCoef[which(getCoefMat(mydata,"l2o")>0)] <- 1
trueCov <- matrix(0,nrow=因子, ncol=因子)
trueCov[which(getCovMatResid(mydata,"l2l")>0)] <- 1
truePattern <- list(Loadings=trueCoef, Covariance=trueCov)
trueChrom <- reorderCELFA( encodeCELFA(truePattern), nof=因子, nov=観測変数)
truePattern <- decodeCELFA(trueChrom, nof=因子, nov=観測変数)
trueFit <- evalFitCELFA(trueChrom, mydata)

judge <- 0
if( all(resultChrom==trueChrom) ){
    judge<-1
    } else if( all(resultPattern$Loadings==truePattern$Loadings) ){
    judge <- 2
    } else if( resultFit<=trueFit ){
    judge <- 3
    }

output <- list(ElapsedGeneration=main@Generation, Judge=judge,
                      TrueRMSEA=trueFit, ExploredRMSEA=resultFit, TrueModel=truePatte
rn, ExploredModel=resultPattern,
                      TrueChromosome=trueChrom, ExploredChromosome=resultChrom,
                      History=main@History, Data=mydata@GenCovMatVar  )

out.hist <- capture.output(output)
name <- paste(format(Sys.time(), "%Y%m%d_%H%M%S"), ".txt", sep="")
write(out.hist, file=name)
file.remove(".Rdata")

return(output)
}

searching1 <- function(){
main<-backupMain

repeat{
backupMain <<- main
backupState <<- 1
save.image()
    #繁殖
    MatingPool_inp <- matrix( 0, nrow=子供生産数, ncol=ncol(main@Population) )
    mateCount <- 1
    while(mateCount<=子供生産数){
        parents <- selectCELFA(main)
        children <- crossCELFA(parents, 因子)
        children <- mutateCELFA(children, rate=突然変異率, nof=因子, nov=観測変数)
        children[1,] <- reorderCELFA(children[1,], nof=因子, nov=観測変数)
        children[2,] <- reorderCELFA(children[2,], nof=因子, nov=観測変数)
        if( chromValidCELFA(children[1,], nof=因子, nov=観測変数)==TRUE && mateCount<
=子供生産数){
            MatingPool_inp[mateCount,] <- children[1,]
            mateCount <- mateCount+1
        }
        if( chromValidCELFA(children[2,], nof=因子, nov=観測変数)==TRUE && mateCount<
=子供生産数){
            MatingPool_inp[mateCount,] <- children[2,]
            mateCount <- mateCount+1
        }
    }
    main@MatingPool <- MatingPool_inp
    MatingFitness_inp <- numeric( nrow(main@MatingPool) )
    for( i in 1:nrow(main@MatingPool) ){
        gc()
        MatingFitness_inp[i] <- evalFitCELFA( main@MatingPool[i,],mydata )
    }
    main@MatingFitness <- MatingFitness_inp
    #局所探索
backupMain <<- main
backupState <<- 2
save.image()
    for(i in 1:1:nrow(main@MatingPool)) main <- localCELFA(main, mydata, i)
backupMain <<- main
backupState <<- 3
save.image()
    for(i in 1:1:nrow(main@Population)) main <- localCELFA2(main, mydata, i)
    #置き換えと世代更新
    main <- replaceCELFA(main, elite=エリート数)
    #終了判定
    if(main@Generation==世代数上限) break
    if(main@Fitness[1]<=適合度基準) break
    if( main@History[[main@Generation]]$Fitness[1]==main@History[[main@Generation-1]]
$Fitness[1] ){
        continue <<- continue+1
    } else{
        continue <<- 0
    }
    if(continue==停滞世代数) break
}

resultChrom <- main@Population[1,]
resultFit <- main@Fitness[1]
resultPattern <- decodeCELFA(resultChrom, nof=因子, nov=観測変数)

trueCoef <- matrix(0,nrow=観測変数, ncol=因子)
trueCoef[which(getCoefMat(mydata,"l2o")>0)] <- 1
trueCov <- matrix(0,nrow=因子, ncol=因子)
trueCov[which(getCovMatResid(mydata,"l2l")>0)] <- 1
truePattern <- list(Loadings=trueCoef, Covariance=trueCov)
trueChrom <- reorderCELFA( encodeCELFA(truePattern), nof=因子, nov=観測変数)
truePattern <- decodeCELFA(trueChrom, nof=因子, nov=観測変数)
```

```
trueFit <- evalFitCELFA(trueChrom, mydata)

judge <- 0
if( all(resultChrom==trueChrom) ){
    judge<-1
    } else if( all(resultPattern$Loadings==truePattern$Loadings) ){
    judge <- 2
    } else if( resultFit<=trueFit ){
    judge <- 3
    }

output <- list(ElapsedGeneration=main@Generation, Judge=judge,
                      TrueRMSEA=trueFit, ExploredRMSEA=resultFit, TrueModel=truePatte
rn, ExploredModel=resultPattern,
                      TrueChromosome=trueChrom, ExploredChromosome=resultChrom,
                      History=main@History, Data=mydata@GenCovMatVar  )

out.hist <- capture.output(output)
name <- paste(format(Sys.time(), "%Y%m%d_%H%M%S"), ".txt", sep="")
write(out.hist, file=name)
file.remove(".Rdata")

return(output)
}

searching2 <- function(){
main<-backupMain

repeat{
    #局所探索
backupMain <<- main
backupState <<- 2
save.image()
    for(i in 1:1:nrow(main@MatingPool)) main <- localCELFA(main, mydata, i)
backupMain <<- main
backupState <<- 3
save.image()
    for(i in 1:1:nrow(main@Population)) main <- localCELFA2(main, mydata, i)
    #置き換えと世代更新
    main <- replaceCELFA(main, elite=エリート数)
    #終了判定
    if(main@Generation==世代数上限) break
    if(main@Fitness[1]<=適合度基準) break
    if( main@History[[main@Generation]]$Fitness[1]==main@History[[main@Generation-1]]
$Fitness[1] ){
        continue <<- continue+1
    } else{
        continue <<- 0
    }
    if(continue==停滞世代数) break
backupMain <<- main
backupState <<- 1
save.image()
    #繁殖
    MatingPool_inp <- matrix( 0, nrow=子供生産数, ncol=ncol(main@Population) )
    mateCount <- 1
    while(mateCount<=子供生産数){
        parents <- selectCELFA(main)
        children <- crossCELFA(parents, 因子)
        children <- mutateCELFA(children, rate=突然変異率, nof=因子, nov=観測変数)
        children[1,] <- reorderCELFA(children[1,], nof=因子, nov=観測変数)
        children[2,] <- reorderCELFA(children[2,], nof=因子, nov=観測変数)
        if( chromValidCELFA(children[1,], nof=因子, nov=観測変数)==TRUE && mateCount<
=子供生産数){
            MatingPool_inp[mateCount,] <- children[1,]
            mateCount <- mateCount+1
        }
        if( chromValidCELFA(children[2,], nof=因子, nov=観測変数)==TRUE && mateCount<
=子供生産数){
            MatingPool_inp[mateCount,] <- children[2,]
            mateCount <- mateCount+1
        }
    }
    main@MatingPool <- MatingPool_inp
    MatingFitness_inp <- numeric( nrow(main@MatingPool) )
    for( i in 1:nrow(main@MatingPool) ){
        gc()
        MatingFitness_inp[i] <- evalFitCELFA( main@MatingPool[i,],mydata )
    }
    main@MatingFitness <- MatingFitness_inp
}

resultChrom <- main@Population[1,]
resultFit <- main@Fitness[1]
resultPattern <- decodeCELFA(resultChrom, nof=因子, nov=観測変数)

trueCoef <- matrix(0,nrow=観測変数, ncol=因子)
trueCoef[which(getCoefMat(mydata,"l2o")>0)] <- 1
trueCov <- matrix(0,nrow=因子, ncol=因子)
trueCov[which(getCovMatResid(mydata,"l2l")>0)] <- 1
truePattern <- list(Loadings=trueCoef, Covariance=trueCov)
trueChrom <- reorderCELFA( encodeCELFA(truePattern), nof=因子, nov=観測変数)
truePattern <- decodeCELFA(trueChrom, nof=因子, nov=観測変数)
trueFit <- evalFitCELFA(trueChrom, mydata)

judge <- 0
if( all(resultChrom==trueChrom) ){
    judge<-1
    } else if( all(resultPattern$Loadings==truePattern$Loadings) ){
    judge <- 2
    } else if( resultFit<=trueFit ){
    judge <- 3
    }

output <- list(ElapsedGeneration=main@Generation, Judge=judge,
                      TrueRMSEA=trueFit, ExploredRMSEA=resultFit, TrueModel=truePatte
rn, ExploredModel=resultPattern,
                      TrueChromosome=trueChrom, ExploredChromosome=resultChrom,
                      History=main@History, Data=mydata@GenCovMatVar  )

out.hist <- capture.output(output)
name <- paste(format(Sys.time(), "%Y%m%d_%H%M%S"), ".txt", sep="")
write(out.hist, file=name)
file.remove(".Rdata")
```

```
return(output)
}

searching3 <- function(){
main<-backupMain

repeat{
backupMain <<- main
backupState <<- 3
save.image()
    for(i in 1:1:nrow(main@Population)) main <- localCELFA2(main, mydata, i)
    #置き換えと世代更新
    main <- replaceCELFA(main, elite=エリート数)
    #終了判定
    if(main@Generation==世代数上限) break
    if(main@Fitness[1]<=適合度基準) break
    if( main@History[[main@Generation]]$Fitness[1]==main@History[[main@Generation-1]]
$Fitness[1] ){
        continue <<- continue+1
    } else{
        continue <<- 0
    }
    if(continue==停滞世代数) break
backupMain <<- main
backupState <<- 1
save.image()
    #繁殖
    MatingPool_inp <- matrix( 0, nrow=子供生産数, ncol=ncol(main@Population) )
    mateCount <- 1
    while(mateCount<=子供生産数){
        parents <- selectCELFA(main)
        children <- crossCELFA(parents, 因子)
        children <- mutateCELFA(children, rate=突然変異率, nof=因子, nov=観測変数)
        children[1,] <- reorderCELFA(children[1,], nof=因子, nov=観測変数)
        children[2,] <- reorderCELFA(children[2,], nof=因子, nov=観測変数)
        if( chromValidCELFA(children[1,], nof=因子, nov=観測変数)==TRUE && mateCount<
=子供生産数){
            MatingPool_inp[mateCount,] <- children[1,]
            mateCount <- mateCount+1
        }
        if( chromValidCELFA(children[2,], nof=因子, nov=観測変数)==TRUE && mateCount<
=子供生産数){
            MatingPool_inp[mateCount,] <- children[2,]
            mateCount <- mateCount+1
        }
    }
    main@MatingPool <- MatingPool_inp
    MatingFitness_inp <- numeric( nrow(main@MatingPool) )
    for( i in 1:nrow(main@MatingPool) ){
        gc()
        MatingFitness_inp[i] <- evalFitCELFA( main@MatingPool[i,],mydata )
    }
    main@MatingFitness <- MatingFitness_inp
    #局所探索
backupMain <<- main
backupState <<- 2
```

```
save.image()
    for(i in 1:1:nrow(main@MatingPool)) main <- localCELFA(main, mydata, i)
}

resultChrom <- main@Population[1,]
resultFit <- main@Fitness[1]
resultPattern <- decodeCELFA(resultChrom, nof=因子, nov=観測変数)

trueCoef <- matrix(0,nrow=観測変数, ncol=因子)
trueCoef[which(getCoefMat(mydata,"l2o")>0)] <- 1
trueCov <- matrix(0,nrow=因子, ncol=因子)
trueCov[which(getCovMatResid(mydata,"l2l")>0)] <- 1
truePattern <- list(Loadings=trueCoef, Covariance=trueCov)
trueChrom <- reorderCELFA( encodeCELFA(truePattern), nof=因子, nov=観測変数)
truePattern <- decodeCELFA(trueChrom, nof=因子, nov=観測変数)
trueFit <- evalFitCELFA(trueChrom, mydata)

judge <- 0
if( all(resultChrom==trueChrom) ){
    judge<-1
    } else if( all(resultPattern$Loadings==truePattern$Loadings) ){
    judge <- 2
    } else if( resultFit<=trueFit ){
    judge <- 3
    }

output <- list(ElapsedGeneration=main@Generation, Judge=judge,
                      TrueRMSEA=trueFit, ExploredRMSEA=resultFit, TrueModel=truePatte
rn, ExploredModel=resultPattern,
                      TrueChromosome=trueChrom, ExploredChromosome=resultChrom,
                      History=main@History, Data=mydata@GenCovMatVar  )

out.hist <- capture.output(output)
name <- paste(format(Sys.time(), "%Y%m%d_%H%M%S"), ".txt", sep="")
write(out.hist, file=name)
file.remove(".Rdata")

return(output)
}


exe.sim <- function(rep.times){
a <- c(1:rep.times)
output.list <- formatC(a, width=3, flag="0")
total.result <- matrix(0, nrow=rep.times, ncol=4)
speed.result <- matrix(0, nrow=rep.times, ncol=3)
for(i in 1:rep.times){
  fname <- output.list[i]
  time.out <- system.time( out <-  try(searching()) )
    out.hist <- capture.output(out)
    write(out.hist, file=paste(fname, "detail.txt", sep="") )
    total.result[i,1] <- out$Judge
    total.result[i,2] <- out$ElapsedGeneration
    total.result[i,3] <- out$ExploredRMSEA
    total.result[i,4] <- out$TrueRMSEA
    write.table(total.result, file="TotalResult.txt")
    speed.result[i,] <- time.out[1:3]
    write.table(speed.result, file="SpeedResult.txt")
  }
}

###############################################
#クラス RAMmodel の形式チェックを行う関数 validRamSpec()
###############################################
validRamSpec <- function( object ){
    ################################
    #スロット ObsVar の形式チェック: integer は確定
    ################################
    #長さが 1 でなければならない
    if( length(object@ObsVar)!=1 ) return("Slot 'ObsVar' must have a single value.")
    #値が有限な実数でなければならない
    if( !isTRUE( all( is.finite(object@ObsVar) ) ) ) return("Slot 'ObsVar' must have
a finite real number value.")
    #値が 0 より大きくなければならない
    if( !isTRUE( object@ObsVar>0) ) return("Slot 'ObsVar' must have a positive value.
")

    ################################
    #スロット LatVar の形式チェック: integer は確定
    ################################
    #長さが 1 でなければならない
    if( length(object@LatVar)!=1 ) return("Slot 'LatVar' must have a single value.")
    #値が有限な実数でなければならない
    if( !isTRUE( all( is.finite(object@LatVar) ) ) ) return("Slot 'LatVar' must have
a finite real number value.")
    #値が 0 以上でなければならない
    if( !isTRUE( object@LatVar>=0) ) return("Slot 'LatVar' must have a value greater
than or equal to zero.")

    ################################
    #スロット MeanVec の形式チェック: numeric は確定
    ################################
    #長さが ObsVar+LatVar でなければならない
    if( length(object@MeanVec) != object@ObsVar + object@LatVar )
        return("Length of 'MeanVec' must be equal to 'ObsVar' + 'LatVar'.")
    #値が有限な実数でなければならない
    if( !isTRUE( all( is.finite(object@MeanVec) ) ) ) return("Slot 'MeanVec' must hav
e finite real number values.")

    ################################
    #スロット CoefMat の形式チェック: matrix は確定
    ################################
    #サイズが (ObsVar+LatVar)*(ObsVar+LatVar) でなければならない
    if( !isTRUE( all( dim(object@CoefMat)==object@ObsVar+object@LatVar ) ) )
        return("'CoefMat' must be a size of ('ObsVar'+'LatVar')*('ObsVar'+'LatVar') m
atrix.")
    #要素は全て numeric でなければならない
    if( !isTRUE( all( is.numeric(object@CoefMat) ) ) ) return("Slot 'CoefMat' must ha
ve numerical values.")
    #要素は全て有限な実数でなければならない
    if( !isTRUE( all( is.finite(object@CoefMat) ) ) ) return("Slot 'CoefMat' must hav
e finite real number values.")

    ################################
    #スロット CovMatResid の形式チェック: matrix は確定
    ################################
    #サイズが (ObsVar+LatVar)*(ObsVar+LatVar) でなければならない
    if( !isTRUE( all( dim(object@CovMatResid)==object@ObsVar+object@LatVar ) ) )
        return("'CovMatResid' must be a size of ('ObsVar'+'LatVar')*('ObsVar'+'LatVar
') matrix.")
    #要素は全て numeric でなければならない
    if( !isTRUE( all( is.numeric(object@CovMatResid) ) ) ) return("'CovMatResid' must
 have numerical values.")
    #要素は全て有限な実数でなければならない
    if( !isTRUE( all( is.finite(object@CovMatResid) ) ) ) return("Slot 'CovMatResid'
must have finite real number values.")
    #要素は全て 0 以上でなければならない
    if( !isTRUE( all( object@CovMatResid>=0 ) ) ) return("'CovMatResid' must have val
ues greater than or equal to zero.")
    #ブロック対角でなければならない
    if( identical( object@CovMatResid[ c(1:object@LatVar), c( (object@LatVar+1):(obje
ct@LatVar+object@ObsVar) ) ],
                   t(object@CovMatResid[ c( (object@LatVar+1):(object@LatVar+object
@ObsVar) ), c(1:object@LatVar) ]) )==FALSE)
        return("Top right block of 'CovMatResid' and bottom left block of 'CovMatResi
d' must be symmetric.")

    #以上全てに引っかからなければ大丈夫
    return(TRUE)
}


##############################################
#RAM モデルにより平均共分散構造を表現するクラス RamSpec の定義
##############################################
setClass(
    "RamSpec",
    representation(
        ObsVar = "integer",     #観測変数の数
        LatVar = "integer",      #潜在変数の数
        MeanVec = "numeric",       #平均ベクトル
        CoefMat = "matrix",          #係数行列
        CovMatResid = "matrix"    #残差の共分散行列
    ),
    validity=validRamSpec
)


##############################################
#クラス RamSpec の標準初期化関数 initRamSpec()
#ObsVar: 観測変数の数. 正の整数.
#LatVar: 潜在変数の数. 0 以上の整数.
#その他のスロットについては, インスタンス生成後にセッタを使って適宜代入する.
##############################################
initRamSpec <- function( ObsVar, LatVar=0 ){
    ################################
    #ObsVar の形式チェック
    ################################
    #ObsVar が数値ベクトルでない場合に警告する.
    if( !is.numeric(ObsVar) ) warning("'ObsVar' is coerced to be a numeric type objec
t.")
```

```
    #ObsVar を数値ベクトルに変換する.
    ObsVar <- as.numeric(ObsVar)
    #ObsVar の長さが 2 以上の場合に警告する.
    if( length(ObsVar) > 1 ) warning("'ObsVar' contains more than one element. Using
the first one.")
    #ObsVar の長さを 1 に合わせる. 長さが 0 の場合は NA が代入される.
    ObsVar <- ObsVar[1]
    #ObsVar が有限の実数値でない場合, 強制終了する.
    if( !isTRUE( is.finite(ObsVar)  ) ) stop("Invalid 'ObsVar'; you must specify fini
te real number value for 'ObsVar'.")
    #ObsVar が整数でない場合, 小数点以下は切り捨てる.
    if( !identical( ObsVar, trunc(ObsVar) ) ) {
        warning("'ObsVar' is truncated toward 0.")
        ObsVar <- trunc(ObsVar)
    }
    #ObsVar が 0 以下の場合, 強制終了する.
    if( ObsVar <= 0 ) stop("Invalid 'ObsVar'; you must specify positive integer value
 for 'ObsVar'.")
    #num.ObsVar の代入値を決定
    ObsVar_inp <- as.integer( ObsVar )
    ################################
    #LatVar の形式チェック
    ################################
    #LatVar が数値ベクトルでない場合に警告する.
    if( !is.numeric(LatVar) ) warning("'LatVar' is coerced to be a numeric type objec
t.")
    #LatVar を数値ベクトルに変換する.
    LatVar <- as.numeric(LatVar)
    #LatVar の長さが 2 以上の場合に警告する.
    if( length(LatVar) > 1 ) warning("'LatVar' contains more than one element. Using
the first one.")
    #LatVar の長さを 1 に合わせる. 長さが 0 の場合は NA が代入される.
    LatVar <- LatVar[1]
    #LatVar が有限の実数値でない場合, 強制終了する.
    if( !isTRUE( is.finite(LatVar)  ) ) stop("Invalid 'LatVar'; you must specify fini
te real number value for 'LatVar'.")
    #LatVar が整数でない場合, 小数点以下は切り捨てる.
    if( !identical( LatVar, trunc(LatVar) ) ) {
        warning("'LatVar' is truncated toward 0.")
        LatVar <- trunc(LatVar)
    }
    #LatVar が 0 未満の場合, 強制終了する.
    if( LatVar < 0 ) stop("Invalid 'LatVar'; you must specify integer value greater t
han zero for 'LatVar'.")
    #num.LatVar の代入値を決定
    LatVar_inp <- as.integer( LatVar )
    ################################
    #その他のスロットは, サイズだけ合わせた値 0 のベクトル, 行列で初期化.
    ################################
    #モデルに含まれる変数の数を算出
    TotalVar <- ObsVar_inp + LatVar_inp
    #平均ベクトルと残差ベクトル
    MeanVec_inp <- rep( 0, TotalVar )
    #係数行列と残差の共分散行列
    CovMatResid_inp <- CoefMat_inp <- matrix( 0, nrow=TotalVar, ncol=TotalVar )
    ################################
    #クラス RamSpec のオブジェクトを新規生成
    ################################
    object <- new(
        "RamSpec",
        ObsVar=ObsVar_inp, LatVar=LatVar_inp,
        MeanVec=MeanVec_inp, CoefMat=CoefMat_inp, CovMatResid=CovMatResid_inp
    )
    return(object)
}


##############################################
#クラス RamSpec のオブジェクトのための閲覧関数 show.RamSpec()
#object: 対象オブジェクト
##############################################
setMethod( "show", "RamSpec",
    function(object){
        message("<Class 'RamSpec' object to represent model specification for SEM in
RAM format>", "\n")
        message("Number of Observed Variables @ Slot 'ObsVar':")
        print(object@ObsVar)
        message("\n")
        message("Number of Latent Variables @ Slot 'LatVar':")
        print(object@LatVar)
        message("\n")
        message("Mean Vector (alpha_zero) @ Slot 'MeanVec':")
        print(object@MeanVec)
        message("\n")
        message("Coefficient Matrix (Alpha) @ Slot 'CoefMat':")
        print(object@CoefMat)
        message("\n")
        message("Residual Covariance Matrix (Sigma_u) @ Slot 'CovMatResid':")
        print(object@CovMatResid)
        }
)


##############################################
#クラス RamSpec のオブジェクトのための要約関数 summary.RamSpec()
#object: 対象オブジェクト
##############################################
setMethod( "summary", "RamSpec",
    function(object){
        check <- validObject(object, test=TRUE)
        if( isTRUE(check) ) message("This is a valid 'RamSpec' object.")
        if( is.character(check) ) {
            message("This is not a valid 'RamSpec' object;", "\n", check)
            return( invisible(check) )
            }
        message("\n")
        message("Number of Observed Variables: ", object@ObsVar)
        message("Number of Latent Variables: ", object@LatVar)
        message("\n")
        message("Covariance Matrix of Observed Variables:", "\n")
        print( getCovMatVar(object) )
        }
)
```

```
###############################################
#クラス RamSpec オブジェクトのためのローレベルゲッター getObsVar()
#object: 対象オブジェクト
##############################################
setGeneric("getObsVar",
    function(object, ...){
        value <- standardGeneric("getObsVar")
        return(value)
    }
)
setMethod("getObsVar", "RamSpec",
    function( object ){
    #適切な"RamSpec"オブジェクトかどうかをチェック
    check <- validObject(object, test=TRUE)
    if( is.character(check) ) stop("Invalid 'RamSpec' object;", "\n", check)
    #スロット num.ObsVar を返す
    return(object@ObsVar)
    }
)

##############################################
#クラス RamSpec オブジェクトのためのローレベルゲッター getLatVar()
#object: 対象オブジェクト
##############################################
setGeneric("getLatVar",
    function(object, ...){
        value <- standardGeneric("getLatVar")
        return(value)
    }
)
setMethod("getLatVar", "RamSpec",
    function( object ){
        #適切な"RamSpec"オブジェクトかどうかをチェック
        check <- validObject(object, test=TRUE)
        if( is.character(check) ) stop("Invalid 'RamSpec' object;", "\n", check)
        #スロット num.LatVar を返す
        return(object@LatVar)
    }
)

##############################################
#クラス RamSpec オブジェクトのためのローレベルゲッター getMeanVec()
#object: 対象オブジェクト
#type: 取得するベクトルの範囲
##############################################
setGeneric("getMeanVec",
    function(object, ...){
        value <- standardGeneric("getMeanVec")
        return(value)
    }
)
setMethod("getMeanVec", "RamSpec",
    function( object, type=c("all", "latent", "observed") ){
        #適切な"RamSpec"オブジェクトかどうかをチェック
        check <- validObject(object, test=TRUE)
        if( is.character(check) ) stop("Invalid 'RamSpec' object;", "\n", check)
        #type 引数の中身を決定
        type <- match.arg(type)
        #スロット MeanVec を返す
        out <- switch(
            type,
            #all の場合は平均ベクトル全体
            all = object@MeanVec,
            #latent の場合は因子の平均ベクトル
            latent = object@MeanVec[ 1 : object@LatVar ],
            #observed の場合は観測変数の平均ベクトル
            observed = object@MeanVec[ (object@LatVar+1) : (object@LatVar+object@ObsV
ar) ]
        )
        return(out)
    }
)

##############################################
#クラス RamSpec オブジェクトのためのローレベルゲッター getCoefMat()
#object: 対象オブジェクト
#type: 取得する行列の範囲
##############################################
setGeneric("getCoefMat",
    function(object, ...){
        value <- standardGeneric("getCoefMat")
        return(value)
    }
)
setMethod("getCoefMat", "RamSpec",
    function( object, type=c("all", "l2l", "l2o", "o2l", "o2o") ){
        #適切な"RamSpec"オブジェクトかどうかをチェック
        check <- validObject(object, test=TRUE)
        if( is.character(check) ) stop("Invalid 'RamSpec' object;", "\n", check)
        #type 引数の中身を決定
        type <- match.arg(type)
        #スロット CoefMat を返す
        out <- switch(
            type,
            #all の場合は係数行列全体
            all = object@CoefMat,
            #l2l の場合は，因子間の係数行列
            l2l = object@CoefMat[ c(1:object@LatVar), c(1:object@LatVar) ],
            #l2o の場合は，因子から観測変数への係数行列
            l2o = object@CoefMat[ c( (object@LatVar+1):(object@LatVar+object@ObsVar)
 ), c(1:object@LatVar) ],
            #o2l の場合は，観測変数から因子への係数行列
            o2l = object@CoefMat[ c(1:object@LatVar), c( (object@LatVar+1):(object@La
tVar+object@ObsVar) ) ],
            #o2o の場合は，観測変数間の係数行列
            o2o = object@CoefMat[ c( (object@LatVar+1):(object@LatVar+object@ObsVar)
 ),
                                                  c( (object@LatVar+1):(object@LatVar
+object@ObsVar) ) ]
        )
        return(out)
    }
)
```

```
##############################################
#クラス RamSpec オブジェクトのためのローレベルゲッター getCovMatResid()
#object: 対象オブジェクト
#type: 取得する行列の範囲
##############################################
setGeneric("getCovMatResid",
    function(object, ...){
        value <- standardGeneric("getCovMatResid")
        return(value)
    }
)
setMethod("getCovMatResid", "RamSpec",
    function( object, type=c("all", "l2l", "l2o", "o2l", "o2o") ){
        #適切な"RamSpec"オブジェクトかどうかをチェック
        check <- validObject(object, test=TRUE)
        if( is.character(check) ) stop("Invalid 'RamSpec' object;", "\n", check)
        #type 引数の中身を決定
        type <- match.arg(type)
        #スロット CoefMat を返す
        out <- switch(
            type,
            #all の場合は係数行列全体
            all = object@CovMatResid,
            #l2l の場合は，因子間の係数行列
            l2l = object@CovMatResid[ c(1:object@LatVar), c(1:object@LatVar) ],
            #l2o の場合は，因子から観測変数への係数行列
            l2o = object@CovMatResid[ c( (object@LatVar+1):(object@LatVar+object@ObsV
ar) ), c(1:object@LatVar) ],
            #o2l の場合は，観測変数から因子への係数行列
            o2l = object@CovMatResid[ c(1:object@LatVar), c( (object@LatVar+1):(objec
t@LatVar+object@ObsVar) ) ],
            #o2o の場合は，観測変数間の係数行列
            o2o = object@CovMatResid[ c( (object@LatVar+1):(object@LatVar+object@ObsV
ar) ),
                                                            c( (object@LatVar+1):(objec
t@LatVar+object@ObsVar) ) ]
        )
        return(out)
    }
)


##############################################
#クラス RamSpec オブジェクトのためのローレベルセッター setMeanVec()
#object: 対象オブジェクト
#value: 設定値
#type: 設定するベクトルの範囲
##############################################
setGeneric("setMeanVec",
    function(object, ...){
        value <- standardGeneric("setMeanVec")
        return(value)
    }
)
setMethod("setMeanVec", "RamSpec",
    function( object, value, type=c("all", "latent", "observed") ){
        ################################
        #適切な"RamSpec"オブジェクトかどうかをチェック
        ################################
        check <- validObject(object, test=TRUE)
        if( is.character(check) ) stop("Invalid 'RamSpec' object;", "\n", check)
        ################################
        #type の形式チェック
        ################################
        #type が文字列ベクトルでない場合に警告する.
        if( !is.character(type) ) warning("'type' is coerced to be a character type o
bject.")
        #type を文字列ベクトルに変換する.
        type <- as.character(type)
        #type 引数の中身を決定
        type <- match.arg(type)
        ################################
        #value の形式チェック
        ################################
        #value が数値ベクトルでない場合に警告する.
        if( !is.numeric(value) ) warning("'value' is coerced to be a numeric type obj
ect.")
        #value を数値ベクトルに変換する.
        value <- as.numeric(value)
        #value の長さが MeanVec に合わない場合に警告する.
        if( length(value)!=switch(
            type,
            #all の場合は平均ベクトル全体
            all = length(object@MeanVec),
            #latent の場合は因子の平均ベクトル
            latent = length( object@MeanVec[ 1 : object@LatVar ] ),
            #observed の場合は観測変数の平均ベクトル
            observed = length( object@MeanVec[ (object@LatVar+1) : (object@LatVar+obj
ect@ObsVar) ] )
            )
        )
            warning("The length of 'value' is not valid for substitution. 'value' is
cut or repeated to be a valid size vector.")
        #value の長さを合わせる.
        value <- switch(
            type,
            #all の場合は平均ベクトル全体
            all = rep( value, length.out=length(object@MeanVec) ),
            #latent の場合は因子の平均ベクトル
            latent = rep( value, length.out=length( object@MeanVec[ 1 : object@LatVar
 ] ) ),
            #observed の場合は観測変数の平均ベクトル
            observed = rep( value, length.out=length( object@MeanVec[ (object@LatVar+
1) : (object@LatVar+object@ObsVar) ] ) )
        )
        #value が有限の実数値でない場合，強制終了する.
        if( !isTRUE( all( is.finite(value) )  ) ) stop("Invalid 'value'; you must spe
cify finite real number values for 'value'.")
        ################################
        #type に応じた代入処理
        ################################
        #all の場合は平均ベクトル全体
        if( isTRUE( identical(type, "all") ) ) object@MeanVec <- value
        #latent の場合は因子の平均ベクトル
        if( isTRUE( identical(type, "latent") ) ) object@MeanVec[ 1 : object@LatVar ]
 <- value
        #observed の場合は観測変数の平均ベクトル
        if( isTRUE( identical(type, "observed") ) ) object@MeanVec[ (object@LatVar+1)
 : (object@LatVar+object@ObsVar) ] <- value
        ################################
        #最終チェック
        ################################
        check <- validObject(object, test=TRUE)
        if( is.character(check) ) stop("Invalid 'RamSpec' object is finally generated.
 Check your arguments again;", "\n", check)
        return( object )
    }
)

##############################################
#クラス RamSpec オブジェクトのためのローレベルセッター setCoefMat()
#object: 対象オブジェクト
#value: 設定値
#type: 設定するベクトルの範囲
##############################################
setGeneric("setCoefMat",
    function(object, ...){
        value <- standardGeneric("setCoefMat")
        return(value)
    }
)
setMethod("setCoefMat", "RamSpec",
    function( object, value, type=c("all", "l2l", "l2o", "o2l", "o2o") ){
        ################################
        #適切な"RamSpec"オブジェクトかどうかをチェック
        ################################
        check <- validObject(object, test=TRUE)
        if( is.character(check) ) stop("Invalid 'RamSpec' object;", "\n", check)
        ################################
        #type の形式チェック
        ################################
        #type が文字列ベクトルでない場合に警告する.
        if( !is.character(type) ) warning("'type' is coerced to be a character type o
bject.")
        #type を文字列ベクトルに変換する.
        type <- as.character(type)
        #type 引数の中身を決定
        type <- match.arg(type)
        ################################
        #value の形式チェック
        ################################
        #value が行列の場合はサイズをチェックし，合わないならば警告する.
            if( is.matrix(value) && !all(
                dim(value)==switch(
                    type,
                    #all の場合は係数行列全体
                    all = dim(object@CoefMat),
                    #l2l の場合は，因子間の係数行列
                    l2l = dim(object@CoefMat[ c(1:object@LatVar), c(1:object@LatVar)
 ]),
                    #l2o の場合は，因子から観測変数への係数行列
                    l2o = dim(object@CoefMat[ c( (object@LatVar+1):(object@LatVar+obj
ect@ObsVar) ), c(1:object@LatVar) ]),
                    #o2l の場合は，観測変数から因子への係数行列
                    o2l = dim(object@CoefMat[ c(1:object@LatVar), c( (object@LatVar+
1):(object@LatVar+object@ObsVar) ) ]),
                    #o2o の場合は，観測変数間の係数行列
                    o2o = dim(object@CoefMat[ c( (object@LatVar+1):(object@LatVar+obj
ect@ObsVar) ),
                                                                c( (object@LatVar+
1):(object@LatVar+object@ObsVar) ) ])
                 )
             )
             ) warning("The size of 'value' is not valid for substitution. 'value' is
 cut or repeated to be a valid size matrix." )
        #value が行列でないならば，問答無用で警告する.
        if( !is.matrix(value) ) warning("'value' is coerced to be a matrix type objec
t.")
        value <- as.vector(value)
        #value が数値ベクトルでない場合に警告する.
        if( !is.numeric(value) ) warning("One or more elements in 'value' are not num
eric. They are coerced to be a numeric object.")
        #value を数値ベクトルに変換する.
        value <- as.numeric(value)
        #value の長さを合わせる.
        value <- switch(
            type,
            #all の場合は係数行列全体
            all = rep( value, length.out=length(object@CoefMat) ),
            #l2l の場合は，因子間の係数行列
            l2l = rep( value, length.out=length(object@CoefMat[ c(1:object@LatVar), c
(1:object@LatVar) ]) ),
            #l2o の場合は，因子から観測変数への係数行列
            l2o = rep( value, length.out=length(object@CoefMat[ c( (object@LatVar+1):
(object@LatVar+object@ObsVar) ),

             c(1:object@LatVar) ]) ),
            #o2l の場合は，観測変数から因子への係数行列
            o2l = rep( value, length.out=length(object@CoefMat[ c(1:object@LatVar),

             c( (object@LatVar+1):(object@LatVar+object@ObsVar) ) ]) ),
            #o2o の場合は，観測変数間の係数行列
            o2o = rep( value, length.out=length(object@CoefMat[ c( (object@LatVar+1):
(object@LatVar+object@ObsVar) ),

              c( (object@LatVar+1):(object@LatVar+object@ObsVar) ) ]) )
        )
        #value が有限の実数値でない場合，強制終了する.
        if( !isTRUE( all( is.finite(value) )  ) ) stop("Invalid 'value'; you must spe
cify finite real number values for 'value'.")
        #value を行列として再構築する.
        value <- switch(
            type,
            #all の場合は係数行列全体
            all = matrix( value, nrow=nrow(object@CoefMat), ncol=ncol(object@CoefMat)
 ),
            #l2l の場合は，因子間の係数行列
            l2l = matrix( value, nrow=nrow(object@CoefMat[ c(1:object@LatVar), c(1:ob
ject@LatVar) ]),
                                          ncol=ncol(object@CoefMat[ c(1:object@LatVa
r), c(1:object@LatVar) ]) ),
            #l2o の場合は，因子から観測変数への係数行列
            l2o = matrix( value,
                          nrow=nrow(object@CoefMat[ c( (object@LatVar+1):(object@LatV
ar+object@ObsVar) ), c(1:object@LatVar) ]),
                          ncol=ncol(object@CoefMat[ c( (object@LatVar+1):(object@LatV
ar+object@ObsVar) ), c(1:object@LatVar) ]) ),
            #o2l の場合は，観測変数から因子への係数行列
            o2l = matrix( value,
                         nrow=nrow(object@CoefMat[ c(1:object@LatVar), c( (object@Lat
Var+1):(object@LatVar+object@ObsVar) ) ]),
                         ncol=ncol(object@CoefMat[ c(1:object@LatVar), c( (object@Lat
Var+1):(object@LatVar+object@ObsVar) ) ]) ),
            #o2o の場合は，観測変数間の係数行列
            o2o = matrix( value, nrow=nrow(object@CoefMat[ c( (object@LatVar+1):(obje
ct@LatVar+object@ObsVar) ),
                                                          c( (object@LatVar+1):(objec
t@LatVar+object@ObsVar) ) ]),
                                       ncol=ncol(object@CoefMat[ c( (object@La
tVar+1):(object@LatVar+object@ObsVar) ),
                                                          c( (object@LatVar+1):(objec
t@LatVar+object@ObsVar) ) ]) )
        )
        ################################
        #type に応じた代入処理
        ################################
        #all の場合は係数行列全体
        if( isTRUE( identical(type, "all") ) ) object@CoefMat <- value
        #l2l の場合は，因子間の係数行列
        if( isTRUE( identical(type, "l2l") ) ) object@CoefMat[ c(1:object@LatVar), c
(1:object@LatVar) ] <- value
        #l2o の場合は，因子から観測変数への係数行列
        if( isTRUE( identical(type, "l2o") ) )
            object@CoefMat[ c( (object@LatVar+1):(object@LatVar+object@ObsVar) ), c
(1:object@LatVar) ] <- value
        #o2l の場合は，観測変数から因子への係数行列
        if( isTRUE( identical(type, "o2l") ) )
            object@CoefMat[ c(1:object@LatVar), c( (object@LatVar+1):(object@LatVar+o
bject@ObsVar) ) ] <- value
        #o2o の場合は，観測変数間の係数行列
        if( isTRUE( identical(type, "o2o") ) )
            object@CoefMat[ c( (object@LatVar+1):(object@LatVar+object@ObsVar) ),
                                        c( (object@LatVar+1):(object@LatVar+object@Ob
sVar) ) ] <- value
        ################################
        #最終チェック
        ################################
        check <- validObject(object, test=TRUE)
        if( is.character(check) ) stop("Invalid 'RamSpec' object is finally generated.
 Check your arguments again;", "\n", check)
        return( object )
    }
)

##############################################
```

```
#クラス RamSpec オブジェクトのためのローレベルセッター setCovMatResid()
#object: 対象オブジェクト
#value: 設定値
#type: 設定するベクトルの範囲
##############################################
setGeneric("setCovMatResid",
    function(object, ...){
        value <- standardGeneric("setCovMatResid")
        return(value)
    }
)
setMethod("setCovMatResid", "RamSpec",
    function( object, value, type=c("all", "l2l", "l2o", "o2l", "o2o") ){
        ################################
        #適切な"RamSpec"オブジェクトかどうかをチェック
        ################################
        check <- validObject(object, test=TRUE)
        if( is.character(check) ) stop("Invalid 'RamSpec' object;", "\n", check)
        ################################
        #type の形式チェック
        ################################
        #type が文字列ベクトルでない場合に警告する.
        if( !is.character(type) ) warning("'type' is coerced to be a character type o
bject.")
        #type を文字列ベクトルに変換する.
        type <- as.character(type)
        #type 引数の中身を決定
        type <- match.arg(type)
        ################################
        #value の形式チェック
        ################################
        #value が行列の場合はサイズをチェックし，合わないならば警告する.
            if( is.matrix(value) && !all(
                dim(value)==switch(
                    type,
                    #all の場合は係数行列全体
                    all = dim(object@CovMatResid),
                    #l2l の場合は，因子間の係数行列
                    l2l = dim(object@CovMatResid[ c(1:object@LatVar), c(1:object@LatV
ar) ]),
                    #l2o の場合は，因子から観測変数への係数行列
                    l2o = dim(object@CovMatResid[ c( (object@LatVar+1):(object@LatVar
+object@ObsVar) ), c(1:object@LatVar) ]),
                    #o2l の場合は，観測変数から因子への係数行列
                    o2l = dim(object@CovMatResid[ c(1:object@LatVar), c( (object@LatV
ar+1):(object@LatVar+object@ObsVar) ) ]),
                    #o2o の場合は，観測変数間の係数行列
                    o2o = dim(object@CovMatResid[ c( (object@LatVar+1):(object@LatVar
+object@ObsVar) ),
                                                                        c( (object@L
atVar+1):(object@LatVar+object@ObsVar) ) ])
                 )
             )
             ) warning("The size of 'value' is not valid for substitution. 'value' is
 cut or repeated to be a valid size matrix." )
        #value が行列でないならば，問答無用で警告する.
        if( !is.matrix(value) ) warning("'value' is coerced to be a matrix type objec
t.")
        value <- as.vector(value)
        #value が数値ベクトルでない場合に警告する.
        if( !is.numeric(value) ) warning("One or more elements in 'value' are not num
eric. They are coerced to be a numeric object.")
        #value を数値ベクトルに変換する.
        value <- as.numeric(value)
        #value が有限の実数値でない場合，強制終了する.
        if( !isTRUE( all( is.finite(value) )  ) ) stop("Invalid 'value'; you must spe
cify finite real number values for 'value'.")
        #value が 0 未満の場合，強制終了する.
        if( !isTRUE( all( value>=0 )  ) ) stop("Invalid 'value'; you must specify val
ues greater than or equal to zero for 'value'.")
        #value の長さを合わせる.
        value <- switch(
            type,
            #all の場合は係数行列全体
            all = rep( value, length.out=length(object@CovMatResid) ),
            #l2l の場合は，因子間の係数行列
            l2l = rep( value, length.out=length(object@CovMatResid[ c(1:object@LatVa
r), c(1:object@LatVar) ]) ),
            #l2o の場合は，因子から観測変数への係数行列
            l2o = rep( value, length.out=length(object@CovMatResid[ c( (object@LatVar
+1):(object@LatVar+object@ObsVar) ),

             c(1:object@LatVar) ]) ),
            #o2l の場合は，観測変数から因子への係数行列
            o2l = rep( value, length.out=length(object@CovMatResid[ c(1:object@LatVa
r),

             c( (object@LatVar+1):(object@LatVar+object@ObsVar) ) ]) ),
            #o2o の場合は，観測変数間の係数行列
            o2o = rep( value, length.out=length(object@CovMatResid[ c( (object@LatVar
+1):(object@LatVar+object@ObsVar) ),

                     c( (object@LatVar+1):(object@LatVar+object@ObsVar) ) ]) )
        )
        #value を行列として再構築する.
        value <- switch(
            type,
            #all の場合は係数行列全体
            all = matrix( value, nrow=nrow(object@CovMatResid), ncol=ncol(object@CovM
atResid) ),
            #l2l の場合は，因子間の係数行列
            l2l = matrix( value, nrow=nrow(object@CovMatResid[ c(1:object@LatVar), c
(1:object@LatVar) ]),
                                           ncol=ncol(object@CovMatResid[ c(1:object@L
atVar), c(1:object@LatVar) ]) ),
            #l2o の場合は，因子から観測変数への係数行列
            l2o = matrix( value,
                      nrow=nrow(object@CovMatResid[ c( (object@LatVar+1):(object@LatV
ar+object@ObsVar) ), c(1:object@LatVar) ]),
                      ncol=ncol(object@CovMatResid[ c( (object@LatVar+1):(object@LatV
ar+object@ObsVar) ), c(1:object@LatVar) ]) ),
            #o2l の場合は，観測変数から因子への係数行列
            o2l = matrix( value,
                       nrow=nrow(object@CovMatResid[ c(1:object@LatVar), c( (object@L
atVar+1):(object@LatVar+object@ObsVar) ) ]),
                       ncol=ncol(object@CovMatResid[ c(1:object@LatVar), c( (object@L
atVar+1):(object@LatVar+object@ObsVar) ) ]) ),
            #o2o の場合は，観測変数間の係数行列
            o2o = matrix( value, nrow=nrow(object@CovMatResid[ c( (object@LatVar+1):
(object@LatVar+object@ObsVar) ),
                                                           c( (object@LatVar+1):(objec
t@LatVar+object@ObsVar) ) ]),
                                               ncol=ncol(object@CovMatResid[ c( (objec
t@LatVar+1):(object@LatVar+object@ObsVar) ),
                                                           c( (object@LatVar+1):(objec
t@LatVar+object@ObsVar) ) ]) )
        )
        ################################
        #type に応じた代入処理
        ################################
        #all の場合は係数行列全体
        if( isTRUE( identical(type, "all") ) ) object@CovMatResid <- value
        #l2l の場合は，因子間の係数行列
        if( isTRUE( identical(type, "l2l") ) ) object@CovMatResid[ c(1:object@LatVar),
 c(1:object@LatVar) ] <- value
        #l2o の場合は，因子から観測変数への係数行列と，その対角ブロック
        if( isTRUE( identical(type, "l2o") ) ){
            object@CovMatResid[ c( (object@LatVar+1):(object@LatVar+object@ObsVar) ),
 c(1:object@LatVar) ] <- value
            object@CovMatResid[ c(1:object@LatVar), c( (object@LatVar+1):(object@LatV
ar+object@ObsVar) ) ] <- t(value)
        }
        #o2l の場合は，観測変数から因子への係数行列と，その対角ブロック
        if( isTRUE( identical(type, "o2l") ) ){
            object@CovMatResid[ c(1:object@LatVar), c( (object@LatVar+1):(object@LatV
ar+object@ObsVar) ) ] <- value
            object@CovMatResid[ c( (object@LatVar+1):(object@LatVar+object@ObsVar) ),
 c(1:object@LatVar) ] <- t(value)
        }
        #o2o の場合は，観測変数間の係数行列
        if( isTRUE( identical(type, "o2o") ) )
            object@CovMatResid[ c( (object@LatVar+1):(object@LatVar+object@ObsVar) ),
                                               c( (object@LatVar+1):(object@LatVar+ob
ject@ObsVar) ) ] <- value
        ################################
        #最終チェック
        ################################
        check <- validObject(object, test=TRUE)
        if( is.character(check) ) stop("Invalid 'RamSpec' object is finally generated.
 Check your arguments again;", "\n", check)
        return( object )
    }
)


##############################################
#クラス RamSpec オブジェクトから変数の共分散行列を計算する関数 getCovMatVar()
#object: 対象オブジェクト
#type: 計算するベクトルの範囲
##############################################
setGeneric("getCovMatVar",
    function(object, ...){
        value <- standardGeneric("getCovMatVar")
        return(value)
    }
)
setMethod("getCovMatVar", "RamSpec",
    function( object, type=c("o2o", "all", "l2l", "l2o", "o2l") ){
        ################################
        #適切な"RamSpec"オブジェクトかどうかをチェック
        ################################
        check <- validObject(object, test=TRUE)
        if( is.character(check) ) stop("Invalid 'RamSpec' object;", "\n", check)
        ################################
        #type の形式チェック
        ################################
        #type が文字列ベクトルでない場合に警告する.
        if( !is.character(type) ) warning("'type' is coerced to be a character type o
bject.")
        #type を文字列ベクトルに変換する.
        type <- as.character(type)
        #type 引数の中身を決定
        type <- match.arg(type)
        ################################
        #変数間の共分散行列の計算
        ################################
        #誘導型の算出
        ReducedForm <- try( ginv( diag( object@ObsVar+object@LatVar ) - object@CoefMa
t ) )
        #一般化逆行列が計算できないなら強制終了
        if( class(ReducedForm)=="try-error" ) stop("Failed to compute inverse matrix:
 Maybe 'CoefMat' has some trouble.")
        #共分散行列の算出
        CovMat <- ReducedForm %*% object@CovMatResid %*% t(ReducedForm)
        #観測変数間の共分散行列を取り出しておく
        CovMatObs = CovMat[ c( (object@LatVar+1):(object@LatVar+object@ObsVar) ),
                                        c( (object@LatVar+1):(object@LatVar+objec
t@ObsVar) ) ]
        #type に応じて返り値を決定
        out <- switch(
            type,
            #all の場合は共分散行列全体
            all = CovMat,
            #l2l の場合は, 因子間の共分散行列
            l2l = CovMat[ c(1:object@LatVar), c(1:object@LatVar) ],
            #l2o の場合は, 因子から観測変数への共分散行列
            l2o = CovMat[ c( (object@LatVar+1):(object@LatVar+object@ObsVar) ), c(1:o
bject@LatVar) ],
            #o2l の場合は, 観測変数から因子への共分散行列
            o2l = CovMat[ c(1:object@LatVar), c( (object@LatVar+1):(object@LatVar+obj
ect@ObsVar) ) ],
            #o2o の場合は, 観測変数間の共分散行列
            o2o = CovMatObs
        )
        #観測変数間の共分散行列が NPD なら警告を出す
        if( any( eigen(CovMatObs)$values<=0 ) )
            warning("This model's covariance matrix among observed variables is not p
ositive definite.")
```

```
        return(out)
    }
)


##############################################
#クラス RamSpec のオブジェクトに対して単純構造ベースの因子分析モデル構造をセットする関
数 genSimpleFact()
#object: 対象オブジェクト
#factLoad: 因子負荷の値を発生させる命令文
#factCov: 因子間共分散の値を発生させる命令文
#obsErr: 観測変数の誤差分散の値を発生させる命令文
#add: 単純構造に追加するパスのパターンを指定する
##############################################
setGeneric("genSimpleFact",
    function(object, ...){
        value <- standardGeneric("genSimpleFact")
        return(value)
    }
)
setMethod("genSimpleFact", "RamSpec",
    function( object, factLoad, factCov, obsErr, add=c("none", "only1", "each1") ){
        ################################
        #適切な"RamSpec"オブジェクトかどうかをチェック
        ################################
        check <- validObject(object, test=TRUE)
        if( is.character(check) ) stop("Invalid 'RamSpec' object;", "\n", check)
        ################################
        #観測変数が因子数で割りきれない場合は却下
        ################################
        if( object@ObsVar %% object@LatVar !=0 ) stop("Invalid number of factors via
observed variables: 'ObsVar' must be dividable by 'LatVar'")
        ################################
        #add の形式チェック
        ################################
        #add が文字列ベクトルでない場合に警告する.
        if( !is.character(add) ) warning("'add' is coerced to be a character type obj
ect.")
        #add を文字列ベクトルに変換する.
        add <- as.character(add)
        #add 引数の中身を決定
        add <- match.arg(add)
        ################################
        #factLoad の形式チェック
        ################################
        ################################
        #factCov の形式チェック
        ################################
        ################################
        #obsErrVar の形式チェック
        ################################
        ################################
        #単純構造の因子負荷行列の生成
        ################################
        #単純構造の因子パタンを決定
        factorPattern <- sample( x=rep( 1:object@LatVar, each=object@ObsVar/object@La
tVar ), size=object@ObsVar, replace=FALSE)
```

```
        #因子パタンに基づいて，指定された乱数の値を持つ単純構造因子パタン行列を生成
        factLoadMat <- matrix(0, nrow=object@ObsVar, ncol=object@LatVar)
        for(i in 1:object@ObsVar) factLoadMat[ i, factorPattern[i] ] <- eval( parse(t
ext=factLoad) )
        ################################
        #因子負荷行列への追加パタンを生成: パターン"only1"の場合
        #パスをどこか 1 本だけ追加
        ################################
        if(add=="only1") factLoadMat[ sample( x=which(factLoadMat==0), size=1 ) ] <-
eval( parse(text=factLoad) )
        ################################
        #因子負荷行列への追加パタンを生成: パターン"each1"の場合
        #各因子ごとにパスを 1 本ずつ追加
        ################################
        if(add=="each1"){
            for(i in 1:object@LatVar) factLoadMat[,i][ sample( x=which(factLoadMat[,
i]==0), size=1 ) ] <- eval( parse(text=factLoad) )
        }
        ################################
        #因子の分散共分散行列の生成
        ################################
        factCovMat <- diag( object@LatVar )
        for(i in 1:( (object@LatVar^2 - object@LatVar)/2 ) ) factCovMat[ lower.tri(fa
ctCovMat) ][i] <- eval( parse(text=factCov) )
        factCovMat <- factCovMat + t(factCovMat)
        diag(factCovMat) <- diag(factCovMat)/2
        ################################
        #観測変数の残差分散行列の生成
        ################################
        obsErrMat <- matrix(0, nrow=object@ObsVar, ncol=object@ObsVar )
        for(i in 1:object@ObsVar) obsErrMat[i,i] <- eval( parse(text=obsErr) )
        ################################
        #クラス RamSpec のオブジェクトに対して値を付置
        ################################
        object <- setCoefMat(object, type="l2o", value=factLoadMat)
        object <- setCovMatResid(object, type="l2l", value=factCovMat)
        object <- setCovMatResid(object, type="o2o", value=obsErrMat)
        return(object)
    }
)


##############################################
#クラス RamSpec のオブジェクトに対してランダムベースの因子分析モデル構造をセットする関
数 genRandomFact()
#object: 対象オブジェクト
#onProb: 因子負荷が存在するかどうかのフラグを発生させる命令文
#factLoad: 因子負荷の値を発生させる命令文
#factCov: 因子間共分散の値を発生させる命令文
#obsErr: 観測変数の誤差分散の値を発生させる命令文
#keep: 最低限保ちたいパスのパターンを指定する
##############################################
setGeneric("genRandomFact",
    function(object, ...){
        value <- standardGeneric("genRandomFact")
        return(value)
    }
```

```
)
setMethod("genRandomFact", "RamSpec",
    function( object, onProb, factLoad, factCov, obsErr, keep=c("least1", "none", "ea
chAlone") ){
        ################################
        #適切な"RamSpec"オブジェクトかどうかをチェック
        ################################
        check <- validObject(object, test=TRUE)
        if( is.character(check) ) stop("Invalid 'RamSpec' object;", "\n", check)
        ################################
        #keep の形式チェック
        ################################
        #keep が文字列ベクトルでない場合に警告する.
        if( !is.character(keep) ) warning("'keep' is coerced to be a character type o
bject.")
        #keep を文字列ベクトルに変換する.
        keep <- as.character(keep)
        #keep 引数の中身を決定
        keep <- match.arg(keep)
        ################################
        #onProb の形式チェック
        ################################
        ################################
        #factLoad の形式チェック
        ################################
        ################################
        #factCov の形式チェック
        ################################
        ################################
        #obsErrVar の形式チェック
        ################################
        ################################
        #因子負荷行列の生成: パターン"none"の場合
        #完全にランダムで決定
        #因子を測定しない観測変数もあり得る
        ################################
        if(keep=="none"){
            #要素ごとに値が存在するかどうかを抽選
            factLoadMat <- matrix(0, nrow=object@ObsVar, ncol=object@LatVar)
            for( i in 1:(object@ObsVar*object@LatVar) ) factLoadMat[i] <- eval( parse
(text=onProb) ) * eval( parse(text=factLoad) )
        }
        ################################
        #因子負荷行列の生成: パターン"least1"の場合
        #完全にランダムで決定
        #ただし各観測変数は, 最低 1 つの因子を測定するように補正
        ################################
        if(keep=="least1"){
            factLoadMat <- matrix(0, nrow=object@ObsVar, ncol=object@LatVar)
            #要素ごとに値が存在するかどうかを抽選
            for( i in 1:(object@ObsVar*object@LatVar) ) factLoadMat[i] <- eval( parse
(text=onProb) ) * eval( parse(text=factLoad) )
            #値が 1 つも入ってない行には, どこか 1 列だけ補填する
            for(i in 1:object@ObsVar){
                if( any(factLoadMat[i,])==FALSE ) factLoadMat[ i, sample(x=1:object@L
atVar, size=1) ] <- eval( parse(text=factLoad) )
            }
        }
        ################################
        #因子負荷行列の生成: パターン"eachAlone"の場合
        #各因子につき 1 つづつ，その因子しか観測しない観測変数を確保
        #他はランダムで決定
        #ただし各観測変数は，最低でも 1 つの因子を測定する
        ################################
        if(keep=="eachAlone"){
            factLoadMat <- matrix(0, nrow=object@ObsVar, ncol=object@LatVar)
            #1 つの因子しか観測しない観測変数を決定
            alonePattern <- sample(x=1:object@ObsVar, size=object@LatVar)
            #単独測定する部分の因子負荷を決定
            for( i in 1:object@LatVar ) factLoadMat[ alonePattern[i], i ] <- eval( pa
rse(text=factLoad) )
            #それ以外の部分についてはランダムで決定
            for( i in setdiff( 1:(object@ObsVar), alonePattern ) ){
                for(j in 1:object@LatVar) factLoadMat[i,j] <- eval( parse(text=onPro
b) ) * eval( parse(text=factLoad) )
            }
            #値が 1 つも入ってない行には，どこか 1 列だけ補填する
            for( i in 1:object@ObsVar ){
                if( any(factLoadMat[i,])==FALSE ) factLoadMat[ i, sample(x=1:object@L
atVar, size=1) ] <- eval( parse(text=factLoad) )
            }
        }
        ################################
        #因子の分散共分散行列の生成
        ################################
        factCovMat <- diag( object@LatVar )
        for(i in 1:( (object@LatVar^2 - object@LatVar)/2 ) ) factCovMat[ lower.tri(fa
ctCovMat) ][i] <- eval( parse(text=factCov) )
        factCovMat <- factCovMat + t(factCovMat)
        diag(factCovMat) <- diag(factCovMat)/2
        ################################
        #観測変数の残差分散行列の生成
        ################################
        obsErrMat <- matrix(0, nrow=object@ObsVar, ncol=object@ObsVar )
        for(i in 1:object@ObsVar) obsErrMat[i,i] <- eval( parse(text=obsErr) )
        ################################
        #クラス RamSpec のオブジェクトに対して値を付置
        ################################
        object <- setCoefMat(object, type="l2o", value=factLoadMat)
        object <- setCovMatResid(object, type="l2l", value=factCovMat)
        object <- setCovMatResid(object, type="o2o", value=obsErrMat)
        return(object)
    }
)


##############################################
#クラス RamSpec のオブジェクトを元に発生したデータを含むクラス GSRam
##############################################
setClass(
    "GSRam",
    representation(
        SampleSize = "integer",       #標本数
        GenData = "data.frame",     #発生させたデータ
        GenCovMatVar = "matrix",       #発生させたデータの共分散行列
        CovMatVar = "matrix"                #真の構造の共分散行列
    ),
    contains="RamSpec"
)


##############################################
#RamSpec クラスのオブジェクトとして表現したモデルに基づく
#データを発生させてクラス GSRam のオブジェクトとして返す関数 genGSRam()
#object: モデル指定を含むクラス RamSpec のオブジェクト
#sample: サンプル数
#実質的には GSRam クラスオブジェクトの標準初期化関数に相当する
##############################################
setGeneric("genGSRam",
    function(object, ...){
        value <- standardGeneric("genGSRam")
        return(value)
    }
)
setMethod("genGSRam", "RamSpec",
    function( object, sample ){
        ################################
        #適切な"RamSpec"オブジェクトかどうかをチェック
        ################################
        check <- validObject(object, test=TRUE)
        if( is.character(check) ) stop("Invalid 'RamSpec' object;", "\n", check)
        ################################
        #sample の形式チェック
        ################################
        #sample が数値ベクトルでない場合に警告する.
        if( !is.numeric(sample) ) warning("'sample' is coerced to be a numeric type o
bject.")
        #sample を数値ベクトルに変換する.
        sample <- as.numeric(sample)
        #sample の長さが 2 以上の場合に警告する.
        if( length(sample) > 1 ) warning("'sample' contains more than one element. Us
ing the first one.")
        #sample の長さを 1 に合わせる. 長さが 0 の場合は NA が代入される.
        sample <- sample[1]
        #sample が有限の実数値でない場合, 強制終了する.
        if( !isTRUE( is.finite(sample)  ) ) stop("Invalid 'sample'; you must specify
finite real number value for 'sample'.")
        #sample が整数でない場合, 小数点以下は切り捨てる.
        if( !identical( sample, trunc(sample) ) ) {
            warning("'sample' is truncated toward 0.")
            sample <- trunc(sample)
        }
        #sample が 0 以下の場合, 強制終了する.
        if( sample <= 0 ) stop("Invalid 'sample'; you must specify positive integer v
alue for 'sample'.")
        #sample の代入値を決定
        Sample_inp <- as.integer( sample )
        ################################
        #真の共分散行列の計算とデータの発生
        ################################
        #モデルにおける観測変数間の共分散行列を計算する
        CovMatVar_inp <- getCovMatVar( object, type="o2o" )
        rownames(CovMatVar_inp) <- colnames(CovMatVar_inp) <- paste("v", 1:object@Obs
Var, sep="")
        #多変量正規分布に基づいてデータを発生
        GenData_inp <- mvrnorm( n=Sample_inp, mu=getMeanVec(object, type="obs"), Sigm
a=CovMatVar_inp )
        colnames(GenData_inp) <- paste("v", 1:object@ObsVar, sep="")
        GenData_inp <- data.frame(GenData_inp)
        #発生したサンプルの共分散行列を計算
        GenCovMatVar_inp <- cov(GenData_inp)
        ################################
        #クラス GSRam オブジェクトの生成
        ################################
        output <- new("GSRam",
                         SampleSize = Sample_inp,
                         GenData = GenData_inp,
                         GenCovMatVar = GenCovMatVar_inp,
                         CovMatVar = CovMatVar_inp,
                         object)
         return(output)
    }
)

##############################################
#クラス GSRam のオブジェクトのための閲覧関数 show.SGRam()
#object: 対象オブジェクト
##############################################
setMethod( "show", "GSRam",
    function(object){
        message("<Class 'GSRam' object including generated sample based on the model
specified in RAM format>", "\n")
        message("[Information about True Model Structure]")
        message("Number of Observed Variables @ Slot 'ObsVar':")
        print(object@ObsVar)
        message("\n")
        message("Number of Latent Variables @ Slot 'LatVar':")
        print(object@LatVar)
        message("\n")
        message("Mean Vector (alpha_zero) @ Slot 'MeanVec':")
        print(object@MeanVec)
        message("\n")
        message("Coefficient Matrix (Alpha) @ Slot 'CoefMat':")
        print(object@CoefMat)
        message("\n")
        message("Residual Covariance Matrix (Sigma_u) @ Slot 'CovMatResid':")
        print(object@CovMatResid)
        message("\n")
        message("Covariance Matrix of Observed Variables (Sigma_v) @ Slot 'CovMatVar
':")
        print(object@CovMatVar)
        message("\n")
        message("[Information about Generated Data]")
        message("Sample Size @ Slot 'SampleSize':")
        print(object@SampleSize)
        message("\n")
        message("Generated Data (Showing the First Part Only...) @ Slot 'GenData':")
        print( head(object@GenData) )
        message("\n")
        message("Covariance Matrix of Generated Data @ Slot 'GenCovMatVar':")
        print(object@GenCovMatVar)
        }
)

##############################################
#クラス GARam のオブジェクトのための要約関数 summary.RamSpec()
#object: 対象オブジェクト
##############################################
setMethod( "summary", "GSRam",
    function(object){
        check <- validObject(object, test=TRUE)
        message("Generated Sample Size: ", object@SampleSize)
        message("\n")
        message("Mean Vector of Generated Data:")
        mvgd <- matrix( mean(object@GenData), nrow=1)
        colnames(mvgd) <- paste("v", 1:object@ObsVar, sep="")
        print( mvgd )
        message("\n")
        message("Mean Vector of True Structure:")
        mvts <- t( as.matrix( getMeanVec(object, type="obs") ) )
        colnames(mvts) <- paste("v", 1:object@ObsVar, sep="")
        print( mvts )
        message("\n")
        message("Covariance Matrix of Generated Data:", "\n")
        print( object@GenCovMatVar )
        message("\n")
        message("Covariance Matrix of True Structure:")
        print(object@CovMatVar)
        #観測変数間の共分散行列が NPD なら警告を出す
        if( any( eigen(object@GenCovMatVar)$values<=0 ) )
            warning("Generated covariance matrix among observed variables is not posi
tive definite.")
        }
)

###############################################
#クラス GSRam オブジェクトのためのローレベルゲッター getSampleSize()
#object: 対象オブジェクト
##############################################
setGeneric("getSampleSize",
    function(object, ...){
        value <- standardGeneric("getSampleSize")
        return(value)
    }
)
setMethod("getSampleSize", "GSRam",
    function( object ){
    #適切な"RamSpec"オブジェクトかどうかをチェック
    check <- validObject(object, test=TRUE)
    if( is.character(check) ) stop("Invalid 'RamSpec' object;", "\n", check)
    #スロット SampleSize を返す
    return(object@SampleSize)
    }
)
```

```
###############################################
#クラス GSRam オブジェクトのためのローレベルゲッター getCovMatVar()
#object: 対象オブジェクト
##############################################
setMethod("getCovMatVar", "GSRam",
    function( object ){
    #適切な"RamSpec"オブジェクトかどうかをチェック
    check <- validObject(object, test=TRUE)
    if( is.character(check) ) stop("Invalid 'RamSpec' object;", "\n", check)
    #スロットを返す
    return(object@CovMatVar)
    }
)


###############################################
#クラス GSRam オブジェクトのためのローレベルゲッター getGenCovMatVar()
#object: 対象オブジェクト
##############################################
setGeneric("getGenCovMatVar",
    function(object, ...){
        value <- standardGeneric("getGenCovMatVar")
        return(value)
    }
)
setMethod("getGenCovMatVar", "GSRam",
    function( object ){
    #適切な"RamSpec"オブジェクトかどうかをチェック
    check <- validObject(object, test=TRUE)
    if( is.character(check) ) stop("Invalid 'RamSpec' object;", "\n", check)
    #スロットを返す
    return(object@GenCovMatVar)
    }
)


###############################################
#クラス GSRam オブジェクトのためのローレベルゲッター getGenData()
#object: 対象オブジェクト
##############################################
setGeneric("getGenData",
    function(object, ...){
        value <- standardGeneric("getGenData")
        return(value)
    }
)
setMethod("getGenData", "GSRam",
    function( object ){
    #適切な"RamSpec"オブジェクトかどうかをチェック
    check <- validObject(object, test=TRUE)
    if( is.character(check) ) stop("Invalid 'RamSpec' object;", "\n", check)
    #スロットを返す
    return(object@GenData)
    }
)
```

```
##############################################
#LABEL: validChromFormat(): クラス ChromFormat オブジェクトの形式チェックを行う関数
##############################################
validChromFormat <- function(object){
    ################################
    #スロット Length の形式チェック: integer は確定
    ################################
    #長さが 1 でなければならない
    if( length(object@Length)!=1 ) return("Slot 'Length' must have a single value.")
    #値が有限な実数でなければならない
    if( !isTRUE( all( is.finite(object@Length) ) ) ) return("Slot 'Length' must have
a finite real number value.")
    #値が 0 より大きくなければならない
    if( !isTRUE( object@Length>0) ) return("Slot 'Length' must have a positive value.
")
    ################################
    #スロット Type の形式チェック: factor は確定
    ################################
    #長さが Length に等しくなければならない
    if( length(object@Type)!=object@Length ) return("The length of Slot 'Type' must b
e as same as Slot 'Length'.")
    #値が有限でなければならない
    if( !isTRUE( all( is.finite(object@Type) ) ) ) return("Slot 'Type' must have a fi
nite value.")
    #factor の水準は"B", "M", "R"のいずれかでなければならない
    if( !all( levels(object@Type)%in%c("B","M","R") ) ) return("Slot 'Type' can have
only three levels; 'B', 'M', and 'R'.")
    ################################
    #スロット Min, Max の形式チェック: numeric は確定
    ################################
    #長さが Length に等しくなければならない
    if( length(object@Min)!=object@Length ) return("The length of Slot 'Min' must be
as same as Slot 'Length'.")
    if( length(object@Max)!=object@Length ) return("The length of Slot 'Max' must be
as same as Slot 'Length'.")
    #値に欠測値があってはならない
    if( any( is.na(object@Min) ) ) return("Slot 'Min' can't contain missing values.")
    if( any( is.na(object@Max) ) ) return("Slot 'Max' can't contain missing values.")
    for(i in 1:object@Length){
        #型式"B"の遺伝子は，Min=0, Max=1 に丸められる値でなければならない
        if( object@Type[i]=="B" ){
            if( trunc(object@Min[i])!=0 || trunc(object@Max[i])!=1 ) return("Binomial
 gene's Slot 'Min' should be zero and Slot 'Max' should be one.")
        }
        #型式"M"の遺伝子は，丸めたときの Min>Max かつ，両者は有限の値でなければならない
        if( object@Type[i]=="M" ){
            if( !isTRUE( all( is.finite(object@Min[i]) ) ) ) return("Multinomial gene
's Slot 'Min' should have finite values.")
            if( !isTRUE( all( is.finite(object@Max[i]) ) ) ) return("Multinomial gene
's Slot 'Max' should have finite values.")
            if( trunc(object@Min[i])>=trunc(object@Max[i]) ) return("Multinomial gene
's Slot 'Max' must have larger value than Slot 'Min'.")
        }
        #型式"R"の遺伝子は，Min>Max かつ，両者は有限の値でなければならないでなければな
らない
        if( object@Type[i]=="R" ){
            if( !isTRUE( all( is.finite(object@Min[i]) ) ) ) return("Real gene's Slot
 'Min' should have finite values.")
            if( !isTRUE( all( is.finite(object@Max[i]) ) ) ) return("Real gene's Slot
 'Max' should have finite values.")
            if( object@Min[i]>=object@Max[i] ) return("Real gene's Slot 'Max' must ha
ve larger value than Slot 'Min'.")
        }
    }
    return(TRUE)
}

##############################################
#LABEL: ChromFormat クラスの定義: 染色体の形式を表すクラス
##############################################
setClass(
    "ChromFormat",
    representation(
        Length = "integer",     #染色体の長さ
        Type = "factor",          #各遺伝子座ごとの値の型式
        Min = "numeric",         #各遺伝子座ごとの値の最小値
        Max = "numeric"         #各遺伝子座ごとの値の最大値
    ),
    validity=validChromFormat
)

##############################################
#LABEL: initChromFormat(): クラス ChromFormat オブジェクトの標準初期化関数
##############################################
initChromFormat <- function(type, ..., values, min, max){
    #type を文字列形式に変更.
    if( !is.character(type) ) warning("'type' is coerced to be a character type objec
t.")
    type <- as.character(type)
    #type の中身に対してマッチングを行い，想定外の値があるならば警告を出して削除
    type <- pmatch( x=type, table=c("binominal","multinomial","real"), duplicates.ok=
TRUE )
    if( any( is.na(type) )==TRUE ) warning("'type' involves one or more unexpected va
lues. They will be ignored.")
    type <- na.omit(type)
    #type の長さを元に，スロット'Length' への代入値を決定. 長さが 0 なら終了.
    Length_inp <- as.integer( length(type) )
    if(Length_inp==0) stop("'type' has no valid values. It have to be a charactor vec
tor including elements as follows; 'binomial', 'multinomial', and 'real'.")
    #type の中身を元に，スロット'Type' への代入値を決定.
    Type_inp <- cut( x=type, breaks=c(-Inf, 1.5, 2.5, Inf), labels=c("B","M","R") )
    #スロット'Min' への代入値を決定. タイプ B は 0，タイプ M は 1，タイプ R は精度内での-Inf に
近いところで決め打ち.
    Min_inp <- rep(NaN, Length_inp)
    for(i in 1:Length_inp) Min_inp[i] <- switch( Type_inp[i], B=0, M=1, R=-1*.Machine
$double.xmax )
    if( any( is.nan(Min_inp) )==TRUE ) stop("Exceptional error has occured. Maybe 'ty
pe' is invalid.")
    Min_inp <- as.numeric(Min_inp)
    #スロット'Max' への代入値を決定. タイプ B は 1，タイプ M は 99，タイプ R は精度内での Inf に
近いところで決め打ち.
    Max_inp <- rep(NaN, Length_inp)
    for(i in 1:Length_inp) Max_inp[i] <- switch( Type_inp[i], B=1, M=99, R=.Machine$d
ouble.xmax )
```

```
    if( any( is.nan(Max_inp) )==TRUE ) stop("Exceptional error has occured. Maybe 'ty
pe' is invalid.")
    Max_inp <- as.numeric(Max_inp)
    #クラス ChromFormat のオブジェクトを生成.
    object <- new("ChromFormat", Length=Length_inp, Type=Type_inp, Min=Min_inp, Max=M
ax_inp)
    #引数が指定されている場合は，setRange メソッドを実行して Min, Max を指定.
    call <- as.character()
    if( !missing(values) ) call <- paste( call, ",values=values", sep="" )
    if( !missing(min) ) call <- paste( call, ",min=min", sep="" )
    if( !missing(max) ) call <- paste( call, ",max=max", sep="" )
    if( length(call)!=0 ) call <- paste( "object <- setRange(object=object", call, ")
", sep="")
    eval( parse(text=call) )
    return(object)
}

###############################################
#LABEL: setRange(): クラス ChromFormat オブジェクトの Min, Max を指定する関数
###############################################
setGeneric("setRange",
    function(object, ...){
        value <- standardGeneric("setRange")
        return(value)
    }
)
setMethod("setRange", "ChromFormat",
    function( object, values, min, max ){
    #values, min, max のうち，どれか一つは与えられている必要がある.
    if( missing(values) && missing(min) && missing(max) ) stop("You must specify at l
east one argument of 'values', 'min', and 'max'.")
    ################################
    #引数'values' に関する処理
    ################################
    if( !missing(values) ){
        #values は行列形式でなければならない.
        if( !is.matrix(values) ) warning("'values' is coerced to be a matrix type obj
ect.")
        values <- as.matrix(values)
        #values が規定のサイズ（行数は最大で Length まで，列数は 3）では無い場合，終了す
る.
        if( nrow(values)>object@Length || ncol(values)!=3 ) stop("'values' should be
a matrix that has fewer number of rows than chromosom length and three columns.")
        #values の値を元に，スロット Min と Max への代入値を決定
        Min_inp <- object@Min
        Max_inp <- object@Max
        for(i in 1:nrow(values) ){
            if( !is.na(values[i,2]) ) Min_inp[ values[i,1] ] <- values[i,2]
            if( !is.na(values[i,3]) ) Max_inp[ values[i,1] ] <- values[i,3]
            }
        }
    ################################
    #引数'min' に関する処理
    ################################
    if( !missing(min) ){
        #'min' は'values' に優先する.
        if( !missing(values) ) warning("Take care that 'min' will have a priority ove
r 'values'.")
        #min が数値ベクトルでない場合に警告する.
        if( !is.numeric(min) ) warning("'min' is coerced to be a numeric type object.
")
        #min を数値ベクトルに変換する.
        min <- as.numeric(min)
        #min の長さが object@Length に合わない場合には終了.
        if( length(min)!=object@Length ) stop("The length of 'min' should be as same
as the chromosomal length.")
        #min の値を元に，スロット Min への代入値を決定
        Min_inp <- min
        }
    ################################
    #引数'max' に関する処理
    ################################
    if( !missing(max) ){
        #'max' は'values' に優先する.
        if( !missing(values) ) warning("Take care that 'max' will have a priority ove
r 'values'.")
        #max が数値ベクトルでない場合に警告する.
        if( !is.numeric(max) ) warning("'max' is coerced to be a numeric type object.
")
        #max を数値ベクトルに変換する.
        max <- as.numeric(max)
        #max の長さが object@Length に合わない場合には終了.
        if( length(max)!=object@Length ) stop("The length of 'max' should be as same
as the chromosomal length.")
        #max の値を元に，スロット Max への代入値を決定
        Max_inp <- max
        }
    #スロット Min と Max の値を書き換える.
    if( exists("Min_inp") ) object@Min <- Min_inp
    if( exists("Max_inp") ) object@Max <- Max_inp
    #書き換えた結果が不適切ならば，エラーを返す.
    check <- validObject(object, test=TRUE)
    if( is.character(check) ) stop("The result would be an invalid 'ChromFormat' obje
ct;", "\n", check)
    return(object)
    }
)

##############################################
#LABEL: getLength(): クラス ChromFormat オブジェクトのためのローレベルゲッター
#object: 対象オブジェクト
##############################################
setGeneric("getLength",
    function(object, ...){
        value <- standardGeneric("getLength")
        return(value)
    }
)
setMethod("getLength", "ChromFormat",
    function( object ){
        #適切な"ChromFormat"オブジェクトかどうかをチェック
        check <- validObject(object, test=TRUE)
        if( is.character(check) ) stop("Invalid 'ChromFormat' object;", "\n", check)
        #スロットの値を返す
        return(object@Length)
    }
)

##############################################
#LABEL: getType(): クラス ChromFormat オブジェクトのためのローレベルゲッター
#object: 対象オブジェクト
##############################################
setGeneric("getType",
    function(object, ...){
        value <- standardGeneric("getType")
        return(value)
    }
)
setMethod("getType", "ChromFormat",
    function( object ){
        #適切な"ChromFormat"オブジェクトかどうかをチェック
        check <- validObject(object, test=TRUE)
        if( is.character(check) ) stop("Invalid 'ChromFormat' object;", "\n", check)
        #スロットの値を返す
        return(object@Type)
    }
)

##############################################
#LABEL: getMin(): クラス ChromFormat オブジェクトのためのローレベルゲッター
#object: 対象オブジェクト
##############################################
setGeneric("getMin",
    function(object, ...){
        value <- standardGeneric("getMin")
        return(value)
    }
)
setMethod("getMin", "ChromFormat",
    function( object ){
        #適切な"ChromFormat"オブジェクトかどうかをチェック
        check <- validObject(object, test=TRUE)
        if( is.character(check) ) stop("Invalid 'ChromFormat' object;", "\n", check)
        #スロットの値を返す
        return(object@Min)
    }
)

##############################################
#LABEL: getMax(): クラス ChromFormat オブジェクトのためのローレベルゲッター
#object: 対象オブジェクト
##############################################
setGeneric("getMax",
    function(object, ...){
        value <- standardGeneric("getMax")
        return(value)
    }
)
setMethod("getMax", "ChromFormat",
    function( object ){
        #適切な"ChromFormat"オブジェクトかどうかをチェック
        check <- validObject(object, test=TRUE)
        if( is.character(check) ) stop("Invalid 'ChromFormat' object;", "\n", check)
        #スロットの値を返す
        return(object@Max)
    }
)

##############################################
#LABEL: show.ChromFormat(): クラス ChromFormat オブジェクトのための閲覧関数
#object: 対象オブジェクト
##############################################
setMethod( "show", "ChromFormat",
    function(object){
        message("<Class 'ChromFormat' object to represent the format of chromosome us
ed in package 'GAAssem'.>", "\n")
        message("Length of Chromosome @ Slot 'Length':")
        print(object@Length)
        message("\n")
        message("Format of Chromosome @ Slot 'Type' ('B' is for binomial, 'M' is for
multinomial, and 'R' is for real.):")
        print(object@Type)
        message("\n")
        message("Minimum Value of Each Gene @ Slot 'Min':")
        print(object@Min)
        message("\n")
        message("Maximum Value of Each Gene @ Slot 'Max':")
        print(object@Max)
        }
)

##############################################
#LABEL: summary.ChromFormat(): クラス ChromFormat オブジェクトのための要約関数
#object: 対象オブジェクト
##############################################
setMethod( "summary", "ChromFormat",
    function(object){
        message("Chromosome Format:")
        print(data.frame(Type=object@Type, MinimumValue=object@Min, MaxValue=object@M
ax))
    }
)

##############################################
#LABEL: genIndividual(): クラス ChromFormat オブジェクトに従う染色体をランダムに発生さ
せる関数
#object: クラス ChromFormat のオブジェクト
##############################################
setGeneric("genIndividual",
    function(object, ...){
        value <- standardGeneric("genIndividual")
        return(value)
    }
)
setMethod("genIndividual", "ChromFormat",
    function( object ){
        #適切な"ChromFormat"オブジェクトかどうかをチェック.
        check <- validObject(object, test=TRUE)
        if( is.character(check) ) stop("Invalid 'ChromFormat' object;", "\n", check)
        #遺伝子ごとの型式に応じた値を発生させる.
        output <- numeric(object@Length)
        for(i in 1:object@Length) {
            output[i] <- switch(object@Type[i],
                                    "B"=rbinom( n=1, size=1, prob=0.5),
                                    "M"=sample( x=trunc(object@Min[i]):trunc(object
@Max[i]), size=1),
                                    "R"=runif( n=1, min=object@Min[i], max=object@M
ax[i] ) )
        }
        return(output)
    }
)


##############################################
#LABEL: GAEval クラスの定義: GA の演算を管理するクラス
##############################################
setClass(
    "GAEval",
    representation(
        Generation = "integer",             #世代数
        Population = "matrix",               #生存している個体
        Fitness = "numeric",                  #個体の適合度
        History = "list",                          #探索履歴
        MatingPool = "matrix",              #繁殖用個体の保存場所
        MatingFitness = "numeric"                  #繁殖用個体の適合度
    ),
    contains="ChromFormat"
)

##############################################
#LABEL: show.GAEval(): クラス GAEval オブジェクトのための閲覧関数
#object: 対象オブジェクト
##############################################
setMethod( "show", "GAEval",
    function(object){
        message("<Class 'GAEval' object; UNDER CONSTRUCTION.>", "\n")
        message("Length of Chromosome @ Slot 'Length':")
        print(object@Length)
        message("\n")
        message("Format of Chromosome @ Slot 'Type' ('B' is for binomial, 'M' is for
multinomial, and 'R' is for real.):")
        print(object@Type)
        message("\n")
        message("Minimum Value of Each Gene @ Slot 'Min':")
        print(object@Min)
        message("\n")
        message("Maximum Value of Each Gene @ Slot 'Max':")
        print(object@Max)
        message("\n")
        message("Slot 'Generation':")
        print(object@Generation)
        message("\n")
        message("Slot 'Population':")
        print(object@Population)
        message("\n")
        message("Slot 'Fitness':")
        print(object@Fitness)
        message("\n")
        message("Slot 'History':")
        print(object@History)
        message("\n")
        message("Slot 'MatingPool':")
        print(object@MatingPool)
        message("\n")
        message("Slot 'MatingFitness':")
        print(object@MatingFitness)
        }
)

##############################################
#LABEL: initGAEval(): クラス GAEval オブジェクトの標準初期化関数
##############################################
initGAEval <- function(object, data, population, genFunc, genFuncOp, evalFunc, evalFu
ncOp, ...){
    #初期個体の器
    Population_inp <- matrix(0, nrow=population, ncol=object@Length)
    #個体発生コマンドの命令文復元
    if( missing(genFuncOp) ) {
        #オプション無しの場合, object のみを引数とする
        genCom <- paste(genFunc, "(object)", sep="")
    } else{
        #オプション有りの場合, genFuncOp を追加する
        optionGen <- character(0)
        for(i in 1:length(genFuncOp)) optionGen <- paste(optionGen, genFuncOp[i], sep
=",")
        genCom <- paste(genFunc, "(object", optionGen, ")", sep="")
    }
#
#efa <- factanal( covmat=mydata@GenCovMatVar, factors=因子, n.obs=サンプル数, rotatio
n="none")
#efaR <- promax( loadings(efa), m=2 )
#efaPattern <- loadings(efaR)[1:観測変数, 1:因子]
#inpPattern <- matrix(0, nrow=観測変数, ncol=因子)
#inpPattern[ which(abs(efaPattern)>0.3) ] <- 1
#inpPattern <- list( Loadings=inpPattern, Covariance=matrix(1,nrow=因子, ncol=因子) )
#inpChrom <- encodeCELFA(inpPattern)
#inpChrom <- reorderCELFA(inpChrom, nof=因子, nov=観測変数)
    #復元したコマンドを使って個体を発生
#    if( chromValidCELFA(inpChrom, nof=因子, nov=観測変数)==TRUE ){
    for(i in 1:(population-5)) {Population_inp[i,] <- eval( parse( text=genCom ) ) }
    Population_inp[(population-4):population,] <- matrix(rep(c(1,7,13,19,20,1,0,0,0,0,
1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,
0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0),5),nrow=5,
byrow=T)
#    } else{
#    for(i in 1:population) Population_inp[i,] <- eval( parse( text=genCom ) )
#    }
    #適合度計算コマンドの命令文復元
    Fitness_inp <- numeric(population)
    for(i in 1:population){
        if( missing(evalFuncOp) ) {
            #オプション無しの場合，染色体のみを引数とする
            evalCom <- paste(evalFunc, "(", "Population_inp[", i,",]", ")", sep="")
        } else{
            #オプション有りの場合，evalFuncOp を追加する
            optionEval <- character(0)
            for(j in 1:length(evalFuncOp)) optionEval <- paste(optionEval, evalFuncOp
[j], sep=",")
            evalCom <- paste(evalFunc, "(", "Population_inp[", i,",]", optionEval, ")
", sep="")
        }
    gc()
     Fitness_inp[i] <- eval( parse( text=evalCom ) )
     gc()
     }
    #適合度順に並び替えて履歴を記録
    Population_inp <- Population_inp[order(Fitness_inp),]
    Fitness_inp <- Fitness_inp[order(Fitness_inp)]
    History_inp <- as.list(NA)
    #オブジェクトの生成
    output <- new("GAEval", Generation=as.integer(1), Population=Population_inp, Fitn
ess=Fitness_inp, History=History_inp, object)
    output@History[[1]] <- list(Generation=output@Generation, Population=output@Popul
ation, Fitness=output@Fitness)
    return(output)
}

# scan の出力を抑制するバージョン
specify.model2 <- function (file = "")
{
    ram <- scan(file = file, what = list(path = "", par = "",
        start = 1, dump = ""), sep = ",", strip.white = TRUE,
        comment.char = "#", fill = TRUE, quiet = TRUE)
    ram <- cbind(ram$path, ram$par, ram$start)
    class(ram) <- "mod"
    ram
}

###############################################
#LABEL: genIndCELFA(): CELFA において初期個体を生成するための関数
#object: クラス ChromFormat のオブジェクト
###############################################
setGeneric("genIndCELFA",
    function(object, ...){
        value <- standardGeneric("genIndCELFA")
        return(value)
    }
)
setMethod("genIndCELFA", "ChromFormat",
    function( object ){
        #適切な"ChromFormat"オブジェクトかどうかをチェック.
        check <- validObject(object, test=TRUE)
        if( is.character(check) ) stop("Invalid 'ChromFormat' object;", "\n", check)
        #マジックナンバー算出
        multinomPart <- which(object@Type=="M")
        nof <- length(multinomPart)
        binomPart <- which(object@Type=="B")
        nov <- ( object@Length-nof-(nof^2-nof)*0.5 ) / nof + nof
        #染色体の容器を生成
        indivChrom <- numeric(object@Length)
        #多変量分布部については、値が重複しないようにまとめて生成
        indivChrom[multinomPart] <- sample( x=trunc(object@Min[1]):trunc(object@Max
[1]), size=nof, replace=FALSE )
        #二項分布部もまとめて生成
        indivChrom[binomPart] <- rbinom( n=length(binomPart), size=1, prob=0.5)
        #染色体の並び替え
        indivChrom <- reorderCELFA(indivChrom, nof, nov)
        #染色体の適切性チェック
        while( chromValidCELFA(indivChrom, nof, nov)==FALSE ) {
            indivChrom[multinomPart] <- sample( x=trunc(object@Min[1]):trunc(object@M
ax[1]), size=nof, replace=FALSE )
            indivChrom[binomPart] <- rbinom( n=length(binomPart), size=1, prob=0.5)
            indivChrom <- reorderCELFA(indivChrom, nof, nov)
        }
        return(indivChrom)
    }
)

##############################################
#LABEL: decodeCELFA(): CELFA において染色体を 0/1 のパタン行列関数に変換する関数
#chromosome: 染色体
#nof: 因子の数
#nov: 観測変数の数
##############################################
decodeCELFA <- function(chromosome, nof, nov){
    #出力の器を生成
    factLoadPat <- matrix(0, nrow=nov, ncol=nof)
    factCovPat <- diag(nof)
    #最初の m 個分の因子パタンを配置
    for(i in 1:nof) factLoadPat[ chromosome[i], i ] <- 1
    #残りの空いている行番号リストを作成
    remain <- setdiff(1:nov,chromosome[1:nof])
    #代入すべきパタンを行列に起こす
    temp <- matrix( chromosome[(nof+1):(nof+(nov-nof)*nof)], nrow=(nov-nof), ncol=nof,
byrow=TRUE )
    #因子パタンを全て配置
    for(i in 1:(nov-nof)) factLoadPat[ remain[i], ] <- temp[i,]
    #因子間共分散のパタンを配置, 下対角に入れてから転写
    factCovPat[lower.tri(factCovPat)] <- tail(chromosome, (nof^2-nof)*0.5)
    factCovPat <- factCovPat + t(factCovPat)
    diag(factCovPat) <- diag(factCovPat)*0.5
    #結果をリストにして返す
    output <- list( Loadings=factLoadPat, Covariance=factCovPat )
    return(output)
}

##############################################
#LABEL: pReorderCELFA(): CELFA においてパタン行列の並べ替えを行う関数
#pattern: 因子負荷行列および因子間共分散行列を 0/1 でパタン化したリスト
##############################################
pReorderCELFA <- function(pattern){
    #マジックナンバー算出
    nof <- ncol(pattern$Loadings)
    nov <- nrow(pattern$Loadings)
    #作業用パタン行列を作成
    keepPattern <- pattern$Loadings
    #列ラベルのマスターと，それを元にした作業用列ラベルを作成
    keepLabel <- masterLabel <- 1:nof
    #最終的な順番を出力する器を生成
    totalOrder <- numeric(0)
    #行カウンタの初期化
    i <- 1
    #因子数-1 列まではループで処理（一列になると行列構造を維持できないため）
    while( length(totalOrder)<(nof-1) ){
        #i 行目において値が 1 である（最大である）列の番号を取得
        keepLabel <- keepLabel[ which( rank(-1*keepPattern[i,], ties="min")==1 ) ]
        #もしそれが 1 列しかないならば
        if(length(keepLabel)==1) {
            #その列は左側へ持ってくる
            totalOrder <- append(totalOrder, keepLabel[1])
            #移動済みの列は探索対象から外す
            keepPattern <- pattern$Loadings[,-totalOrder]
            keepLabel <- masterLabel[-totalOrder]
            #行カウンタの初期化
            i <- 1
        } else{
            #もし 2 列以上において値が 1 であるならば，それらだけをキープして
            keepPattern <- keepPattern[ ,which( rank(-1*keepPattern[i,], ties="min")=
=1 ) ]
            #探索行を一つ下へ
            i <- i+1
        }
    }
    #最後の一列分を列リストに追加
    totalOrder <- append( totalOrder, keepLabel[1] )
    #因子負荷行列パタンの並び替え
    neoLoadings <- pattern$Loadings[,totalOrder]
    #因子間共分散行列パタンの並び替え
    neoCovariance <- matrix(0, nrow=nof, ncol=nof)
    for(i in 1:nof){
        for(j in 1:nof){
            neoCovariance[i,j] <- pattern$Covariance[ totalOrder[i], totalOrder[j] ]
        }
    }
    #結果を返して終了
    output <- list( Loadings=neoLoadings, Covariance=neoCovariance )
    return(output)
}

##############################################
#LABEL: encodeCELFA(): CELFA においてパタン行列を染色体形式に変換する関数
#pattern: 因子負荷行列および因子間共分散行列を 0/1 でパタン化したリスト
##############################################
encodeCELFA <- function(pattern){
    #因子負荷行列のパタンにおいて 1 因子しか測定していない観測変数の番号を抽出
    targetList <- which( rowSums(pattern$Loadings)==1 )
    #上のリストのうち，各因子を観測する一番若い観測変数の番号を探索
    forSimple <- numeric( ncol(pattern$Loadings) )
    for(i in rev(targetList) ){
        forSimple[ which( pattern$Loadings[i,]==1 ) ] <- i
    }
    #最初の三つは上のリスト，因子パタンの残りの要素，因子間共分散パタンの下三角を結合
```

```
して染色体形式へ
    output <- c( forSimple, as.vector( t(pattern$Loadings[-forSimple,]) ), pattern$Co
variance[ lower.tri(pattern$Covariance) ] )
    return(output)
}

##############################################
#LABEL: reorderCELFA(): CELFA においてパタン行列の並べ替えを行う関数
#chromosome: 染色体
#nof: 因子の数
#nov: 観測変数の数
##############################################
reorderCELFA <- function(chromosome, nof, nov){
    pattern <- decodeCELFA(chromosome, nof, nov)
    pattern <- pReorderCELFA(pattern)
    out <- encodeCELFA(pattern)
    return(out)
}

##############################################
#LABEL: chromValidCELFA(): CELFA において染色体が形式的に正しいかどうかを判定する関数
#chromosome: 染色体
#nof: 因子の数
#nov: 観測変数の数
##############################################
chromValidCELFA <- function(chromosome, nof, nov){
    pattern <- decodeCELFA(chromosome, nof, nov)
    #一つも因子を観測しない観測変数がある場合，アウト
    if( any( apply(pattern$Loadings,1,sum)<=0 ) ) return(FALSE)
    #一つの因子に付き二つ以上の観測変数がない場合，アウト
    if( any( apply(pattern$Loadings,2,sum)<=1 ) ) return(FALSE)
    return(TRUE)
}

##############################################
#LABEL: writeSemSpec(): パタン行列を specify.model 関数用のテキストに変換する関数
#pattern: 因子負荷行列および因子間共分散行列を 0/1 でパタン化したリスト
##############################################
writeSemSpec <- function(pattern){
    #マジックナンバー算出
    nof <- ncol(pattern$Loadings)
    nov <- nrow(pattern$Loadings)
    #因子パタン部の書き出し
    factLoad <- character(0)
    for(i in 1:nov){
        for(j in 1:nof){
            if(pattern$Loadings[i,j]==1) factLoad <- append(factLoad, paste( "F", j,
" -> V", i, ", alpha_", i, j, ", NA", sep="" ) )
        }
    }
    #観測変数の誤差分散
    obsVar <- paste( "V", 1:nov, " <-> ", "V", 1:nov, ", Delta_", 1:nov, ", NA", sep=
"")
    #因子の分散
    factVar <- paste( "F", 1:nof, " <-> ", "F", 1:nof, ", NA", ", ", 1, sep="")
    #因子間共分散
    rail <- which( upper.tri(pattern$Cov)==TRUE, arr.ind=TRUE)
    factCov <- character(0)
    for(i in 1:nrow(rail)){
        if(pattern$Covariance[ rail[i,1], rail[i,2] ]==1) factCov <- append( factCov,
 paste( "F", rail[i,1], " <-> F", rail[i,2], ", corr_", rail[i,1], rail[i,2], ", NA",
 sep="" ) )
    }
    #統合してファイルへ書き出し
    specification <- matrix(c(factLoad, obsVar, factVar, factCov), ncol=1)
    write(specification, "specification_temp.txt")
}

##############################################
#LABEL: evalFitCELFA(): 染色体に対応するモデルの RMSEA を計算する関数
#chromosome: 染色体
#dataobject:分析する共分散行列
##############################################
evalFitCELFA <- function(chromosome, dataobject){
    nof<-getLatVar(dataobject)
    nov<-getObsVar(dataobject)
    data<-getGenCovMatVar(dataobject)
    sample<-getSampleSize(dataobject)
    pattern <- decodeCELFA(chromosome, nof, nov)
    writeSemSpec(pattern)
    model <- specify.model2(file="specification_temp.txt")
    estimation <- try( sem(ram=model, S=data, N=sample) )
    rmsea <- 9999
    if( class(estimation)=="try-error" ) return(999)
    if( estimation$convergence>=4 ) return(99)
    if( estimation$convergence<=3 & is.null(estimation$cov)==TRUE ) return(9)
    if( estimation$convergence<=3 & is.null(estimation$cov)==FALSE ) rmsea <- sqrt( m
ax( estimation$criterion / ( 0.5 * estimation$n * ( estimation$n + 1 ) - estimation$t
 ) - 1 / (estimation$N - 1), 0) )
    return(rmsea)
}

##############################################
#LABEL: evalFitCELFA2(): 染色体に対応するモデルの RMSEA を計算する関数
#chromosome: 染色体
#dataobject:分析する共分散行列
##############################################
evalFitCELFA2 <- function(chromosome, dataobject){
    nof<-getLatVar(dataobject)
    nov<-getObsVar(dataobject)
    data<-getGenCovMatVar(dataobject)
    sample<-getSampleSize(dataobject)
    pattern <- decodeCELFA(chromosome, nof, nov)
    writeSemSpec(pattern)
    model <- specify.model2(file="specification_temp.txt")
    estimation <- try( sem(ram=model, S=data, N=sample) )
    return(estimation)
}


##############################################
#LABEL: selectCELFA(): CELFA において繁殖個体を選択するための関数
#object: クラス GAEval のオブジェクト
##############################################
```

```
setGeneric("selectCELFA",
    function(object, ...){
        value <- standardGeneric("selectCELFA")
        return(value)
    }
)
setMethod("selectCELFA", "GAEval",
    function( object ){
     #適合度の逆数を選択比率にしてルーレット選択
    ratio <- 1/object@Fitness
    #ただし RMSEA=0 の場合は 1e+99 に
    ratio <- replace(ratio, is.infinite(ratio), 1e+99)
    #親になる 2 個体を選択して
    choice <- sample(1:length(ratio), 2, prob=ratio)
    #染色体を返す
    output <- object@Population[choice,]
    }
)

##############################################
#LABEL: crossCELFA(): CELFA において交叉を行うための関数
#chromosomes: 両親の染色体を含む 2 行の行列
#nof:因子の数
##############################################
crossCELFA <-  function( chromosomes, nof ){
    #三つの部分に染色体を分割
    part1 <- 1:nof
    part3 <- rev( seq( from=ncol(chromosomes), length=(nof^2-nof)/2, by=-1) )
    part2 <- setdiff( 1:ncol(chromosomes), c(part1,part3) )
    #単体負荷部分は部分写像交叉
    child1 <- matrix( 0, nrow=2, ncol=length(part1) )
    cut1 <- sample( 0:length(part1), 1)
    if( cut1==0 ) {
        child1[1,] <- chromosomes[2,part1]
        child1[2,] <- chromosomes[1,part1]
    }else if( cut1==length(part1) ) {
        child1[1,] <- chromosomes[1,part1]
        child1[2,] <- chromosomes[2,part1]
    }else{
        former1 <- chromosomes[,part1][,1:cut1,drop=FALSE]
        latter1 <- chromosomes[,part1][,-(1:cut1),drop=FALSE]
        child1[1,-(1:cut1)] <- latter1[2,]
        for( i in 1:cut1 ){
            if( is.element( former1[1,i], child1[1,-(1:cut1)] )==TRUE ){
            child1[1,i] <- latter1[1,][ which( child1[1,-(1:cut1)]==former1[1,i] ) ]
            while( is.element( child1[1,i], child1[1,-(1:cut1)] )==TRUE ) child1[1,i]
 <- latter1[1,][ which( child1[1,-(1:cut1)]==child1[1,i] ) ]
            } else{
                child1[1,i] <- former1[1,i]
            }
        }
        child1[2,-(1:cut1)] <- latter1[1,]
        for( i in 1:cut1 ){
            if( is.element( former1[2,i], child1[2,-(1:cut1)] )==TRUE ){
            child1[2,i] <- latter1[2,][ which( child1[2,-(1:cut1)]==former1[2,i] ) ]
            while( is.element( child1[2,i], child1[2,-(1:cut1)] )==TRUE ) child1[2,i]
 <- latter1[2,][ which( child1[2,-(1:cut1)]==child1[2,i] ) ]
```

```
            } else{
                child1[2,i] <- former1[2,i]
            }
        }
    }
    #その他の因子負荷部はランダム一点交叉
    child2 <- matrix(0, nrow=2, ncol=length(part2))
    cut2 <- sample( 0:length(part2), 1)
    if( cut2==0 ) {
        child2[1,] <- chromosomes[2,part2]
        child2[2,] <- chromosomes[1,part2]
    }else if( cut2==length(part2) ) {
        child2[1,] <- chromosomes[1,part2]
        child2[2,] <- chromosomes[2,part2]
    }else{
        former2 <- chromosomes[,part2][,1:cut2, drop=FALSE]
        latter2 <- chromosomes[,part2][,-(1:cut2), drop=FALSE]
        child2[1,1:cut2] <- former2[1,]
        child2[1,-(1:cut2)] <- latter2[2,]
        child2[2,1:cut2] <- former2[2,]
        child2[2,-(1:cut2)] <- latter2[1,]
    }
    #因子間相関部もランダム一点交叉
    child3 <- matrix(0, nrow=2, ncol=length(part3))
    cut3 <- sample( 0:length(part3), 1)
    if( cut3==0 ) {
        child3[1,] <- chromosomes[2,part3]
        child3[2,] <- chromosomes[1,part3]
    }else if( cut3==length(part3) ) {
        child3[1,] <- chromosomes[1,part3]
        child3[2,] <- chromosomes[2,part3]
    }else{
        former3 <- chromosomes[,part3][,1:cut3, drop=FALSE]
        latter3 <- chromosomes[,part3][,-(1:cut3), drop=FALSE]
        child3[1,1:cut3] <- former3[1,]
        child3[1,-(1:cut3)] <- latter3[2,]
        child3[2,1:cut3] <- former3[2,]
        child3[2,-(1:cut3)] <- latter3[1,]
    }
    #結果の書き出し
    out <- cbind(child1,child2,child3)
    return(out)
}

##############################################
#LABEL: mutateCELFA(): CELFA において交叉を行うための関数
#chromosomes: 子供の染色体を含む 2 行の行列
#rate:突然変異率
#nof:因子数
#nov:観測変数
##############################################
mutateCELFA <- function(chromosomes, rate, nof, nov){
    #2 項乱数により突然変異が起きるかどうかを判定
    flag <- matrix(0, nrow=2, ncol=ncol(chromosomes))
    flag[1,] <- rbinom( n=ncol(chromosomes), size=1, prob=rate)
    flag[2,] <- rbinom( n=ncol(chromosomes), size=1, prob=rate)
    #出力のための準備
    out <- chromosomes
    part1 <- 1:nof
    part3 <- rev( seq( from=ncol(chromosomes), length=(nof^2-nof)/2, by=-1) )
    part2 <- setdiff( 1:ncol(chromosomes), c(part1,part3) )
    #変異フラグが立っている場所は遺伝子の値を変更
    for(i in 1:2){
        #単純構造部は重複しないように
        for(j in part1) if(flag[i,j]==1) out[i,j] <- sample( setdiff(1:nov, out[i,par
t1]), 1 )
        #残りはビット反転
        for(j in part2) if(flag[i,j]==1) out[i,j] <- setdiff(0:1, chromosomes[i,j])
        for(j in part3) if(flag[i,j]==1) out[i,j] <- setdiff(0:1, chromosomes[i,j])
    }
    return(out)
}

##############################################
#LABEL: localCELFA(): CELFA において手当たり次第局所探索を行うための関数
#object: クラス GAEval のオブジェクト
#dataobject:クラス GSRam のオブジェクト
#position:計算対象が何番目の染色体か
##############################################
localCELFA <- function(object, dataobject, position){
    #マジックナンバー算出
    nof <- dataobject@LatVar
    nov <- dataobject@ObsVar
    #元となる染色体と適合度を取得
    baseChrom <- object@MatingPool[position,]
    baseFit <- object@MatingFitness[position]
    #不適解となる個体なら計算しない
    if(baseFit>=1) return(object)
    #二値遺伝子部分について 1 フリップ近傍を列挙，リオーダーする
    locals1 <- matrix(0, nrow=length( (nof+1):object@Length), ncol=object@Length )
    for(i in (nof+1):object@Length){
        newChrom <- baseChrom
        if(baseChrom[i]==0) newChrom[i] <- 1
        if(baseChrom[i]==1) newChrom[i] <- 0
        locals1[(i-nof),] <- reorderCELFA( newChrom, nof=nof, nov=nov )
    }
    #
    locals2 <- matrix(0, nrow=(nov-nof)*nof, ncol=object@Length )
    candy <- setdiff(1:nov, baseChrom[1:nof])
    ct <- 1
       for(i in 1:nof){
           for(j in 1:length(candy) ){
               newChrom <- baseChrom
               newChrom[i] <- candy[j]
               locals2[ct,] <- reorderCELFA( newChrom, nof=nof, nov=nov )
               ct <- ct+1
           }
    }
    locals <- rbind(locals1, locals2)
    locals <- locals[sample(1:nrow(locals)),]
    #染色体が計算可能な形ならば適合度を求める，不適解なら 999
    for(i in 1:nrow(locals) ) {
        if( chromValidCELFA(locals[i,], nof=nof, nov=nov)==TRUE ){
            gc()
            locFit <- evalFitCELFA( locals[i,], dataobject)
        } else{
            locFit <- 999
        }
    #適合度が元よりも改善されているならば，それで置き換える
    if( locFit<baseFit ) {
        object@MatingPool[position,] <- locals[i,]
        object@MatingFitness[position] <- locFit
        return(object)
        }
    }
    return(object)
}

##############################################
#LABEL: localCELFA2(): CELFA において手当たり次第局所探索を行うための関数
#object: クラス GAEval のオブジェクト
#dataobject:クラス GSRam のオブジェクト
#position:計算対象が何番目の染色体か
##############################################
localCELFA2 <- function(object, dataobject, position){
    #マジックナンバー算出
    nof <- dataobject@LatVar
    nov <- dataobject@ObsVar
    #元となる染色体と適合度を取得
    baseChrom <- object@Population[position,]
    baseFit <- object@Fitness[position]
    #不適解となる個体なら計算しない
    if(baseFit>=1) return(object)
    #1 フリップ近傍を列挙，リオーダーする
    locals1 <- matrix(0, nrow=length( (nof+1):object@Length), ncol=object@Length )
    for(i in (nof+1):object@Length){
        newChrom <- baseChrom
        if(baseChrom[i]==0) newChrom[i] <- 1
        if(baseChrom[i]==1) newChrom[i] <- 0
        locals1[(i-nof),] <- reorderCELFA( newChrom, nof=nof, nov=nov )
    }
    #
    locals2 <- matrix(0, nrow=(nov-nof)*nof, ncol=object@Length )
    candy <- setdiff(1:nov, baseChrom[1:nof])
    ct <- 1
       for(i in 1:nof){
           for(j in 1:length(candy) ){
               newChrom <- baseChrom
               newChrom[i] <- candy[j]
               locals2[ct,] <- reorderCELFA( newChrom, nof=nof, nov=nov )
               ct <- ct+1
           }
    }
    locals <- rbind(locals1, locals2)
    locals <- locals[sample(1:nrow(locals)),]
    #染色体が計算可能な形ならば適合度を求める，不適解なら 999
    for(i in 1:nrow(locals) ) {
        if( chromValidCELFA(locals[i,], nof=nof, nov=nov)==TRUE ){
            gc()
            locFit <- evalFitCELFA( locals[i,], dataobject)
        } else{
            locFit <- 999
        }
    #適合度が元よりも改善されているならば，それで置き換える
    if( locFit<baseFit ) {
        object@Population[position,] <- locals[i,]
        object@Fitness[position] <- locFit
        return(object)
        }
    }
    return(object)
}


##############################################
#LABEL: localModCELFA(): CELFA において ModInd による局所探索を行うための関数
#object: クラス GAEval のオブジェクト
#dataobject:クラス GSRam のオブジェクト
#position:計算対象が何番目の染色体か
##############################################
localModCELFA <- function(object, dataobject, position){
    #マジックナンバー算出
    nof <- dataobject@LatVar
    nov <- dataobject@ObsVar
    #元となる染色体と適合度を取得
    baseChrom <- main@MatingPool[position,]
    baseFit <- main@MatingFitness[position]
    #不適解となる個体なら計算しない
    if(baseFit>=9) return(object)
    #
    pattern <- decodeCELFA(baseChrom, nof, nov)
    writeSemSpec(pattern)
    model <- specify.model2(file="specification_temp.txt")
    estimation <- try( sem(ram=model, S=dataobject@GenCovMatVar, N=dataobject@SampleS
ize) )
    modification <- mod.indices(estimation)
    modA <- replace( modification$mod.A, is.na(modification$mod.A), 0)[1:nov,(nov+1):
(nov+nof)]
    modP <- replace( modification$mod.P, is.na(modification$mod.P), 0)[(nov+1):(nov+n
of),(nov+1):(nov+nof)]
    #
    position <- which(modA==max(modA),arr.ind=T)
    pattern2 <- pattern
    pattern2$Loadings[position] <- 1
    writeSemSpec(pattern2)
    model <- specify.model2(file="specification_temp.txt")
    estimation2 <- try( sem(ram=model, S=dataobject@GenCovMatVar, N=dataobject@Sample
Size) )
    neoFit <- 9999
    if( class(estimation2)=="try-error" ) neoFit <- 999
    if( estimation2$convergence>=4 ) neoFit <- 99
    if( estimation2$convergence<=3 & is.null(estimation2$cov)==TRUE ) neoFit <- 9
    if( estimation2$convergence<=3 & is.null(estimation2$cov)==FALSE ) neoFit <- sqrt
( max( estimation2$criterion / ( 0.5 * estimation2$n * ( estimation2$n + 1 ) - estima
tion2$t ) - 1 / (estimation2$N - 1), 0) )
    #
    if(neoFit<baseFit)  {
        object@MatingPool[position,] <- minChrom
        object@MatingFitness[position] <- minFit
    }
    return(object)
}

##############################################
#LABEL: replaceCELFA(): CELFA において世代交代を行うための関数
#object: クラス GAEval のオブジェクト
#elite: エリート個体数
##############################################
replaceCELFA <- function(object, elite){
    #現在の個体と発生した子供を合併
    masterChrom <- rbind(object@Population, object@MatingPool)
    masterFit <- c(object@Fitness, object@MatingFitness)
    masterChrom <- masterChrom[ order(masterFit), ]
    masterFit <- masterFit[ order(masterFit) ]
    #エリート戦略を採る場合にはエリートを保存し，選択対象から外す
    if(elite>0){
        elChrom <-matrix(0, nrow=elite, ncol=object@Length)
        elFit <- numeric(elite)
        for(i in 1:elite){
            elChrom[i,] <- masterChrom[i,]
            elFit[i] <- masterFit[i]
        }
        masterChrom <- masterChrom[-(1:elite),]
        masterFit <- masterFit[-(1:elite)]
    }
    #適合度の逆数に基づいて次世代に生き残る個体を抽選
    ratio <- 1/masterFit
    ratio <- replace(ratio, is.infinite(ratio), 1e+99)
    choice <- sample( x=1:nrow(masterChrom), size=nrow(object@Population)-elite, prob
=ratio )
    #次世代集団の作成
    nextChrom <- masterChrom[choice,]
    nextFit <- masterFit[choice]
    if(elite>0){
        nextChrom <- rbind(nextChrom, elChrom)
        nextFit <- c(nextFit, elFit)
    }
    #GAEval オブジェクトの更新
    object@Generation <- as.integer(object@Generation+1)
    object@Population <- nextChrom
    object@Fitness <- nextFit
    object@MatingPool <- matrix(0)
    object@MatingFitness <- numeric(0)
    #Population のソート
    object@Population <- object@Population[order(object@Fitness),]
    object@Fitness <- object@Fitness[order(object@Fitness)]
    #履歴の記録
    object@History[[object@Generation]] <- list(Generation=object@Generation, Populat
ion=object@Population, Fitness=object@Fitness)
    return(object)
}

if(file.exists(".Rdata")==FALSE) searching()
if(file.exists(".Rdata")==TRUE){
    load(".Rdata")
    if(backupState==1) searching1()
    else if(backupState==2) searching2()
    else if(backupState==3) searching3()
    else searching()
}
```

References

Aitchison, J., & Silvey, D. C. (1958). Maximum likelihood estimation of parameters subject to restraints. *Annals of Mathematical Statistics*, *29*, 813–828.

Akaike, H. (1974). A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, *AC-19*, 716–723.

Akaike, H. (1987). Factor analysis and AIC. *Psychometrika*, *52*, 317–332.

Arbuckle, J. L. (1996). Full information estimation in the presence of incomplete data. In G. A. Marcoulides & R. E. Schumacker (Eds.), *Adbanced structural equation modeling: Issues and techniques* (pp. 243–277). Hillsdale, NJ: Lawrence Erlbaum Associates.

Arbuckle, J. L. (2006). *AMOS 7.0* [Computer software]. Chicago, IL: SPSS Inc.

Arminger, G. (1986). Linear stochastic differential equation models for panel data with unobserved variable. In N. B. Tuma (Ed.), *Sociological methodology* (pp. 187–213). San Francisco, CA: Jossey-Bass.

Bäck, T., Fogel, D. B., & Michalewicz, Z. (Eds.). (1997). *Handbook of evolutionary computation.* New York: IOP Publishing and Oxford University Press.

Bagozzi, R. P., & Yi, Y. (1992). Testing hypothesis about methods, traits, and communalities in the direct product model. *Applied Psychological Measurement*, *16*, 373–380.

Baker, J. E. (1985). Adaptive selection methods for genetic algorithms. *Proceedings of Genetic Algorithms and their Applications*, 101–111.

Beasley, J. E. (1993). Lagrangian heuristics for location problems. *European Journal of Operational Research*, *65*, 383–399.

Bentler, P. M. (1983). Some contributions to efficient statistics in structural models: Specification and estimation of moment structures. *Psychometrika*, *48*, 493–517.

Bentler, P. M. (1986). *Lagrange multiplier and wald tests for EQS and EQS/PC.* Los Angeles, CA: BMDP Statistical Software.

Bentler, P. M. (1990). Comparative fit indexes in structural models. *Psychological Bulletin*, *107*, 238–246.

Bentler, P. M. (1995). *EQS structural equations program manual.* Encino, CA:

Multivariate Software.

Bentler, P. M. (2003). *EQS 6.1 for windows* [Computer software]. Encino, CA: Multivariate Software.

Bentler, P. M., & Bonnet, D. G. (1980). Significance tests and goodness of fit in the analysis of covariance structures. *Psychological Bulletin*, *88*, 588–606.

Bentler, P. M., & Dijkstra, T. (1985). Efficient estimation via linierization in structural models. In P. R. Krishnaiah (Ed.), *Multivariate anakysis VI* (pp. 9–42). Amsterdam, Holland: North-Holland.

Bentler, P. M., Poon, W. Y., & Lee, S.-Y. (1988). Generalized multimode latent variable models: Implementation by standard programs. *Computational Statistics And Data Analysis*, *7*, 107–118.

Bentler, P. M., & Weeks, D. G. (1980). Linear structural equations with latent variables. *Psychometrika*, *45*, 289–308.

Blalock, H. M. (1963). Making causal inferences for unmeasured variables from correlations among indicators. *American Journal of Sociology*, *69*, 53–62.

Blau, P. M., & Duncan, O. D. (1967). *The american occupational structure.* New York, NY: John Wiley & Sons.

Boes, K. D., Kahng, A. B., & Muddu, S. (1994). A new adaptive multi-start technique for combinatorial global optimization. *Operations Research Letters*, *16*, 101–113.

Bollen, K. A. (1989). *Structural equations with latent variables.* New York, NY: John Wiley & Sons.

Bollen, K. A., & Paxton, P. (1998). Interactions of latent variables in structural equation modeling. *Structural Equation Modeling*, *5*, 267–293.

Boomsma, A. (1983). *On the robustness of LISREL (maximum likelihood estimation) against small sample size and nonnormality.* Amsterdam, Holland: Sociometric Research Foundation.

Browne, M. W. (1974). Generalized least-squares estimators in the analysis of covariance structures. *South African Statistical Journal*, *8*, 1–24.

Browne, M. W. (1982). Covariance structure. In D. M. Hawkins (Ed.), *Topics in multivariate analysis* (pp. 72–141). Cambridge, England: Cambridge University Press.

Browne, M. W. (1984). Asymptotically distribution-free methods in the analysis of covariance structures. *British Journal of Mathematical and Statistical Psychology*, *37*, 62–83.

Browne, M. W., & Cudeck, R. (1993). Alternative ways of assesing model fit. In K. Bollen & J. S. Long (Eds.), *Testing structural equations* (pp. 137–162). Newbury Park, CA: SAGE Publications.

Caprara, A., Fischetti, M., & Toth, P. (1999). A heuristic method for the set covering problem. *Operations Research*, *47*, 730–743.

Carroll, J. B. (1993). *Human cognitive abilities: A survey of factor-analysis studies.* Cambridge: Cambridge University Press.

Cattell, R. B. (1990). Biological dimensions of personality. In L. A. Pervin (Ed.), *Handbook of personality: Theory and research* (pp. 101–110). New York, NY: Guilford Press.

Cerny, V. (1985). hermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Application*, *45*, 41–51.

Chambers, L. (Ed.). (1995). *Practical handbook of genetic algorithms: Applications* (Vol. I). Boca Raton, FL: CRC Press.

Charon, I., & Hudry, O. (1993). The noising method: a new method for combinatorial optimization. *Operations Research Letters*, *14*, 133–137.

Codenotti, B., Manzini, G., Margara, L., & Resta, G. (1996). Perturbation: an efficient technique for the solution of very large instances of the Euclidean TSP. *INFORMS Journal on Computing*, *8*, 125–133.

Cohoon, J. P., & Paris, W. D. (1987). Genetic placement. *IEEE Transactions on Computer-Aided Design*, *CAD-10*, 483–492.

Costa Jr., P. T., & McCrae, R. R. (1992). *NEO-PI-R professional manual: Revised NEO personality inventory(NEO-PI-R) and NEO five-factor inventory(NEO-FFI).* Odessa, FL: Psychological Assessment Resources.

Darema-Rogers, F., Kirkpatrick, S., & Norton, V. A. (1987). Parallell algorithms for chip placement by simulated annealing. *IBM Journal of Research and Development*, *31*, 391–402.

Davis, L. (1985). Applying adaptive algorithms to epistatic domains. *Proceedings of the 9th Joint Conference on Artificial Intelligence*, 162–164.

Davis, L. (1991). *Handbook of genetic algorithms.* New York: Van Nostrand Reinhold.

DeJong, K. (1975). *The analysis and behaviour of a class of genetic adaptive systems.* Unpublished doctoral dissertation, University of Michigan.

Dempster, A. P., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm (with discussion). *Journal of the Royal Statistical Society, Series B*, *39*, 1–38.

Dorigo, M. (1992). *Optimization, learning, and natural algorithms.* Unpublished doctoral dissertation, Politecnico di Milano.

Dueck, G. (1993). New optimization heuristics: the great deludge algorithm and the recored-torecored travel. *Journal of Computational Physics*, *104*, 86–92.

Dueck, G., & Scheuer, T. (1990). Threshold accepting: a general purpose optimization algorithm appearing superiot to simmulated annealing. *Journal of Computational Physics*, *90*, 161–175.

Duncan, O. D. (1966). Path analysis: Sociological examples. *American Journal* *of Sociology*, *74*, 119–137.

Duncan, S. C., & Duncan, T. E. (1996). A multivariate growth curve model of adolescent substance use. *Structural Equation Modeling*, *3*, 323–347.

Duncan, T. E., Duncan, S. C., Alpert, A., Hops, H., Stoolmiller, M., & Muthén, B. O. (1997). Latent variable modeling of longitudinal and multilevel substance use data. *Multivariate Behavioral Research*, *32*, 275–318.

Eysenck, H. J. (1990). Biological dimensions of personality. In L. A. Pervin (Ed.), *Handbook of personality: Theory and research* (pp. 244–276). New York, NY: Guilford Press.

Feo, T. A., & Resende, M. G. C. (1989). A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, *8*, 67–71.

Feo, T. A., & Resende, M. G. C. (1995). Greedy randomized adaptive search procedure for maximum independent set. *Journal of Global Optimization*, *6*, 109–133.

Feo, T. A., Venkartraman, K., & Bard, J. F. (1991). A GRASP for a difficult single machine scheduling problem. *Computers and Operations Research*, *18*, 635–643.

Fogel, D. B. (1992). *Evolutionary computation: Toward a new philosophy of machine intelligence.* IEEE Press.

Fornell, C. (Ed.). (1982a). *A second generation of multivariate analysis* (Vol. 1). New York, NY: Praeger Pub.

Fornell, C. (Ed.). (1982b). *A second generation of multivariate analysis* (Vol. 2). New York, NY: Praeger Pub.

Foulds, L. R. (1984). *Combinatorial optimization for undergraduates.* New York, NJ: Springer-Verlag.

Fox, J. (1984). *Linear statistical models and related methods.* New York, NY: John Wiley & Sons.

Fox, J. (2006). *sem: Structural equation models. R package version 0.9-2* [Computer software].

Gabbert, P., Brown, D., Huntley, C., Markowicz, B., & Sappingston, D. (1991). A system for learning routes and schedules with genetic algorithms. In *Proceedings of ICGA-91.* Morgan Kaufmann Publishers.

Garey, M., & Johnson, D. (1979). *Computer and intractability: A guide to the theory of NP-completeness.* San Francisco, LA: W. H. Freeman.

Geman, S., & Geman, D. (1984). Stochastic relaxation, gibbs distribution and the bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *6*, 721–741.

Gillies, A. M. (1985). *Machine learning procedures for generating image domain feature detectors.* Unpublished doctoral dissertation, University of Michigan.

Glover, F. (1989). Tabu search - part i. *ORSA Journal of Computing*, *1*, 190–206.

Glover, F. (1990). Tabu search - part ii. *ORSA Journal of Computing*, *2*, 4–32.

Glymour, C., Scheines, R., Spirtes, P., & Kelly, K. (1987). *Discovering causal structure.* Orland, FL: Academic Press.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization and machine learning.* Massachusetts: Addison Wesley.

Goldberg, D. E. (1990a). *Real-coded genetic algorithms, virtual alphabets, and block* (Tech. Rep. No. 90001). University of Illinois.

Goldberg, D. E. (1990b). A note on boltzmann tournament selection for genetic algorithms and population-oriented simulated annealing. *Complex Systems*, *4*, 445–460.

Goldberg, D. E., & Lingle, R. (1985). Alleles, loci, and the traveling salesman problem. *Proceedings of the International Conference on Genetic Algorithms and their Applications.*

Goldberg, L. R. (1990). An alternative "description of personality": The big five factor structure. *Journal of Personality and Social Psychology*, *59*, 1216–1229.

Goldberg, L. R. (1993). The structure of phenotypic personality traits. *American Psychologist*, *46*, 26–34.

Goldstein, H. (1995). *Multilevel statistical models* (Second ed.). London, England: Edward Arnold.

Green, P. J. (1995). Reversible jump markov chain monte carlo computation and bayesian model determination. *Biometrika*, *82*, 711-732.

Grefenstette, J. J. (1986). Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, *16*, 122–128.

Gu, J., & Huang, X. (1994). Efficient local search with search space smoothing: a case study of the traveling salesman problem (TSP). *IEEE Transactions on Systems, Man, and Cybernetics*, *24*, 728–735.

Guttman, L. (1952). Multiple group methods for common-factor analysis: Their basis, computation, and interpretation. *Psychometrika*, *17*, 209–222.

Guttman, L. A. (1954). A new approach to factor analysis. the radex. In P. F. Lazarsfeld (Ed.), *Mathematical thinking in the social sciences* (pp. 258–348). New York, NY: Columbia University Press.

Harwood, S., & Scheines, R. (2002). *Genetic algorithm search over causal models* (Tech. Rep. No. CMU-PHIL-131). Pgh, PA 15213: Dept. of Philosophy, Carnegie Mellon Univ.

Hastings, W. K. (1970). Monte carlo sampling methods using markov chains and their applications. *Biometrika*, *57*, 97–109.

Henderson, D. A., & Denison, D. R. (1989). Stepwise regression in social and psychological research. *Psychological Reports*, *64*, 251–257.

Holland, J. H. (1975). *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence.* Michigan: The University of Michigan Press.

Hollstien, R. B. (1971). *Artificial genetic adaptation in computer control systems.* Unpublished doctoral dissertation, University of Michigan.

Holzinger, K. J., & Swineford, F. (1939). *A study in factor analysis: The stability of a bi-factor solution* (Supplementary Educational Monographs No. 48). Illinois: The University of Chicago.

Horowitz, E., & Sahni, S. (1984). *Fundamentals of computer algorithms.* Rockville, MD: Computer Science Press.

Johnson, D. S. (1990). Local optimization and the traveling salesman problem. In M. S. Paterson (Ed.), *Automata, languages and programming, vol.443 of lecture notes in computer science* (pp. 256–278). New York, NY: Springer-Verlag.

Jong, D. (1975). An analysis of the behavior of a class of genetic algorithms. *Dissertation Abstracts International*, *36*, 5140B.

Jöreskog, K. G. (1973). A general method for estimating a linear structural equation system. In A. S. Goldberger & O. D. Duncan (Eds.), *Structural equation models in the social sciences* (pp. 85–112). New York, NY: Academic Press.

Jöreskog, K. G. (1993). Testing structural equation models. In K. A. Bollen & J. S. Lang (Eds.), *Testing structural equation modeling* (pp. 294–316). Newbury Park, CA: Sage.

Jöreskog, K. G., & Sörbom, D. (2006). *LISREL 8.8 for windows* [Computer software]. Lincolnwood, IL: Scientific Software International.

Jöreskog, K. G., & Sörborm, D. (1986). *LISREL VI: Analysis of linear structural relationships by maximum likelihood and least square methods.* Mooresville, IN: Scientific Software Inc.

Jöreskog, K. G., & Sörborm, D. (1990). Model search with TETRAD II and LISREL. *Sociological Methods and Research*, *19*, 93–106.

Kano, Y. (2001). Structural equation modeling with experimental data. In R. Cudeck, S. Du Toit, & D. Sörbom (Eds.), *Structural equation modeling: Present and future* (pp. 381–402). Lincolnwood, IL: Scientific Software International.

Kano, Y., & Miura, A. (2002). *Graffical multivariate analysis with AMOS, EQS, and CALIS.* Kyoto, Japan: Gendai Suugaku Sha. (AMOS, EQS, CALIS によるグラフィカル多変量解析)

Keesling, J. W. (1972). *Maximum likelihood approaches to causal analysis.* Unpublished doctoral dissertation, Department of Education: University of Chicago.

Kernighan, B. W., & Lin, S. (1970). An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, *49*, 291–307.

Kirkpatrick, S., Gelatt Jr., C., & Vecchi, M. (1983). Optimization by simulated annealing. *Science*, *220*, 498–516.

Kline, R. B. (2005). *Principles and practice of structural equation modeling* (Second ed.). New York, NY: Guilford Press.

Kling, R. M., & Banerjee, P. (1991). Empirical and theoritical studies of the simulated evolution method applied to standard cell placement. *IEEE Transaction on Computer-Aided Design*, *10*, 1303–1315.

Lawler, E. L. (1976). *Combinational optimization: Networks and matroids.* New York, NY: Holt, Rinehart and Winston.

Lee, S.-Y. (1985). On testing functional constraints in structural equation models. *Biometrika*, *72*, 125–131.

Lee, S.-Y. (1990). Multilevel analysis of structural equation models. *Biometrika*, *77*, 763–772.

Lee, S.-Y., & Bentler, P. M. (1980). Some asymptotic properties of constrained generalized least squares estimation in covariance structure analysis. *South African Statistical Journal*, *14*, 121–136.

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

Lord, F. (1980). *Applications of item response theory to practical testing problems.* Lawrence Erlbaum Associates.

MacCallum, R. C. (1986). Specification searches in covariance structure modeling. *Psychological Bulletin*, *100*, 107–120.

Man, K. F., Tang, K. S., & Kwong, S. K. (1999). *Genetic algorithms: Concepts and designs.* London: Springer-Verlag.

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

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.

Martin, O., & Otto, S. W. (1996). Combining simulated annealing with local search heuristic. *Annals of Operations Research*, *63*, 57–75.

Martin, O., Otto, S. W., & Felten, E. W. (1991). Large-step markov chains for the travel salesman problem. *Complex Systems*, *5*, 299–326.

Maruyama, G. M. (1998). *Basiscs of structural equation modeling.* Thousand Oaks, CA: Sage.

McArdle, J. J., & Hamagami, F. (1996). Multilevel models from a multiple group structural equation perspective. In G. Marcoulides & R. Schumacker (Eds.), *Advanced structural equation modeling: Issues and techniques* (pp. 89–124). Mahwah, NJ: Lawrence Erlbaum Associates.

McArdle, J. J., & McDonald, R. P. (1984). Some algebraic properties of the reticular action model for moment structures. *British Journal of Mathematical and Statistical Psychology*, *37*, 234–251.

McCrae, R. R., & Costa Jr., P. T. (1997). *American Psychologist*, *52*, 509–516.

McDonald, R. P. (1985). *Factor analysis and related method.* Hillsdale, NJ: Lawrence Erlbaum Associates.

McDonald, R. P., & Goldstein, H. (1989). Balanced versus unbalanced designs for linear structural relation in two level data. *British Journal of Mathematical and Statistical Psychology*, *42*, 215–232.

McDonald, R. P., & Ho, M.-H. R. (2002). Principles and practice in reporting structural equation analyses. *Psychological Methods*, *7*, 64–82.

Meredith, W., & Tisak, J. (1990). Latent curve analysis. *Psychometrika*, *55*, 107–122.

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., & Teller, E. (1953). Equations of state calculations by fast computing machine. *Journal of Chemical Physics*, *21*, 1087–1091.

Michalewicz, Z. (1996). *Genetic algorithms + data structure = evolution program.* New York, NY: Springer-Verlag.

Mills, J. D., Olejnik, S. F., & Marcoulides, G. A. (2005). The tabu search procedure: An alternative to the variable selection methods. *Multivariate Behavioral Research*, *3*, 351–371.

Mladenović, N., & Hansen, P. (1997). Variable neighborhood search. *Computers and Operational Research*, *24*, 1097–1100.

Mooijaart, A. (1985). Factor analysis for non-normal variables. *Psychometrika*, *50*, 323–342.

Mulaik, S. A. (1986). Factor analysis and psychometrika: Major development. *Psychometrika*, *51*, 23–33.

Murohashi, H. (2003). chi square test. In H. Toyoda (Ed.), *Q & A's for covariance structure analysis* (pp. 120–121). Tokyo, Japan: Asakura-shoten. (共分散構造分析【疑問編】—構造方程式モデリング—)

Murohashi, H., & Toyoda, H. (2004). Structural equation modeling using ability parameters:analysis in the situation where item parameters have been estimated by item response theory. *The Japanese Journal of Psychology*, *75*, 381–388. (被験者母数を用いた構造方程式モデリング—IRT において項目母数が推定されている場合の分析—)

Muthén, B. O. (1984). A general structural equation modeling with dichotomous, ordered, categorical and continuous latent variable indicators. *Psychometrika*, *49*, 115–132.

Muthén, B. O. (1994). Multilevel covariance structure analysis. *Sociological Methods and Research*, *22*, 376–398.

Muthén, B. O. (2004). *Mplus technical appendices.* Los Angeles, CA: Muthén & Muthén.

Muthén, B. O., & Satorra, A. (1995). Complex sample data in structural equation modeling. In P. Marsden (Ed.), *Sociological methodology* (pp. 216–316). Oxford, England: Basil Blackwell.

Muthén, B. O., & Shedden, K. (1999). Finite mixture modeling with mixture outcomes using the EM algorithm. *Biometrics*, *55*, 463–469.

Muthén, B. O., Shedden, K., & Spisic, D. (1999). *General latent variable mixture modeling* (Tech. Rep.).

Muthén, L., & Muthén, B. (2006). *MPlus (version 4.1)* [Computer software]. Los Angeles, CA: Muthén and Muthén and.

Neale, M. C., Boker, S. M., Xie, G., & Maes, H. H. (2004). *Mx 1.54a for windows* [Computer software]. Richmond, VA: Department of Human Genetics, Virginia Institute for Psychiatric and Behavioral Genetics, Virginia Commonwealth University.

Nonobe, K., & Ibaraki, T. (1998). A tabu search approach to the constraint satisfaction problem as a general problem solver. *European Journal of Operational Research*, *106*, 599–623.

Nonobe, K., & Ibaraki, T. (1999). *An improved tabu search method for the weighted constraint satisfaction problem* (Tech. Rep. No. 99022). Kyoto, Japan: Dept. of Applied Mathematics and Physics, Graduate School of Informatics, Kyoto Univ.

Norman, W. T. (1963). Toward and adequate taxonomy of personality attributes: Replicated factor structure in peer nomination personality ratings. *Journal of Abnormal and Social Psychology*, *66*, 574–583.

Oliver, I., Smith, D., & Holland, J. R. (1987). A study of permutation crossover operators on the traveling salesman problem. In J. Grefenstette (Ed.), *Genetic algorithms and their applications: Proceedings of the 2nd international conference on genetic algorithms* (pp. 224–230). Hillsdale, NJ: Lawrence Erlbaum.

Papadimitriou, C., & Steiglitz, K. (1982). *Combinatorial optimization: Algorithms and complexity.* Englewood Cliffs, NJ: Prentice Hall.

Peabody, D., & Goldberg, L. R. (1989). Some determinants of factor structures from personality-trait descriptions. *Journal of Personality and Social Psychology*, *57*, 552–567.

Ping, J. R. A. (1996). Interaction and quadratic effect estimation: A two step technique using structural equation analysis. *Psychological Bulletin*, *119*, 166–175.

Raftery, A. E. (1993). Bayesian model selection in structural equation models. In K. A. Bollen & J. S. Long (Eds.), *Testing structural equation models* (pp. 163–180). Newburry Park, CA: Sage.

Rao, C. R. (1948). Large sample tests of statistical hypothesis concerning several parameters with application to problems of estimation. *Proceedings of the Cambridge Philosphical Society*, *44*, 50–57.

Raykov, T., & marcoulides, G. A. (2000). *A first course in structural equation modeling*. Mahwah, NJ: Lawrence Earlbaum Associates.

R Development Core Team. (2006). *R: a language and environment for statistical computing* [Computer software]. Vienna, Austria: R Foundation for Statistical Computing.

Reiter, S., & Rice, D. B. (1966). Discrete optimizing solution procedures for linear and nonlinear integer programming problems. *Management Science*, *12*, 829–850.

Saab, Y. G., & Rao, V. B. (1991). Combinational optimization by stochastic evolution. *IEEE Transactions on Computer-Aided Design*, *10*, 535–545.

Sait, S. M., & Youssef, H. (2000). *Iterative computer algorithms with applications in engineering: Solving combinational optimization problems.* New York: Wiley-IEEE Computer Society Press.

Sarris, W. E., Satorra, A., & Sörbom, D. (1987). The detection and correction of specification errors in structural equation models. In C. Clogg (Ed.), *Sociological methodology 1987* (pp. 105–129). San Francisco, CA: Jossey Bass.

SAS Institute Inc. (2004). *The SAS system for windows (version 9)* [Computer software]. Cary, NC: Author.

Satorra, A., & Sarris, W. E. (1985). Power of the likelihood test in covariance structure analysis. *Psychometrika*, *50*, 83–90.

Schumacker, R. E., & Marcoulides, G. A. (1998). *Interaction and nonlinear effects in structural equation modeling.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Shimizu, S., & Kano, Y. (In press). Use of non-normality in structural equation modeling: Application to direction of causation. *Journal of Statistical Planning and Inference.*

Sörbom, D. (1982). Structural equation models with structured means. In K. G. Jöreskog & H. Wold (Eds.), *Systems under indirect observation; causality, structure, prediction* (Vol. 1, pp. 183–195). Amsterdam, Holland: North-Holland.

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

Spearman, C. (1904a). General intelligence, objectively determined and measured. *American Journal of Psychology*, *15*, 201–293.

Spearman, C. (1904b). The proof and measurement of association between two things. *American Journal of Psychology*, *15*, 72–101.

Spears, W. M., & DeJong, K. (1991). An analysis of multi-point crossover. In J. E. Rawlins (Ed.), *Foundations of genetic algorithms* (pp. 301–315). San Francisco, CA: Morgan Kaufmann.

StatSoft Inc. (2004). *STATISTICA (version 7)* [Computer software]. Tulsa, OK: Author.

Steiger, J. H., & Lind, J. C. (1980). *Statistically based tests for the number of common factors.* Paper presented at the annual meeting of the Psychometric Society, Iowa.

Sun, M., & Mckeown, P. G. (1993). Tabu search applied to the general fixed charge problem. *Annals of Operations Research*, *41*, 405–420.

Systat Software Inc. (2004). *SYSTAT (version 11)* [Computer software]. Richmond, CA: Author.

Syswerda, G., & Palmucci, J. (1991). The application of genetic algorithm to resource scheduling. In *Proceedings of ICGA-91.* Morgan Kaufmann Publishers.

Thomarken, A. J., & Waller, N. G. (2003). Potential problems with "well-fitting" models. *Journal of Abnormal Psychology*, *112*, 578–598.

Thurstone, L. L. (1935). *The vectors of mind: Multiple-factor analysis for the isolation of primary traits.* Chicago, IL: University of Chicago Press.

Thurstone, L. L. (1947). *Multiple factor analysis: A development and expansion of the vector of mind.* Chicago, IL: University of Chicago Press.

Thurstone, L. L., & Thurstone, T. G. (1941). *Factorial studies of intelligence.* University of Chicago Press.

Timothy, A., & Burke, P. J. (1994). Exploratory and confirmatory tests of the big five and tellegen's three- and four-dimensional models. *Journal of Personality and Social Psychology*, *66*, 93–114.

Toyoda, H. (1992). *Covariance structure analysis with sas.* Tokyo, Japan: University of Tokyo Press. (SAS による共分散構造分析)

Toyoda, H. (1994). A new identification rule and an estimator for the simultaneous equation model using the notation of the reticular action model. *Behaviormetrika*, *21*, 163–175.

Toyoda, H. (1998). *Introduction to covariance structure analysis: Structural equation modeling.* Tokyo, Japan: Asakura-Shoten. (共分散構造分析 [入門編]—構造方程式モデリング—)

Tupes, E. C., & Christal, R. E. (1961). *Resurrent personality factors based on trait ratings* (Tech. Rep. No. ASD-TR-61-97). Lackland Air Force Base, TX: U.S. Air Force.

Verhees, J., & Wansbeek, T. J. (1990). A multimode direct product model for covariance structure analysis. *British Journal of Mathematical and Statistical Psychology*, *43*, 231–240.

Wald, A. (1943). Tests of statistical hypotheses concerning several parameters when the number of observation is large. *Transactions of the American Mathematical Society*, *54*, 426–482.

Wiggins, J. S. (Ed.). (1996). *The five-factor model of personality.* New York, NY: The Guilford Press.

Wiley, D. E. (1973). The identification problem for structural equation models with unmeasured variables. In A. S. Goldberger & O. D. Duncan (Eds.), *Structural equation models in the social sciences* (pp. 69–83). New York, NY: Academic Press.

Wright, A. H. (1991). Genetic algorithms for real parameter optimization. In J. E. Rawlins (Ed.), *Foundations of genetic algorithms* (pp. 205–218). San Francisco, CA: Morgan Kaufmann.

Wright, S. G. (1918). On the nature of size factors. *Genetics*, *3*, 367–374.

Yanagiura, M., & Ibaraki, T. (2001). *Combinatorial optimization with metaheuristics.* Tokyo, Japan: Asakura-shoten. (組み合わせ最適化—メタ戦略を中心として—)

Youssef, H., Sadiq, M., Nassar, K., & Benten, M. S. T. (1995). Performance driven standard-cell placement using the genetic algorithm. In *Fifth great lakes symposium on VLSI.* GLSVLSI'95.