ARTICLES

Inferring the structure of latent class models using a genetic algorithm

HAN L. J. VAN DER MAAS, MAARTJE E. J. RAJMAKERS, and INGMAR VISSER
University of Amsterdam, Amsterdam, The Netherlands

Present optimization techniques in latent class analysis apply the expectation maximization algorithm or the Newton–Raphson algorithm for optimizing the parameter values of a prespecified model. These techniques can be used to find maximum likelihood estimates of the parameters, given the specified structure of the model, which is defined by the number of classes and, possibly, fixation and equality constraints. The model structure is usually chosen on theoretical grounds. A large variety of structurally different latent class models can be compared using goodness-of-fit indices of the chi-square family, Akaike’s information criterion, the Bayesian information criterion, and various other statistics. However, finding the optimal structure for a given goodness-of-fit index often requires a lengthy search in which all kinds of model structures are tested. Moreover, solutions may depend on the choice of initial values for the parameters. This article presents a new method by which one can simultaneously infer the model structure from the data and optimize the parameter values. The method consists of a genetic algorithm in which any goodness-of-fit index can be used as a fitness criterion. In a number of test cases in which data sets from the literature were used, it is shown that this method provides models that fit equally well as or better than the models suggested in the original articles.

Latent class analysis (LCA) is routinely used in the analyses of numerous types of data sets in psychology and other fields of investigation. For example, in developmental psychology, LCA is used to classify subjects according to the different strategies that they use in solving proportional reasoning problems (e.g., Jansen & van der Maas, 1997). LCA belongs to the family of latent structure models (Heinen, 1996), which are used to classify subjects on the basis of categorical response variables. For an overview of the theory and applications of LCA, see, for instance, Hagenaars and McCutcheon (2002), Heinen (1996), and Rost and Langeheine (1997). In the next section, we will present an overview of the main concepts that are necessary for understanding LCA.

Algorithms for optimizing latent class model parameters require that the structure of the model has to be chosen prior to the optimization of parameter values. For instance, the expectation maximization (EM) algorithm

(Dempster, Laird, & Rubin, 1977) provides maximum likelihood (ML) estimates of the parameters, given the number of classes and, possibly, constraints, such as fixed parameters and equalities between parameters. The Newton–Raphson (NR) method similarly provides ML estimates of latent class model parameters, possibly subjected to nonlinear constraints, which is not possible within the EM algorithm (McCutcheon, 1987). However, the EM and NR algorithms cannot be used to infer the structure of the model—that is, the number of classes or the constraints to which the parameters are subjected. Naturally, the structure of a model can vary in many ways. Hence, finding the optimal model structure can be cumbersome when there is no systematic way to search the space of candidate model structures. In particular, models with many classes and many constraints can be very difficult to find. In cases in which a limited set of a priori models can be specified, this is a minor problem. However, a priori hypotheses may not completely determine the model structure. More important, it is always useful to learn whether better-fitting models exist and, if so, how they compare with the models that were specified in advance.

Similar model search problems occur when other types of statistical analyses are used, and hence, the suggested approach can also be applied in those situations. For example, in graphical modeling, which is used to determine the causal relations between a number of variables, it is not in general known which model structures and how

The research of M.E.J.R. has been made possible by a fellowship of the Royal Netherlands Academy of Art and Sciences. The research of I.V. has been made possible by a grant from the Netherlands Organisation for Scientific Research (NWO). Correspondence concerning this article should be addressed to H. L. J. van der Maas, Department of Psychology, University of Amsterdam, Roetersstraat 15, 1018 WB Amsterdam, The Netherlands (e-mail: hvandermaas@fmg.uva.nl). The computer program GALCA can be downloaded at users.fmg.uva.nl/hvandermaas.

Note—This article was accepted by the previous editor, Jonathan Vaughan.

many fit a given covariance structure (see Pearl, 2000, pp. 145ff. for a discussion). Similarly, in factor analysis, there may be many different latent structures that result in (statistically) appropriate models (MacCallum, Wegener, Uchino, & Fabrigar, 1993). Whether such models lend themselves to useful interpretations is, of course, a different question. However, this does not diminish the utility of having at hand a series of appropriate models.

In this article, we describe a method for inferring the structure of latent class models and, simultaneously, optimizing their parameters. Kwong, Chau, Man, and Tang (2001) used a similar approach in hidden Markov modeling. They used a genetic algorithm (GA) to search adequate starting values (of so-called *left-right*, or Bakis, models, which is a highly constrained class of models typically used in speech recognition), which are then further optimized with the EM algorithm.¹ The description of our method is sufficiently general to be applicable to other types of models as well, such as those discussed above. A natural method for searching the space of possible model structures, which is our goal, is random search. The main disadvantage of random search is that it is very slow and does not guarantee optimal solutions. Our method, based on GAs, may perform much better, because it utilizes partial solutions from different candidate models and recombines those to form new candidate models.

GAs are inspired by biological evolution. Many fine introductions to GAs have appeared on the Internet (e.g., Wikipedia) and in books (e.g., Mitchell, 1996). A population of parameter strings (representing latent class models) is recombined (by crossing over) and modified (by mutation) in a series of generations. In each generation, the parameter strings that prove to be the fittest (i.e., in terms of yielding the best value of a given fit index) are apportioned more *offspring* than strings that are less fit are. This way “good” parameter values are selected, and optimal parameter strings emerge after a series of generations. This optimization technique is very efficient, as compared with random search, and often is applied for multipeaked nonsmooth search spaces for which standard optimization techniques fail (Beasley, Bull, & Martin, 1993a). GAs are related to simulated annealing in that they use random forces to avoid local minima, but they differ with respect to the application of parallel search.

Any search method, be it systematic or random, results in a series of candidate models. Consequently, it is necessary to compare resulting models, using goodness-of-fit criteria. Well-known examples of such model selection measures are Akaike’s information criterion (AIC) and the Bayesian information criterion (BIC; see the next section for formal definitions). Both of these measures are based on the likelihood ratio, which is related to the chi-square test, and they include a penalty term for the number of freely estimated parameters. In the present article, we use AIC and BIC as a fitness criteria for the GA. The choice of a fitness criterion is not essential

to our approach. Any preferred model selection criterion can be used in GAs. Because of this, we will not discuss the choice of AIC and BIC extensively but will refer to the literature whenever necessary (for an overview of model selection criteria, see Myung, Foster, & Browne, 2000).

Applying a GA in inferring the structure of latent class models requires a representation of model structure that is sufficiently general to allow different numbers of latent classes and different types of constraints on the parameters. By using an intelligently chosen representation for model structures, the efficiency of the GA can be enormously increased. The next section will provide an example of LCA that explains the main concepts necessary for understanding the results. Next, we will describe the representation of latent class models in genetic strings. Furthermore, and depending on the choice of representation, different mutation and crossing-over operators may be added to the GA. Further, we will discuss various options and choices we have made that resulted in the computer program Genetic Algorithm for Latent Class Analysis (GALCA) that we use throughout the article. In the final two sections, GALCA will be used successfully to analyze a number of simulated and empirical data sets.

Latent Class Analysis

Latent class models belong to the family of latent structure models whose goal is to explain the relationship between a number of observed variables by assuming unobserved (latent) variables. The factor model is a well-known example, in which it is assumed that both observed and latent variables are continuous. LCA, on the other hand, is used when both latent and observed variables are assumed to be discrete or categorical. In particular, in LCA, a typology may be constructed from a number of observed variables, whereas in factor analysis a trait is constructed from the observables. For overviews of techniques and applications, see, for example, Hage-naars and McCutcheon (2002), McCutcheon (1987), and Rost and Langeheine (1997).

McCutcheon (1987) discussed a number of example data sets that are analyzed using LCA. One of those data sets is analyzed below. The data are from a postelection survey in which people were asked the following four questions about their level of political involvement:

1. Did you vote?
2. During the campaign, did you talk to any people and try to show them why they should vote for one of the parties or candidates?
3. Did you go to any political meetings, rallies, fund-raising dinners, or things like that?
4. Did you do any work for one of the parties or candidates?

These questions are binary items—that is, only *yes* or *no* responses are allowed. As a result, there are 16 possible answer patterns: 1111, 1110, . . . , 0000, where 1 is the code for *yes* and 0 for *no*. In general, LCA is con-

cerned with finding patterns in data sets of this kind. In this case, for example, it may be expected that successive answers are dependent, meaning that people who answer *yes* to the fourth question are very likely to have answered *yes* to the previous questions as well. This assumption is formalized in the Proctor model, in which people can be classified into five classes of decreasing political involvement. These classes correspond to the following true-type or latent answer patterns: 1111, 1110, 1100, 1000, and 0000. The Proctor model is a so-called *scaling* model in which both items and people are ordered (see Dayton, 1998, for an overview of scaling models). Table 1 provides the parameters of the Proctor model for four binary items.

In Table 1, Class 1 corresponds to the latent answer pattern 1111. The parameter a is the item correct probability (in this case, the probability of answering *yes*)—that is, the probability of answering according to this latent pattern. In general, a is allowed to differ from 1 in order to allow for the possibility of measurement error. In particular, in the Proctor model, the measurement errors are constrained to be equal for each item and each class, whereas in general, error may vary between items and classes. Maximum likelihood parameter estimation for the above model for McCutcheon's (1987) data results in an estimate for a of .953—that is, quite close to one.

This model provides a rather poor goodness of fit with an L^2 (log likelihood ratio) of 138.19 and 10 degrees of freedom (df). The df is calculated as $df = \#f - \#p - 1$. Here, $\#f$ is the number of frequencies to be explained by the model, and $\#p$ is the number of freely estimated parameters in the model. In the election data above, there are 16 possible answer patterns, and hence, $\#f = 16$. The Proctor model for these data has six parameters—that is, the latent class proportions Pc_i , $i = 1, \dots, 5$ and the probability correct parameter a . One of the latent class proportions does not need to be estimated, since the proportions sum to unity. Consequently, the number of freely estimated parameters ($\#p$) equals 5, resulting in $df = 10$.

Both the L^2 and the χ^2 goodness-of-fit statistics are χ^2 distributed with df degrees of freedom. Hence, their expected values are equal to df . These measures are so-called *absolute* goodness-of-fit measures. When they are nonsignificant, the model provides an adequate description of the data. Also, throughout the present article, the AIC and BIC statistics are used as relative measures of goodness-of-fit. These are meant to determine whether one model is better or worse than another model (e.g., Clogg, 1995). In latent class analysis, the AIC and BIC are computed as follows:

$$AIC = L^2 + 2 \#p \quad (1)$$

and

$$BIC = L^2 + \#p \ln(N), \quad (2)$$

where N is the number of subjects (1,404 in the election data). Both AIC and BIC consist of two terms, the first

Table 1
Proctor Model for McCutcheon's (1987) Data

Class	Proportion	Q1	Q2	Q3	Q4
1	Pc_1	a	a	a	a
2	Pc_2	a	a	a	$1-a$
3	Pc_3	a	a	$1-a$	$1-a$
4	Pc_4	a	$1-a$	$1-a$	$1-a$
5	Pc_5	$1-a$	$1-a$	$1-a$	$1-a$

Note— Pc_i is the proportion of class i ; a is the probability of a positive response. Q, question.

of which is L^2 ; the second contains the number of parameters. The rationale is that the L^2 provides the goodness of fit of the model, whereas the second term is a penalty for the number of parameters used in the model. So, the best models are models with the lowest AIC and BIC values.

Representing Latent Class Parameters and Model Structure in Genetic Strings

We will discuss the representation of latent class model parameters, using an example that reveals the important dimensions on which models may vary. Suppose that we have five binary items and that the data consist of the frequencies of the 32 possible answer patterns. A possible three-class model for these data is shown in Table 2.

This model has three latent classes with varying latent class proportions: a large class that has consistently low conditional probabilities of a correct answer, a smaller class that has consistently high conditional probabilities of a correct answer, and a small class that has three high and two low conditional probabilities. The model incorporates 17 free parameters (three classes times five conditional probabilities for items, plus three minus one class proportions). In this model, some conditional probabilities are equal [$p(2,5) = p(3,5)$]. If an equality is specified beforehand between N parameters, we lose only one df instead of N .

Furthermore, 2 parameters equal 1.0, a boundary value. If they are fixed beforehand, they can be subtracted from the number of parameters (see McCutcheon, 1987), which likewise results in an increase in df . Finally, there is one more equality between Items 4 and 5 for Class 3. The probability of a correct answer on Item 4 is equal to the probability of getting a wrong answer on Item 5. So, in total, there are 13 free parameters, which means that we have $32 - 1 - 13 = 18$ degrees of freedom. If the model fits the data well, the likelihood ratio (L^2) and Pearson's

Table 2
A Three-Class Latent Class Model for Five Binary Items

Class	Proportion	Item 1	Item 2	Item 3	Item 4	Item 5
1	.50	.10	.21	.15	.18	.19
2	.30	.91	.99	.87	.67	.95
3	.20	1.0	.12	1.0	.05	.95

Note—The classes differ in proportion and in the conditional probabilities that specify the probability of a correct response to an item, given the class.

chi-square (χ^2) have an expected value of 18. The AIC ($AIC = L^2 + 2\#p$) has an expected value equal to 44.

Standard programs find ML estimates, given an a priori choice of model structure and constraints. To infer the model structure and optimize the parameters simultaneously, we could add a parameter for the number of classes and add parameters that represent constraints. This would lead, however, to a large number of parameters. Fortunately, it is also possible to represent the model structure in terms of the model parameters. Suppose that we have the following string of parameters (representing the model in Table 2): .50 .10 .21 .15 .18 .19 .60 .91 .99 .87 .67 .95 .50 1.0 .12 1.0 .05 .95 0.0 .33 .34 .56 .22 .11.

This string can be read from left to right as a string of parameters of four classes, including the latent class proportions (shown in *italic*). These proportions are transformed to their proper values (which sum to 1.0) according to Equations 3 and 4 below. If parameters that represent conditional probabilities are equal,² we count them as equalities. If parameters that represent conditional probabilities equal 0 or 1, they are counted as fixed parameters. If parameters that represent latent class proportions are 0, the class does not exist. This way, we have represented the model above with the number of classes and the constraints. Note that we do not take into account equalities between latent class proportions. These types of equalities are rarely applied in LCA.

Translation of parameter strings to latent class models and computation of the likelihood and AIC are rather simple. The evaluation of a string requires only an enumeration of the free parameters (the detection of equalities can easily be automated) and the computation of the likelihood ratio, which is a function of the observed and expected frequencies of the 2^n answer patterns. The only thing that needs to be specified a priori is the maximum number of classes. This is not a real limitation, since this maximum can be set to a high value. A minor complication is that latent class proportions have to sum to one. To ensure that this is the case, we set

$$Pc_i = Gc_i (1 - \sum_{j=1, i-1} Pc_j),$$

for $i = 1$ to $n-1$ for n classes (3)

and

$$Pc_n = 1 - \sum_{j=1, n-1} Pc_j,$$

for the last class, (4)

where $Gc_i \in [0,1]$ is the value of the parameter in the genetic string and Pc_i is the latent class proportion of class i . In this computation, the classes with a latent class proportion of zero are discarded. In the example string above, it can be easily seen that the first class has a proportion of .50, the second class has a proportion of .30 (i.e., .60 times the remaining .50), and the third class has the remaining proportion of .20. In computing the latter proportion, the .50 in the genetic string is not used; the last class, with a proportion of zero, is discarded altogether in computing the proportions.

We experimented with a number of different approaches. For example, we represented latent class models by a string of positive integers that define only the equalities and fixations on 0 and 1: 2 3 4 5 6 7 8 9 10 11 12 13 14 1 15 1 16 13 0 17 18 19 20 21, where 0 and 1 are fixations and equal integers represent equal parameter values. This representation could then be optimized by, for instance, the EM algorithm, which results in an L^2 and an AIC for each string of integers. In this case, we have a double optimization procedure. The GA optimizes the structure, and within the function evaluation, the EM algorithm optimizes the parameter values. A problem is that the EM algorithm needs good initial values. Random initial values will strongly influence the speed of convergence of the EM algorithm. Moreover, using this representation leads to many repeated calls of the EM algorithm, slowing the optimization process. In the first representation that is described above, these problems do not occur, and hence, we chose to use that representation in the implementation of GALCA that will be described in the following section.

The GALCA Program

There are many different implementations of GAs, and there exist many related techniques, such as evolutionary strategies and evolutionary programming (e.g., Bäck, 1996). The GA applied in the present article is a variant of Goldberg's (1989a) original implementation. This means that parameter values are coded and manipulated in binary form. For coding decimal numbers into binary numbers, we use the so-called *Gray code* (advantages over the binary code are described in, for instance, Caruana & Schaffer, 1988). The precision of the parameter estimates depends on the number of bits reserved for each parameter. For instance, a precision of three decimal points requires 10 bits.³ Mutation takes place by randomly changing bits, and crossing-over takes place by exchanging substrings of bits between two different strings. As in most implementations, our program works with a fixed population size (e.g., the number of individuals in each generation). We determine the number of offspring by a procedure based on ranks (Whitley, 1989) and a fitness distribution parameter (user specified, default $Fd = .9$). The probability f_i that a certain string is the parent of a specific individual in the next generation is

$$f_i = Fd^{rank_i}, \quad (5)$$

and

$$fitness_i = f_i / \sum_{i=1, n} f_i. \quad (6)$$

This ensures that fitness is a function of rank, with a slope depending on Fd (see Davis, 1989, for discussion). The GALCA program can also apply tournament selection to select offspring for the next generation.

In our GALCA program, a number of parameters have to be specified by the user. These include the number of individuals in the population (default = 100),⁴ the mutation rate P_{nc} ,⁵ crossing-over probability (default two-point crossing over with a probability of .7), and a spe-

cial parameter that determines sudden increases in the mutation rate, the so-called *hypermutation*.⁶ The term *hypermutation* stems from biological evolution. The idea is that the mutation rate temporarily increases when the selection pressure increases. Hypermution is applied in GAs for changing environments (Cobb & Grefenstette, 1993) or to solve the problem of premature convergence (e.g., Herrera & Lozano, 2000), as in our case. Related techniques for dealing with premature convergence are soft restart and random immigrants (Cobb & Grefenstette, 1993). All these techniques are aimed at reintroducing diversity into the population. Hypermution leads to the loss of the current best string and, consequently, to a temporary decrease in fitness (see Goldberg, 1989b). It appears that hypermutation often helps the GA to find better solutions—in particular, with the most difficult optimization problems (cf. Mathias, Schaffer, Eshelman, & Mani, 1998). Finally, we apply an additional mutation operator that copies parameters within parameter strings (i.e., the inversion operator; see Holland, 1975). Since equalities between parameters are important, this mutation operator is especially useful for the LCA problem. This mutation operator has its own probability parameter (default $P_{\text{copy}} = .01$).

We believe that none of these choices is essential. We could also use real-valued genes, tournament selection instead of fitness ranking, adaptive mutation rates, or sharing instead of hypermutation to prevent premature convergence (see Beasley et al., 1993a, 1993b, for an explanation of these options). From the literature on GA, it appears that differences in the performance of these options are not large. The general GA method is very robust. What is essential in this article is the representation of the latent class models as genetic strings, which we can easily apply to many existing implementations of GA.

We tested our GA on a benchmark problem suggested by Keane (1995a, 1995b). This typical multi-peaked problem with 50 parameters appears to be difficult for most optimization techniques. Keane (1995a) compared four methods: his own advanced GA, evolutionary programming (EP), evolutionary strategies (ES), and simulated annealing (SA). The function optimum is .8. The averages over five runs were GA = .779, EP = .673, ES = .578, and SA = .395. Using GALCA without hypermutation, we found an average over five runs of .743, and with hypermutation we found an average of .771. The effect of hypermutation was significant [$F(1,8) = 14.19$, $p \leq .01$].

Simulation Examples

Example A: An unrestricted two-class model for four binary items. Our first example concerns a simple unrestricted two-class model for four items. The data for 1,000 subjects were simulated according to the model in Table 3.

The data are the frequencies of the answer patterns 0000, 0001, . . . to 1111 ($N = 1,000$): 58 49 31 29 31 18

20 29 21 18 15 55 25 84 96 421. This model has an L^2 of 4.41, an χ^2 (Pearson's chi-square) of 4.28, and an AIC of 22.41. We performed two different optimizations: one in which L^2 was used as fitness, allowing for a maximum of two classes, and one in which the AIC was used, with a maximum of eight classes. The aim of the first optimization was to compare the GA with the EM algorithm with regard to standard parameter optimization. Constraints were not searched for in this optimization, because constraints always lead to a decrease in L^2 , which is the fitness criterion, instead of an increase. In the second optimization, we optimized the model structure in addition to the parameters. We allowed the GA to start with eight classes to see how long it would take before it reduced the number of classes to the correct value of two. In the latter optimization, constraints between parameters were allowed to occur.

The first optimization took about 75 generations with a population size of 144 to find the ML estimates. The top panel in Figure 1 shows the L^2 of the best individual per generation for three runs. The conclusion is that the GA can be used to optimize the parameters but that it is, as was expected, slower (about 10 sec on a G4 Macintosh) than the EM algorithm that solves this problem within 1 sec. In the second optimization, the GA started with eight classes and slowly reduced the number of classes. It took about 400 generations to find an AIC lower than 22.41. By then, it found models with two classes and one equality or models with more classes and more constraints, the best of which had an AIC of 18.92. These two examples show that the GA approach to LCA works quite well. We will now look at some more difficult cases.

Example B: A highly constrained seven-class model for four binary items. Our second example concerns a model with many classes, equalities, and boundary values, which is difficult to find "by hand" (i.e., checking all possible seven-class models for four binary items is not a practical option). We simulated data for four binary items on the basis of the model and parameter values shown in the upper part of Table 4.

The observed frequencies are ($N = 750$): 86 95 20 196 7 28 20 110 1 15 1 7 75 3 1 85 for the 16 possible response patterns, respectively. When the model shown in Table 4 was fitted to the data, using the EM algorithm, we found a χ^2 of 11.31, an L^2 of 11.58 ($df = 8$), an AIC of 25.58, and a BIC of 57.92. The largest possible unrestricted model has three classes, because a model with four or more classes would not be identified. The unrestricted three-class model does not fit the data: $L^2 = 91.13$, $\chi^2 = 89.11$, $df = 6$, including five parameters es-

Table 3
Latent Class Model Used to Simulate Data for Example A
(EM Estimates Are Shown in Parentheses)

Class	Proportion	Item 1	Item 2	Item 3	Item 4
1	.70 (.70)	.95 (.96)	.90 (.90)	.85 (.84)	.80 (.82)
2	.30 (.30)	.25 (.21)	.30 (.31)	.35 (.35)	.40 (.43)

Figure 1. Top panel: three runs of GALCA on the data from Example A. Middle panel: genetic search on the deterministic Proctor data. The best Akaike's information criterion (AIC) of each generation is shown. Bottom panel: genetic search on McCutcheon's (1987) data.

Table 4
Latent Class Model Used to Simulate Data for
Example B (Top) and the GALCA Solution (Bottom)

Class	Proportion	Item 1	Item 2	Item 3	Item 4
Simulation Model					
1	.10	0	0	0	0
2	.10	1	1	1	1
3	.10	0	0	1	1
4	.10	1	1	0	0
5	.20	.1	.1	.1	.9
6	.20	.1	.1	.9	.9
7	.20	.1	.9	.9	.9
GALCA Solution					
1	.113	.000	.053	.000	.000
2	.105	1.000	1.000	1.000	1.000
3	.100	.000	.000	1.000	1.000
4	.100	1.000	1.000	.000	.000
5	.166	.144	.053	.000	.947
6	.199	.053	.000	1.000	.856
7	.219	.053	1.000	.856	.856

timated at boundary values. We could try all kinds of constrained models, but if we have no clues about the model, it is very difficult to find a model that fits this data set.

We allowed the GA to fit a model with a maximum of eight classes. GALCA found a model with $L^2 = 1.37$, $\chi^2 = 1.46$, AIC = 17.37, BIC = 54.33, and $df = 7$. The parameter values of the resulting model are listed in the lower part of Table 4. The model does not differ substantially from the original model and fits slightly better than the original model. It appears that the GA method makes it possible to find such highly constrained models even when a priori knowledge is not available.

Example C: The Proctor model. Our third example is a Proctor model, which is a highly constrained scale model (Dayton, 1998; McCutcheon, 1987). We simulated data according to the parameter values shown in the first part of Table 5. In this model, the four items can be ordered on a scale from *difficult* to *easy*. The classes represent five scale values. There is only one conditional probability, which is called the *error of measurement*, which is .1. Hence, we have $1 + 4$ parameters, and so $df = 10$ (equalities between the latent class proportions are ignored).

We simulated data in two ways. In the deterministic approach, we computed the observed frequencies directly from the model parameters. In the stochastic approach, we constructed answer patterns by drawing from the multinomial distribution. In the first case, the expected L^2 equaled 0; in the second case, the expected L^2 was equal to df .

The deterministic data were ($N = 500$): 74 8 9 2 16 3 10 8 81 10 17 9 82 16 81 74. The AIC for the correct solution, which was extremely difficult to find, was 10.116. With an allowed maximum of six classes, GALCA takes many generations to find solutions with an AIC close to 10. Interestingly, GALCA found an even better fitting model after about 1,500 generations. The parameter values of this model are listed in the second part of Table 5.

The model has an L^2 of .116, an χ^2 of .119, $df = 11$, an AIC of 8.116, and a BIC of 24.975. Note that this model has one parameter fewer than the original model but exactly the same L^2 and, hence, a smaller AIC. As it turns out, this model is exactly equivalent to the Proctor model—that is, it generates the same probability for each answer pattern.⁷

The middle panel in Figure 1 depicts the optimization process. Within 200 generations, the GA found solutions with an AIC of about 23. After 1,300 generations, it found solutions with an AIC of approximately 10. About 200 generations later, it jumped to AIC = 8. The upward jumps in the values of the AIC were due to the hypermutation process.

In the second application of the Proctor model, we generated data stochastically. The data were ($N = 5,000$): 717 78 76 16 152 19 123 84 730 101 186 93 843 184 847 751. Fitting a Proctor model with the EM algorithm led to the ML parameter estimates shown in the third part of Table 5. This model had an L^2 of 13.05, with $df = 10$, AIC = 23.05, and BIC = 55.63.

Using GALCA with a maximum of six classes resulted in the model parameters listed in the fourth part of Table 5. The GA clearly had difficulties with this data set, seeing that it took 2,700 generations to find this solution. This model had an $L^2 = 3.462$, $\chi^2 = 3.420$, AIC = 19.462, BIC = 71.600, $df = 7$. Note that the BIC (we optimized the AIC) of the original Proctor model is lower and, thus, better.

Empirical Examples

In this section, the GALCA approach will be tested on a number of empirical data sets. Analyzing these data sets also illustrates some possible pitfalls of the use of GAs and how to avoid them.

Table 5
Example C

Class	Proportion	Item 1	Item 2	Item 3	Item 4
Simulation Model					
1	.20	.9	.9	.9	.9
2	.20	.9	.9	.9	.1
3	.20	.9	.9	.1	.1
4	.20	.9	.1	.1	.1
5	.20	.1	.1	.1	.1
GALCA Solution Deterministic Data					
1	.200	.900	.900	.900	.900
2	.400	.900	.900	.500	.100
3	.400	.500	.100	.100	.100
EM Solution Stochastic Data					
1	.204	.898	.898	.898	.898
2	.218	.898	.898	.898	.102
3	.208	.898	.898	.102	.102
4	.181	.898	.102	.102	.102
5	.190	.102	.102	.102	.102
GALCA Solution Stochastic Data					
1	.477	.888	.888	.805	.488
2	.024	.805	.000	.000	.839
3	.137	.000	.000	.079	.079
4	.362	.839	.488	.112	.000

Rindskopf's (1987) data. To illustrate the power of GAs in LCA, the first empirical data set that we analyzed was taken from Rindskopf (1987, Table 2). The items were hypothesized to form a hierarchy, and Rindskopf fitted scale and unrestricted models to the data. The data for four binary items were ($N = 78$): 7 16 2 13 0 6 0 7 0 0 2 0 0 0 25. Rindskopf tested four models, the best of which was a two-class model with an AIC of 17.86.

Using GALCA (with a maximum number of six classes allowed), we found a three-class model with an AIC of 10.08 ($L^2 = \chi^2 = 0.086$, $df = 10$, $BIC = 21.87$). The GALCA parameter estimates are presented in Table 6. Clearly, the model fits much better than the models proposed by Rindskopf (1987); it has fewer parameters and may be interpreted as a kind of scale model (see, e.g., McCutcheon, 1987).

Feick's (1987) data. This illustration was based on the analysis of data from Feick (1987, Table 3). The data for five binary items were ($N = 443$): 58 1 1 0 12 1 3 0 22 0 1 0 17 0 0 0 80 3 7 3 43 2 7 2 61 2 10 2 75 6 18 6. Feick tested the models listed in Table 7.

The reader is referred to Feick (1987) for an explanation of the exact meaning of the models. Feick prefers Model 14, which has the lowest AIC, although he admits that a choice is difficult, since the majority of the models fit the data according to the L^2 criterion. We used GALCA with AIC as fitness criterion, which resulted in a much better model still. The goodness-of-fit indices for this model are also listed in Table 7. The estimated parameter values are provided in Table 8.

This model has roughly the structure of a scaling model, as was expected by Feick (1987). The low AIC and BIC of this model are partly due to equalities that do not make much sense [for instance, $p(1,1) = 1 - p(2,1)$ and $p(3,2) = 1 - p(4,5)$]. Mooijart and van der Heijden (1992) have distinguished between four types of constraints. The fourth type, equalities in different variable-latent-class combinations, is too difficult to optimize with EM, in contrast to an NR algorithm. The GA method often finds these type of equalities, which can be an advantage, as compared with EM. However, it may be useful to allow the GA to count only equalities rowwise and columnwise and to exclude equalities in different variable-latent-class combinations, since these are often hard to interpret.⁸ Excluding equalities of the type $a = 1 - b$ results in the parameter estimates in the lower part of Table 8, which has the following goodness-of-fit indices: $\chi^2 = 10.51$, $L^2 = 10.95$, $AIC = 26.95$, $BIC = 59.69$, and $df =$

23. This model fits the data well and much better than the models of Feick do.

This example demonstrates the importance of having several options concerning the types of constraints that are allowed in inferring model structure with GAs. Moreover, by using GAs, arbitrary types of constraints can be optimized, which is not possible when latent class models are optimized using the EM algorithm.

McCutcheon's (1987) data. Our third empirical example was adapted from McCutcheon (1987). The data, based on four binary scaling items, on which McCutcheon fitted a number of scale models, were ($N = 1,402$): 27 40 16 339 2 32 4 543 0 3 0 83 0 2 1 310. Table 9 lists the models that McCutcheon tested, along with the associated goodness-of-fit measures.

Only the last two models in McCutcheon (1987) fit the data (L^2 approximately equals df), of which, according to the AIC and BIC, the bifactor scale with Type 2 excluded was the best model. We did two runs of GALCA, one with the AIC and one with the BIC as the fitness criterion. The goodness-of-fit measures of the resulting models are shown at the bottom in Table 9. The bottom panel in Figure 1 shows the run based on AIC. In this run, the GA very quickly found AICs below 24.77 and even below 20. After more than 3,000 generations, however, the GA found its best solution. This shows that it is important to let the GA run for a long time. This run, using the AIC, led, after 3,425 generations, to the model displayed in the top half of Table 10. In eight replication runs (10,000 generations each), the GA found solutions very close to the present solution (mean $AIC = 16.05$ and $SD = 0.65$).

The GA AIC model is a modified scale model similar to models tested by McCutcheon (1987). Class 1 combines subjects that solve all items and those that solve all but one item. Classes 2, 3, and 4 form the other scale values. Besides extreme conditional probabilities for correct and incorrect answers, it has an intermediate probability of .390. In the bottom half of Table 10, the GA BIC model parameters are listed; this model combines the last two scale values instead of the first two, and it also has intermediate probabilities.

The improvement in fit of the GA models, as compared with the models tested by McCutcheon (1987), is impressive. The L^2 and the χ^2 are much lower than those of the other models, whereas df is large. The GA BIC model (Table 10) uses the same number of parameters as the highly constrained Proctor model, which does not fit. Moreover, the AICs and BICs are much lower.

Overfitting or Chance Capitalization

There are three possible explanations for GA's superior fit. The first is simply that GALCA found a superior model, the second has to do with the simplistic assessment of model complexity, and third concerns overfitting or capitalization on chance. The way we enumerate the free parameters is open to criticism. Our correction

Table 6
GALCA Solution for Rindskopf's (1987) Data

Class	Proportion	Item 1	Item 2	Item 3	Item 4
1	.345	1.000	.928	1.000	1.000
2	.390	.000	.431	.569	1.000
3	.265	.000	.000	.234	.569

Table 7
The Fit of Various Latent Class Models to the Data in Feick (1987)

Model	<i>df</i>	L^2	χ^2	AIC	BIC
Probabilistic Models					
1. Independence	26	106.3	142.06	116.28	136.75
2. Uniform error	25	86.34	75.92	98.34	122.90
3. False positive/ false negative	24	44.72	36.87	58.72	87.37
4. Item-specific error	21	23.88	22.27	43.88	84.82
5. Latent distance	18	18.3	14.89	44.30	97.52
Intrinsically Unscalable					
6. Goodman scale	20	27.63	29.67	49.63	94.66
7. Goodman uniform	19	27.6	30.03	51.60	100.72
8. Goodman false positive/ false negative	18	19.17	19.05	45.17	98.39
9. Goodman item specific	15	16.35	16.72	48.35	113.85
Models for Ordering					
10. Item-specific error	21	42.11	41.25	62.11	103.05
11. Latent distance	18	19.06	17.65	45.06	98.28
Biform Scale Model					
12. Goodman scale	19	24.03	24.14	48.03	97.15
Unrestricted Model					
13. Three class	14	12.47	14.76	46.47	116.06
Characteristic Models					
14. Latent distance	19	18.32	15.04	42.32	91.44
15. Item-specific error	24	72.27	71.79	86.27	114.92
GA AIC model ($a = 1 - b$)	24	12.61	13.81	26.61	55.26

Note—GA, genetic algorithm; AIC, Akaike's information criterion.

for boundary values is rather simplistic (see Shapiro, 1985), although commonly applied in LCA. Moreover, one could argue that we have to add parameters for the fact that we allow the number of classes and the number of constraints to vary freely. This needs further analysis, but we do not think that this completely explains the superior fit. Overfitting or capitalization on chance, however, constitutes a real problem for all kinds of exploratory techniques. LCA is known to be a rather exploratory technique, especially when it is applied in the absence of a priori models. It is clear that the GALCA method is even more susceptible to the danger of capitalization on chance. Fortunately, it is possible to detect and study overfitting by cross-validation.

Cross-validation. To test for overfitting, we performed a cross-validation on McCutcheon's (1987) data, comparing his and our models. In the analyses, McCutcheon's data were randomly split into two halves, and the model was fitted on one half only. Next, all the parameter values were fixed at their best-fitting values, and the resulting models were tested against the other half of the data. To obtain reliable results, this procedure was repeated at least 20 times with different random splits of the data. In the upper part of Table 11, the fit (on the first half) and cross-validation (on the second half) for the eight models in McCutcheon are reported. The means and standard deviations of the main fit indices are displayed. Note that no parameters have been estimated in the case of cross-validation, so *df* equals 15 in all cases.

For GALCA we report two cross-validations. First, we cross-validate the models presented in Table 10, which were found with all the data. Second, we ran GALCA again with only one random half of the data, instead of all the data, for both the AIC and the BIC as fitness criterion. The fit and cross-validation results are reported at the bottom of the table. The first GALCA cross-validation compares better with the McCutcheon (1987) models, which were also constructed using all the data, but the second GALCA cross-validation gives a stricter test of overfitting in GALCA.

As can be seen in Table 11, McCutcheon's (1987) models perform well in the cross-validation procedure (with the exception of the intrinsically unscalable model). Note that with 15 degrees of freedom, the chi-square be-

Table 8
GALCA Solutions for Feick's (1987) Data

Class	Proportion	Item 1	Item 2	Item 3	Item 4	Item 5
With $a = 1 - b$ Equalities						
1	.120	1.000	.690	.690	.690	.310
2	.180	1.000	.690	.690	.070	.070
3	.500	.690	.550	.450	.070	.020
4	.200	.450	.070	.000	.000	.020
Without $a = 1 - b$ Equalities						
1	.219	1.000	.654	.654	.535	.208
2	.385	.803	.654	.654	.000	.000
3	.098	.535	.208	.535	.208	.120
4	.297	.535	.208	.000	.000	.000

Table 9
The Fit of Various Latent Class Scale Models
to the Data in McCutcheon (1987)

Model	<i>df</i>	<i>L</i> ²	χ^2	AIC	BIC
Proctor model	10	138.20	137.27	148.19	174.42
Item-specific error rates	7	36.63	36.77	52.63	94.60
True-type-specific error rates	7	89.02	86.54	105.02	146.99
Lazarsfeld's model	5	14.79	12.38	34.79	87.25
Intrinsically unscalable model	6	20.55	17.51	38.55	85.76
Proctor Goodman model	5	20.63	16.31	40.63	93.09
Biform scale	5	6.77	6.62	26.77	79.23
Biform scale with Type 2 excluded	6	6.77	6.62	24.77	71.98
GA AIC model	8	1.61	1.61	15.61	52.33
GA BIC model	10	6.35	5.90	16.30	42.58

Note—The last two models were found with GALCA, the first using AIC as fitness criterion, the second using BIC. GA, genetic algorithm; AIC, Akaike's information criterion; BIC, Bayesian information criterion.

comes significant above 25. Furthermore, for the purpose of cross-validation, the AIC seems to more useful than the BIC, since the AICs for the fit and the cross-validation are often very similar (cf. Stone, 1977).

The cross-validation of the GALCA models reported in Table 10 shows that the model found by optimizing the BIC suffers from overfitting, whereas the cross-validation fit of the AIC model is clearly much better [$t(61.4) = -3.07, p = .003$]. The cross-validation of the GALCA models, based on only one half of the data, point to a much stronger effect of overfitting than that found in the first analyses. Also, there is no indication in this cross-validation that the AIC is better in preventing overfitting than the BIC is.

We do not have a good solution for this problem, but we have added a cross-validation option to the program. If this option is turned on, the data are split (default, 50% split), and the final fit is also evaluated on the second part of the data. Perhaps overfitting can be prevented by using procedures such as bootstrapping (Efron & Tibshirani, 1993), or built-in cross-validation (Westbury, Buchanan, Sanderson, Rhemtulla, & Phillips, 2003) within GALCA. In such a bootstrap procedure, each string in each generation is evaluated on a different sample of the data (default, 90% of the data). This way, overfitting is much more difficult (see Baumann, 2003, for a comparable approach). However, this will increase the computational load a lot, and we have not found any conclusive evidence that it really works. Another possible solution is *backwarding* (Robilliard & Fonlupt, 2001). This simply means going backward in the evolution process in order to retrieve models that do overfit the data. Of course, this also requires a split of the data into two parts. We have looked at this solution, and it certainly has potential. Further research along this line may lead to measures of overfitting that indicate when to stop the GALCA run.

Discussion

Inferring model structure is a notorious problem in LCA, as it is in fitting all kinds of exploratory models to

psychological data sets. We have shown that GAs may be able to alleviate this problem, since they provide an efficient means of searching the space of possible model structures while simultaneously optimizing model parameters. We showed that it is possible to optimize both the parameters and the model structure, using a GA in which the AIC (or the BIC) features as a fitness criterion. We applied a simple but effective representation of latent class models. For three simulated data sets and three empirical data sets, we showed that GALCA successfully optimizes the models. In all three empirical examples, it found better (and plausible) models than those produced by the extensive original analyses. In the case of the Proctor model, an added bonus was the finding of a statistically equivalent, previously unknown model—that is, a reparametrization of the Proctor model. This is significant because finding such reparametrizations is generally a very difficult problem, as it is, for example, in covariance structure analysis (MacCallum et al., 1993; Pearl, 2000).

A restriction of the present implementation of GALCA is that it works only with binary observables, whereas other LCA programs allow for polytomous items—that is, items with more answer options. Extending GALCA to polytomous items does not require a change in representation or any other conceptual changes. If we allow for more than two values of the items, we encounter a similar problem as with the latent class proportions. The conditional probability on the latest answer category is redundant. However, applying similar equations to Equations 3 and 4 provides a nice solution that incorporates the constraint that conditional probabilities should sum to unity. Other limitations are inherent to GAs. They do not guarantee the best solution, and they are relatively slow.⁹ An inherent disadvantage of the present method is the risk of capitalization on chance. To some degree, this is true for all (explorative) methods of model fitting, but especially for GA and related methods. A general solution for overfitting is cross-validation. In cases in which GALCA is used as the only method to find latent class models, cross-validation is strongly recommended. Cross-validation can automatically be performed by GALCA.

Table 10
GALCA Solutions for McCutcheon's (1987) Data

Class	Proportion	Item 1	Item 2	Item 3	Item 4
Fitness: AIC					
1	.534	.390	.995	1.000	.995
2	.341	.193	.193	1.000	1.000
3	.091	.075	.390	.390	1.000
4	.033	.000	.075	.390	.000
Fitness: BIC					
1	.008	.880	1.000	1.000	.880
2	.484	.444	.880	1.000	1.000
3	.468	.120	.444	.880	1.000
4	.039	.000	.120	.444	.120

Note—In the top panel, Akaike's information criterion (AIC) was used as the fitness criterion; in the bottom panel, the Bayesian information criterion (BIC) was used.

Table 11
Results of the Cross-Validation Study

Results of the Cross-Validation Study									
Model	df	L^2		χ^2		AIC		BIC	
		M	SD	M	SD	M	SD	M	SD
McCutcheon (1987) Models									
Proctor model	10	71.7	8.0	68.9	9.4	81.7	8.0	104.4	8.0
Cross-validation	15	83.6	7.1	86.2	13.7	83.6	7.1	83.6	7.1
Item-specific error rates	7	21.6	6.3	20.1	6.0	37.6	6.3	74.1	6.3
Cross-validation	15	36.2	10.5	39.4	17.2	36.2	10.5	36.2	10.5
True-type-specific error rates	6	47.1	6.8	45.0	7.0	65.1	6.8	106.1	6.8
Cross-validation	15	60.5	7.1	62.3	11.4	60.5	7.1	60.5	7.1
Intrinsically unscalable model	7	9.5	3.3	7.8	2.2	25.5	3.3	61.9	3.3
Cross-validation	15	49.2	22.8	45.5	29.3	49.2	22.8	49.2	22.8
Lazarfeld's model	6	13.6	4.1	11.6	3.8	31.6	4.1	72.6	4.1
Cross-validation	15	26.5	6.5	27.0	7.9	26.5	6.5	26.5	6.5
Proctor Goodman model	5	14.0	4.4	11.3	3.8	34.0	4.4	79.5	4.4
Cross-validation	15	29.7	6.5	29.6	7.8	29.7	6.5	29.7	6.5
Biform scale	5	5.8	2.7	5.3	2.4	25.8	2.7	71.4	2.7
Cross-validation	15	22.9	8.7	24.2	10.8	22.9	8.7	22.9	8.7
Biform scale without Type 2	6	5.9	2.7	5.4	2.4	23.9	2.7	64.9	2.7
Cross-validation	15	22.3	7.9	23.4	9.3	22.3	7.9	22.3	7.9
Models in Table 10									
GA AIC model	8	3.6	1.7	3.1	1.5	17.6	1.7	49.5	1.7
Cross-validation	15	23.3	19.8	16.1	9.3	23.3	19.8	23.3	19.8
GA BIC model	10	6.9	3.4	5.7	2.5	16.9	3.4	39.7	3.4
Cross-validation	15	38.9	25.5	14.6	6.5	38.9	25.5	38.9	25.5
Complete Cross-Validation									
GA AIC cross-validation run	9.7 (.8)	3.7	1.7	3.4	1.2	14.4	2.0	38.6	4.9
Cross-validation	15.0	38.8	20.8	31.2	16.5	38.8	20.8	38.8	21.4
GA BIC cross-validation run	10.5 (.8)	11.7	6.9	10.7	6.3	20.7	5.8	41.3	4.5
Cross-validation	15.0	43.8	19.6	40.6	25.4	43.8	18.7	43.8	16.9

Note—GA, genetic algorithm; AIC, Akaike's information criterion; BIC, Bayesian information criterion.

Nevertheless, using the GA when one has to perform LCA, is useful for (1) finding models when a priori models are absent, (2) checking whether better models exist than the a priori models, (3) optimizing the fit function in cases in which computation of initial parameter values is difficult, and (4) getting a general idea of the distribution of the AIC (or any other desired statistic) over a number of model structures.

We did not provide an example of the third advantage, computation of initial parameter values. The other optimization methods in LCA are known to be sensitive to initial parameter values in many cases. GAs do not have this problem, since they always start with random populations (Beasley et al., 1993a). The fourth advantage is demonstrated in the first two empirical examples and also in the last simulation example, in which a better-fitting model was found than the one used to simulate the data. In the empirical examples, the GALCA runs indicate the minimum possible value for the AIC and show how important differences in AIC between a priori models are.

GALCA can be extended in several ways. We have already mentioned polytomous items and restrictions on

the types of equalities that are allowed. Other possibilities are improvements of the GA-like mutation operators based on the EM algorithm, sharing, and more advanced crossover techniques. Finally, it might be of interest to extend this technique to other statistical models, such as finite mixture models, item response models, hidden Markov models, factor models, and possibly others.

REFERENCES

- BÄCK, T. (1996). *Evolutionary algorithms in theory and practice*. New York: Oxford University Press.
- BAUMANN, K. (2003). Cross-validation as the objective function for variable selection. *Trends in Analytical Chemistry*, **22**, 395-406.
- BEASLEY, D., BULL, D. R., & MARTIN, R. R. (1993a). An overview of genetic algorithms: Pt. 1. Fundamentals. *University Computing*, **15**, 58-69.
- BEASLEY, D., BULL, D. R., & MARTIN, R. R. (1993b). An overview of genetic algorithms: Pt. 2. Research topics. *University Computing*, **15**, 170-181.
- CARUANA, R. A., & SCHAFER, J. D. (1988). Representation and hidden bias: Gray vs. binary coding for genetic algorithms. In J. E. Laird (Ed.), *Proceedings of the 5th International Conference on Machine Learning* (pp. 153-161). San Mateo, CA: Morgan Kaufmann.
- CLOGG, C. C. (1995). Latent class models. In G. Arminger, C. C. Clogg,

- & M. E. Sobel (Eds.), *Handbook of statistical modeling for the social and behavioral sciences* (pp. 311-360). New York: Plenum.
- COBB, H., & GREFFENSTETTE, J. J. (1993). Genetic algorithms for tracking changing environments. In S. Forest (Ed.), *Proceedings of the 5th International Conference on Genetic Algorithms* (pp. 523-530). San Mateo, CA: Morgan Kaufmann.
- DAVIS, L. (1989). Adapting operator probabilities in genetic algorithms. In J. D. Schaffer (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms* (pp. 61-69). San Mateo, CA: Morgan Kaufmann.
- DAYTON, C. M. (1998). *Latent class scaling analysis*. Thousand Oaks, CA: Sage.
- DEMPSTER, A. P., LAIRD, N. M., & RUBIN, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society B*, **39**, 1-38.
- EFRON, B., & TIBSHIRANI, R. J. (1993). *An introduction to the bootstrap*. New York: Chapman & Hall.
- FEICK, L. F. (1987). Latent class models for the analysis of behavioral hierarchies. *Journal of Marketing Research*, **24**, 174-186.
- GOLDBERG, D. E. (1989a). *Genetic algorithms in search, optimization and machine learning*. Reading, MA: Addison-Wesley.
- GOLDBERG, D. E. (1989b). Zen and the art of genetic algorithms. In J. D. Schaffer (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms* (pp. 80-85). San Mateo, CA: Morgan Kaufmann.
- HAERTEL, E. H. (1989). Using restricted latent class models to map the skill structure of achievement items. *Journal of Educational Measurement*, **26**, 301-321.
- HAGENAARS, J. A., & MCCUTCHEON, A. L. (2002). *Applied latent class analysis*. Cambridge: Cambridge University Press.
- HEINEN, T. (1996). *Latent class and discrete latent trait models: Similarities and differences*. London: Sage.
- HERRERA, F., & LOZANO, M. (2000). Two-loop real-coded genetic algorithms with adaptive control of mutation step sizes. *Applied Intelligence*, **13**, 187-204.
- HOLLAND, J. H. (1975). *Adaptation in natural and artificial systems*. Cambridge, MA: MIT Press.
- JANSEN, B. R. J., & VAN DER MAAS, H. L. J. (1997). Statistical test of the rule assessment methodology by latent class analysis. *Developmental Review*, **17**, 321-357.
- KEANE, A. J. (1995a). A brief comparison of some evolutionary optimization methods. In V. Rayward-Smith, I. Osman, C. Reeves, & G. D. Smith (Eds.), *Modern heuristic search methods* (pp. 255-272). New York: Wiley.
- KEANE, A. J. (1995b). Genetic algorithm optimization of multipeak problems: Studies in convergence and robustness. *Artificial Intelligence in Engineering*, **9**, 75-83.
- KWONG, S., CHAU, C. W., MAN, K. F., & TANG, K. S. (2001). Optimization of HMM topology and its model parameters by genetic algorithms. *Pattern Recognition*, **34**, 509-522.
- MACCALLUM, R. C., WEGENER, D. T., UCHINO, B. N., & FABRIGAR, L. R. (1993). The problem of equivalent models in applications of covariance structure analysis. *Psychological Bulletin*, **114**, 185-199.
- MATHIAS, K. E., SCHAFFER, J. D., ESHELMAN, L. J., & MANI, M. (1998). The effects of control parameters and restarts on search stagnation in evolutionary programming. In A. Eiben (Ed.), *Parallel problem solving from nature* (pp. 398-407). Berlin: Springer-Verlag.
- MCCUTCHEON, A. L. (1987). *Latent class analysis* (Sage university paper series on quantitative applications in the social sciences, No. 07-064). Beverly Hills, CA: Sage.
- MITCHELL, M. (1996). *An introduction to genetic algorithms*. Cambridge, MA: MIT Press.
- MOOIJAAART, A., & VAN DER HEIJDEN, P. G. M. (1992). The EM algorithm for latent class analysis with equality constraints. *Psychometrika*, **57**, 261-269.
- MYUNG, I. J., FORSTER, M. R., & BROWNE, M. W. (2000). Special issue on model selection. *Journal of Mathematical Psychology*, **44**, 1-2.
- PEARL, J. (2000). *Causality: Models, reasoning, and inference*. Cambridge: Cambridge University Press.
- RINDSKOPF, D. (1987). Using latent class analysis to test developmental models. *Developmental Review*, **7**, 66-85.
- ROBILLIARD, D., & FONLUP, C. (2001). Backwarding: An overfitting control for genetic programming in a remote sensing application. In P. Collet, C. Fonlup, J. K. Hao, E. Lutton, & M. Schoenauer (Eds.), *Artificial Evolution 5th International Conference, Evolution Artificielle, EA 2001* (pp. 245-254). Berlin: Springer-Verlag.
- ROST, J., & LANGEHEINE, R. (Eds.) (1997). *Applications of latent trait and latent class models in the social sciences*. Münster: Waxmann Verlag.
- SHAPIRO, A. (1985). Asymptotic distribution of test statistics in the analysis of moment structures under inequality constraints. *Biometrika*, **72**, 133-144.
- SLIMANE, M., VENTURINI, G., ASSELIN DE BEAUVILLE, J.-P., BROUARD, T., & BRANDEAU, A. (1996). Optimizing hidden Markov models with a genetic algorithm. In J. M. Alliot, E. Lutton, E. Ronald, M. Schoenauer, & D. Sayers (Eds.), *Artificial Evolution: European Conference, AE. Selected Papers* (No. 1063, pp. 384-396). Berlin: Springer-Verlag.
- SUN, F., & HU, G. R. (1998). Speech recognition based on genetic algorithm for training HMM. *Electronics Letters*, **34**, 1563-1564.
- STONE, M. (1977). An asymptotic equivalence of choice of model by cross-validation and Akaike's criterion. *Journal of the Royal Statistical Society B*, **38**, 44-47.
- WESTBURY, C., BUCHANAN, L., SANDERSON, S., RHEMTULLA, M., & PHILLIPS, L. (2003). Using genetic programming to discover nonlinear variable interactions. *Behavior Research Methods, Instruments, & Computers*, **35**, 202-216.
- WHITLEY, D. (1989). The Genitor algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In J. D. Schaffer (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms* (pp. 116-121). San Mateo, CA: Morgan Kaufmann.

NOTES

1. In other work on hidden Markov models, GAs have been used instead of the EM algorithm—that is, GAs were used to optimize parameter values within a fixed model structure (Slimane, Venturini, Asselin de Beauville, Brouard, & Brandeau, 1996; Sun & Hu, 1998).

2. These probabilities are represented as bitstrings, and hence equalities can be determined easily by checking equality of bits. See the next section for implementational details.

3. In our GALCA program, the user has to choose both the number of bits and the percentage of substrings that are used to represent the boundary values (i.e., 0 and 1). For instance, with 12 bits we can represent 4,096 different numbers. If the user sets the percentage reserved for 0 and 1 at 5%, 0-101 is used to represent 0, 102 to 3,993 is used for (0,1), and 3,994-4,095 represent 1. This way, the user can influence the frequencies of 0 and 1 in the parameter string. This option is helpful only in extreme cases. The default percentage is 1%. If the parameter is set to zero, the distribution is uniform on [0,1].

4. The program allows much larger population sizes (up to 50,000, depending on available RAM). There is, however, a tradeoff between population size and the number of generations that can be computed in a given amount of time. In our experience, the choice of this tradeoff is not so important.

5. Instead of the mutation rate, the user of GALCA has to set another probability: the probability that a string will not be changed by mutation (default $P_{\text{no change}} = .01$). This way, the mutation rate is adjusted automatically to the length of the strings of parameters. We use the following formula:

$$P_{\text{mutation}} = 1.0 - (P_{\text{no change}})^{1/\text{\#bits}}.$$

6. In our GA, this process is regulated by three parameters. The first two determine the criterion for convergence. The *window of convergence* is the number of generations over which convergence is tested. When the increase in fitness within the window is monotone but stays below the convergence criterion, the hypermutation process is started. This means that the mutation rate is strongly increased and then slowly decreased until it reaches its normal level. The strength of the effect is determined by the hypermutation factor (0.4 is strong, 0.8 is middle, 1.4 is weak) as follows:

If convergence, then

$$r = 0.$$

Each following generation:

$r = r + \text{hypermutation factor}$

$$P_{\text{mutation}} = P_{\text{mutation}} + \{1.0 - [P_{\text{no change}} / (\text{population size})]^{1/\#\text{bits}}\} / r.$$

As can be seen, the hypermutation effect starts suddenly and then decreases slowly until the normal mutation rate is reached.

7. Following this general pattern, we found equivalents of Proctor models of up to 10 classes. The 4-class Proctor model reduces to 2 classes, the 6-class model reduces to 3 classes, and the 10-class Proctor model (i.e., one with nine items) reduces to a 5-class model. In each case, the model has one parameter less than the original model. Note that in this example, 2 classes are joined together, and 1 class remains identical—namely, Class 1. This is reminiscent of an identification

problem described in Haertel (1989), where 2 classes cannot be distinguished because they have item-specific error rates. In the case described here, however, there is no identification problem, but a more parsimonious model is possible.

8. To minimize the risk of overfitting, one might consider counting equalities only when they involve all the parameters within a class or an item.

9. The output model of the GA can be subjected to an additional optimization with an EM algorithm, which is available in the GALCA software.

(Manuscript received August 29, 2003;
revision accepted for publication August 8, 2004.)