Pergamon

PII:S0952-1976(96)00018-3

Engng Applic. Artif. Intell. Vol. 10, No. 5, pp. 473–483, 1997

0952-1976/97 $17.00+0.00

Contributed Paper

Evolutionary Neural Trees for Modeling and Predicting Complex Systems

BYOUNG-TAK ZHANG
German National Research Center for Information Technology (GMD), Sankt Augustin, Germany

PETER OHM
German National Research Center for Information Technology (GMD), Sankt Augustin, Germany

HEINZ MÜHLENBEIN
German National Research Center for Information Technology (GMD), Sankt Augustin, Germany

(Received October 1996)

Modeling and predicting the behavior of many technical systems is complicated because they are generally characterized by a large number of variables, parameters and interactions, and limited amounts of collected data. This paper investigates an evolutionary method for learning models of such systems. The models thus evolved are based on trees of heterogeneous neural units. The set of different neuron types is defined by the application domain, and the specific type of each unit is determined during the evolutionary learning process. The structure, size, and weights of the neural trees are also adapted by evolution. Since the genetic search used for training does not require error derivatives, a wide range of neural models can be constructed. This generality is contrasted with various existing methods for complex system modeling, which investigate only restricted topological subsets rather than the complete class of architectures. An improvement in the predictive accuracy and parsimony of models is reported, against backpropagation networks and other well-engineered polynomial-based methods for two problems: MacKey–Glass and Lorenz-like chaotic systems. The authors also demonstrate the importance of the selection pressure towards model parsimony for the improvement of prediction accuracy.

Keywords: Evolutionary algorithms, neurocomputing, evolutionary neural trees, machine learning, system identification, complex systems, time series prediction.

1. INTRODUCTION

Modeling and predicting the behavior of many technical systems is difficult because these systems are often complex. They are generally characterized by a large number of variables, parameters and interactions, and limited amounts of collected data. System-identification techniques are used in many engineering fields in order to predict the behavior of unknown systems. Assume that the single-valued output y of an unknown system behaves as a function of n input values, i.e.

$$y=f(\mathbf{x}_1,x_2,\ldots,x_n). \tag{1}$$

Given a set of input–output observations, the system-identification task is to build a model $\hat{f}$ of the true function f. Once this model has been learned, a predicted output $\hat{y}$ can be found for any input vector $(x_1,x_2,\ldots,x_n)$:

$$\hat{y}=\hat{f}(x_1,x_2,\ldots,x_n). \tag{2}$$

An example of system identification is time-series prediction, i.e. predicting future values of a variable from its previous values. Past values $x(t-1),x(t-2),\ldots,x(t-\phi)$ can be considered as an input vector to the unknown system. Once a model $\hat{f}$ of the system has been constructed, the future values $x(t+\tau)$ are predicted as a function of previous values:

Correspondence should be sent to: Dr Byoung-Tak Zhang, German National Research Center for Information Technology (GMD), Schloss Birlinghoven, D-53754 Sankt Augustin, Germany [E-mail: zhang@borneo.gmd.de].

$$x(t+\tau)=f(x(t-1),x(t-2),\ldots,x(t-\phi)) \quad (3)$$

where τ is the distance into the future where the model will be trained to predict, and ϕ is the maximum past value.

Most system-identification techniques are based on parameter and function estimates. Unfortunately these earlier approaches suffered from combinatorial explosion as the number of training data and parameters increased. One of these approaches is a heuristic algorithm called the Group Method of Data Handling (GMDH) (Ivakhnenko, 1971). However, this method suffers from local extreme problems, limiting its application (Tenorio and Lee, 1990). Iba *et al.* (1993) have recently shown that the weaknesses of the GMDH approach can be largely overcome by combining GMDH with a genetic algorithm.

Genetic algorithms (or "evolutionary" algorithms) are search methods based on a population of individuals, each of which represents a search point in the space of potential solutions to a given problem (Goldberg, 1994; Fogel, 1995; Koza, 1992). The population is arbitrarily initialized, and it evolves toward better and better regions of the search space by means of randomized processes of selection, mutation and recombination. The environment delivers the fitness values of the search points, and the selection process favors those individuals of higher fitness, to reproduce more often than those of lower fitness. The recombination mechanism allows the mixing of parental information while passing it to their descendants, and mutation introduces innovation into the population.

This paper presents an evolutionary method that uses neural trees for modeling and predicting the behavior of complex technical systems. Neural trees are tree-structured representations consisting of artificial neurons (Zhang *et al.*, 1995a, 1995b). Unlike most conventional neural-network models, neural trees employ different types of neurons in a single model. The set of different types is defined by the application domain, and the specific type of each unit is determined during the evolutionary learning process. Any arbitrary units can be employed, since the training algorithm used, i.e. the breeder genetic algorithm (Mühlenbein and Schlierkamp-Voosen, 1994), makes no assumptions as to the differentiability of the activation function. The structure and size of the trees are also automatically adapted by evolution.

Section 2 briefly introduces the neural trees. Section 3 describes the genetic search algorithm for evolving problem-specific neural-tree models of unknown complex systems. Section 4 presents a regularization technique for evolving parsimonious trees. In Sections 5 and 6 the method is applied to the prediction of two chaotic technical systems. The predictive accuracy and parsimony of the models are compared with those of backpropagation neural networks and GMDH-based system identification methods. Implications of the current work are discussed in Section 7.

2. NEURAL TREES

A neural tree consists of nonterminal nodes and terminal nodes. The nonterminal nodes represent neural units, and the neuron type is an element of the basis function set $\mathcal{F}=\{\text{neuron types}\}$. Each terminal node is labeled with an element from the terminal set $\mathcal{T}=\{x_1,x_2,\ldots,x_n\}$, where x_i is the ith component of the external input x. Each link (j,i) represents a directed connection from node j to node i, and is associated with a value w_{ij}, called the synaptic weight. The root node is also called the output unit, and the terminal nodes are called input units. The rest are hidden units. The layer of a node is defined as the longest path length among the paths to the terminal nodes of its subtree.

Different neuron types compute different net inputs. One of the most popular neuron types is the sigma unit, which computes the sum of weighted inputs from the lower layer:

$$net_i=\sum_{j\in R(i)} w_{ij}y_j \quad (4)$$

where y_j are the inputs from the receptive field $R(i)$ to the ith neuron. Another useful, but not so popular, neuron type is the pi unit, which calculates the product of weighted inputs:

$$net_i=\prod_{j\in R(i)} w_{ij}y_j \quad (5)$$

where y_j are the inputs to i. One might also define "max" units for flexible logical operations:

$$net_i=w_{ij^*}y_{j^*}=\max_{j\in R(i)} w_{ij}y_j \quad (6)$$

where j^* is the unit that maximizes the weighted input $w_{ij}y_j$. Likewise, "min" units can be defined. The output of the neurons is computed by either the threshold response function

$$y_i=\sigma(net_i)=\begin{cases}1 & : \ net_i\geq 0\\ -1 & : \ net_i<0\end{cases} \quad (7)$$

or the sigmoid transfer function

$$y_i=f(net_i)=\frac{1}{1+e^{-net_i}} \quad (8)$$

where net_i is the net input to the unit.

An instance of the neural tree is shown in Fig. 1. The tree is implemented as an embedded list:

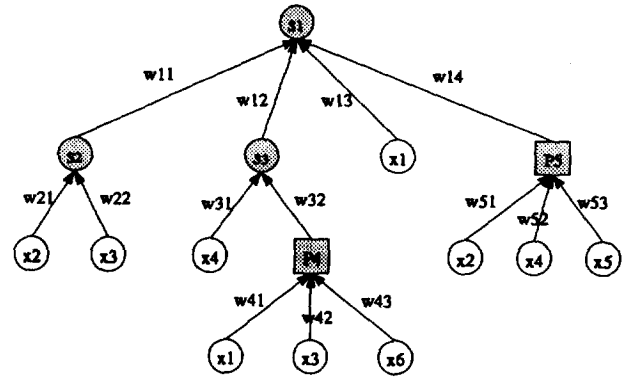

Fig. 1. The structure of a neural tree. The tree consists of one output unit (root), four hidden units (two different neuron types), and six input units ($x_1,\ldots,x_6$) distributed over terminal nodes.

```
procedure EvolvingNeuralTrees(M, τ, LS_max)
    g ← 0
    A(0) ← Initialize(M, F, T)
    while ( termination criterion not satisfied ) do
        F_i(g) ← Evaluate(A_i(g))    ∀i ∈ {1, ..., M}
        B(g) ← Select(A(g), τ)
        A(g + 1) ← CrossoverMutation(B(g), M) ∪ {A_best(g)}
        A_i(g) ← LocalSearch(A_i(g), LS_max)    ∀i ∈ {1, ..., M}
        g ← g + 1
    endwhile
endprocedure
```

Fig. 2. The procedure for evolving neural trees.

$$(S_1 w_{10} w_{11}(S_2 w_{20} w_{21}(x_2) w_{22}(x_3))$$

$$w_{12}(S_3 w_{30} w_{31}(x_4)$$

$$w_{32}(P_4 w_{40} w_{41}(x_1) w_{42}(x_3) w_{43}(x_5)))$$

$$w_{13}(x_1)$$

$$w_{14}(P_5 w_{50} w_{51}(x_2) w_{52}(x_4) w_{53}(x_5))).$$

In functionality, neural trees are more general than multilayer perceptrons, since any feedforward networks can be represented using a forest of neural trees. In addition, neural trees can simulate direct connections between non-neighboring layers by replicating input units on the terminal nodes. The output of one unit can be used as an input to more than one unit in the upper layers by using it more than once. Thus neural trees are flexible in representing general feedforward networks consisting of heterogeneous neurons with sparse, irregular, and partial connectivity.

Neural trees are used here to build models of complex systems, and later to predict their behavior, based on the models. The observations of the input–output behavior of the system is used as a training set D, and forms the basis for modeling the unknown process f:

$$D=\{(x_c,y_c)\}_{c=1}^{N} \tag{9}$$

where $x_c \in X$, $y_c \in Y$ and $y_c=f(x_c)$. The domain X and the range Y are defined by the application. The goodness of the model, i.e. neural tree A, is measured in terms of the error:

$$\sum_{c=1}^{N} (y_c - f_A(x_c))^2, \tag{10}$$

where f_A is the function realized by A.

In this setting, the modeling process is formulated as a search for a highly fit neural tree, A_{best}, in the whole neural tree space $\mathcal{A}=\{A_1,A_2,\ldots\}$. To solve this problem evolutionary algorithms are used, as described in the next section.

3. EVOLVING NEURAL TREES

For the construction of models for a particular system, a population $\mathcal{A}$ consisting of neural trees A_i:

$$\mathcal{A}(g)=(A_1,A_2,\ldots,A_M) \tag{11}$$

is maintained, where M is the size of the population. The initial population $\mathcal{A}(0)$ is created at random; the sizes, shapes and weights of the trees are randomly generated. In each generation g, the fitness values $F_i(g)$ of trees are evaluated as described in the next section, and the upper $\tau\%$ are selected to be in the mating pool $\mathcal{B}(g)$. The next generation $\mathcal{A}(g+1)$ of M individuals is then created by crossover, mutation, and weight adaptation. The best neural tree is always retained in the next generation, so that the population's performance does not decrease as generation goes on (elitist strategy). New populations are generated until an acceptable solution is found.

The crossover operator selects two parents, B_i and B_j, at random, and exchanges their subtrees to produce two offspring $B_{i'}$ and $B_{j'}$. In this way, the sizes, depths and receptive field shapes of neural trees are adapted. Mutation changes the neuron types of nonterminal nodes, and the labels of terminal units of trees.

Between generations, the weights of neural trees are adapted by stochastic hill-climbing. This local search is based on the breeder genetic algorithm (BGA), which has proved very robust for a wide range of parameter-optimization problems (Mühlenbein and Schlierkamp-Voosen, 1994). In BGA, the step size Δw_{ij} of weight w_{ij} is determined with a random value $\epsilon \in [0,1]$:

$$\Delta w_{ij}=R\cdot 2^{-\epsilon K}, \tag{12}$$

where R and K are constants specifying the range and slope of the exponential curve. The density function $\phi(Z)$ and the distribution function $\Phi(Z)$ for the random variable $Z=2^{-\epsilon K}$ are given as:

$$\phi(z)=\frac{1}{K\cdot\ln(2)\cdot z} \tag{13}$$

$$\Phi(z)=Pr(Z\leq z)=\int_{2^{-K}}^{z}\phi(u)\,du=1+\frac{1}{K\cdot\ln(2)}\cdot\ln(z). \tag{14}$$

The exponential characteristic of $\delta=2^{-\epsilon K}\in[2^{-K},1]$ ensures that the mutation step size of maximum vale R is possible, but small steps are more frequent.

Due to the large costs of local searches, various heuristics have been used. One heuristic is to apply a local search immediately after fitness evaluation to some portion, say the top 50%, of the population instead of all its members. Another heuristic is to adapt the intensity of the local search, LS_{max}, during the run. For instance, a small number of local search steps is used during the earlier phase, while a large number of local search steps are used after fit individuals have appeared. A combination of both strategies is also possible. For instance, a local search is applied with a low intensity at each generation to only 50% of the whole population until a specified fitness is reached, after which local search alone is intensively applied to the best 10% of the population.

For the following discussion, it is important to note that although the set of neuron types and external inputs is finite, any arbitrarily large neural trees can be evolved by repeated application of crossover. Though the error for the training data [equation (10)] can be used as a single measure for fitness, the principle of Occam's Razor says that the simpler model should be preferred to complex models if all other things are equal (Zhang and Mühlenbein, 1993). In addition, large structures require more computer resources in space and time for the evolution. Thus, a mechanism is required that keeps the resulting neural trees as sparse as possible.

The following section describes a regularization technique, i.e. an adaptive fitness evaluation method for dealing with the parsimony problem using the minimum description length (MDL) principle (Rissanen, 1986; Zhang and Mühlenbein, 1995). This method is designed to grow and prune the program size dynamically by balancing the ratio of training accuracy to solution complexity. Parsimonious solutions are obtained without losing the population diversity that is necessary for achieving the user-defined training accuracy.

4. MDL-BASED FITNESS EVALUATION

As outlined above, the goal of the algorithm is formulated as finding a neural tree, A, whose evaluation f_A best approximates the underlying relation f, where the approximation quality is measured by

$$E(D|A)=\frac{1}{N}\sum_{c=1}^{N}(\mathbf{y}_c-f_A(\mathbf{x}_c))^2. \tag{15}$$

Considering the tree as a Gaussian model of the data, the likelihood of the training data is described by

$$P(D|A)=\frac{1}{Z(\beta)}\exp(-\beta E(D|A)), \tag{16}$$

where $Z(\beta)$ is a normalizing constant, and β is a positive constant determining the sensitivity of the probability to the error value.

Bayes' rule states that the posterior probability of a model is:

$$P(A|D)=\frac{P(D|A)P(A)}{P(D)} \tag{17}$$

where $P(A)$ is the prior probability of the models and

$$P(D)=\int P(D|A)P(A)\,dA. \tag{18}$$

The most plausible model, given the data, is then inferred by comparing the posterior probabilities of all models.

According to coding theory (Rissanen, 1986), if $P(\mathbf{x})$ is given, then its code length is given as $L(P(\mathbf{x}))=-\log(P(\mathbf{x}))$. Maximizing $P(D|A)P(A)$ is thus equivalent to minimizing the total code length:

$$\begin{aligned}L(A|D)&=L(P(D|A)P(A))=-\log(P(D|A)P(A)) \\ &=L(D|A)+L(A),\end{aligned} \tag{19}$$

where $L(D|A)=-\log P(D|A)$ and $L(A)=-\log P(A)$. Here $L(D|A)$ is the code length of the data when encoded using the model A as a predictor for the data D, and $L(A)$ is the length of the model itself. This leads to the minimum description length (MDL) principle (Rissanen, 1986).

An implementation of MDL typically necessitates knowing the true underlying probability distribution, or an approximation of it. In general, however, the distribution of the underlying data structure is unknown, and the exact formula for the fitness function is impossible to obtain. Thus, it is proposed here to measure the fitness of a tree A, given a training set D, as

$$F(A|D)=F_D+F_A=\beta E(D|A)+\alpha C(A), \tag{20}$$

where the parameters α and β control the trade-off between complexity $C(A)$ and fitting error $E(D|A)$ of the program. The problem is how to balance α and β in unknown environments.

The basic approach used here is to fix the error factor at each generation, and to change the complexity factor adaptively with respect to the error. Let $E_i(g)$ denote the error defined by some criterion, i.e.

$$E_i(g)=E(D|A_i^g). \tag{21}$$

The training set D is assumed to be fixed with size N during evolution. For later use, the error of the best individuals is up to the gth generation:

$$E_{best}(g)=E_{i^*}(g) \tag{22}$$

$$i^* = arg \min_i \{F_i(g): i = 1, \ldots, M\}. \quad (23)$$

Here $F_i(g)$ is the total fitness value of the ith individual at generation g, used for reproduction of the next population.

Let $C_i(g)$ be the complexity value of the ith individual in the gth population. The complexity of the neural tree is a linear sum:

$$C_i(g) = C(A_i^g) = W(A_i^g) + U(A_i^g) + L(A_i^g), \quad (24)$$

where $W(A_i^g), U(A_i^g)$ and $L(A_i^g)$ denote the numbers of weights, units, and layers respectively. This enforces a selection pressure, so that shallow neural trees with a small number of units and weights are preferred to deep structures with a large number of units and weights.

At the end of each generation g the complexity of the best individual is also retained:

$$C_{best}(g) = C_{i^*}(g) \quad (25)$$

$$i^* = arg \min_i \{F_i(g): i = 1, \ldots, M\}. \quad (26)$$

Based on $C_{best}(g)$, the size of the best individual in the next generation is estimated as

$$\hat{C}_{best}(g+1) = C_{best}(g) + \Delta C_{sum}(g), \quad (27)$$

where $\Delta C_{sum}(g)$ is a moving average that keeps track of the difference in the complexity between the best individual of one generation and the best individual of the next:

$$\Delta C_{sum}(g) = \frac{1}{2}\{C_{best}(g) - C_{best}(g-1) + \Delta C_{sum}(g-1)\} \quad (28)$$

with $\Delta C_{sum}(0) = 0$. At the beginning of generation g, the Occam factor, $\alpha(g)$, is computed as a function of the best error $E_{best}(g-1)$ of the last generation and the estimated best size $\hat{C}_{best}(g)$ of the current generation:

$$\alpha(g) = \begin{cases} \dfrac{1}{N^2}\dfrac{E_{best}(g-1)}{\hat{C}_{best}(g)} & \text{if } E_{best}(g-1) > \epsilon \\ \dfrac{1}{N^2}\dfrac{1}{E_{best}(g-1)\cdot\hat{C}_{best}(g)} & \text{otherwise,} \end{cases} \quad (29)$$

where N is the size of training set. The Occam factor is then used in the fitness function as

$$F_i(g) = E_i(g) + \alpha(g)C_i(g). \quad (30)$$

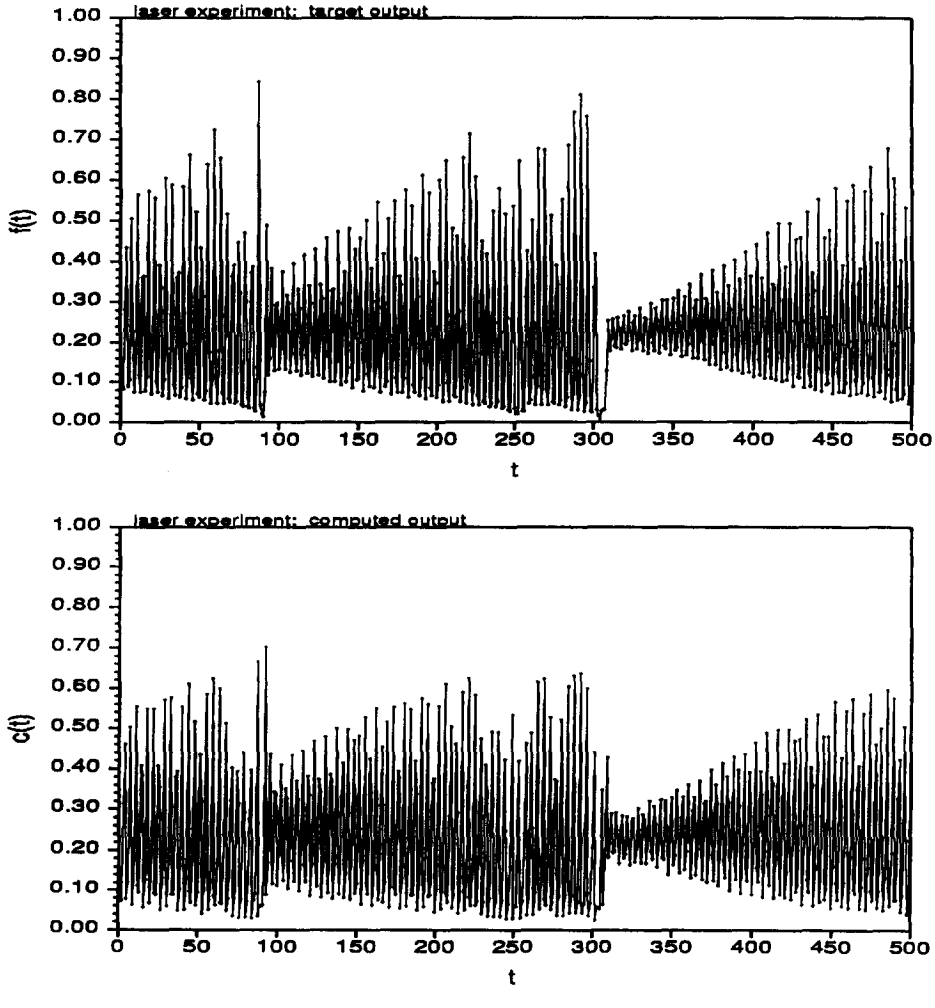

Fig. 3. Training performance for the Lorenz-like time series. (*top*) The time series used for training and (*bottom*) one-step ahead prediction for the training data.

This equation is a realization of the general form derived from the MDL approach [equation (20)] where β is fixed and α is expressed as a function of g:

$$\beta = 1.0 \text{ and } \alpha = \alpha(g). \tag{31}$$

Note that $\alpha(g)$ depends on $E_{best}(g-1)$ and $\hat{C}_{best}(g)$.

The user-defined parameter ϵ in equation (29) specifies the maximum training error allowed for the final solution. When $E_{best}(g-1) > \epsilon$, $\alpha(g)$ decreases as the training error falls, since $E_{best}(g-1) < 1$ is multiplied. This encourages fast error reduction in the early stages of evolution. In contrast, for $E_{best}(g-1) \leq \epsilon$, as $E_{best}(g)$ approaches 0 the relative importance of complexity increases due to the division by a small value $E_{best}(g-1) < 1$. This encourages stronger complexity reduction in the final stages, to obtain parsimonious solutions.

On the other hand, the Occam factor $\alpha(g)$ decreases as the complexity $\hat{C}_{best}(g)$ increases for a fixed $E_{best}(g-1)$, so that once the size of the best program grows, the individuals have an increasingly higher chance of growth. This is intended to give a counter effect to Occam's Razor, to ensure growth if necessary. This method is applicable independently of the complexity definition, since the Occam factor is controlled by the ratio of the current complexity to the estimated best complexity, and not by the absolute value.

5. LORENZ-LIKE CHAOTIC SYSTEMS

The system to be modeled in this section is a far-infrared NH_3 laser which generates chaotic intensity fluctuations (Hübner *et al.*, 1993). The data used for modeling is a series of 1000 measurements in the physics laboratory. On the one hand, the time series can be described by a relatively simple "correct" model of three nonlinear differential equations, the same equations that Lorenz used to approximate weather phenomena. On the other hand, since the first 500 datapoints training set showed only two of five collapses, it is difficult to predict the next collapse on the basis of so few instances. This data set was used as a benchmark problem in the 1992 Santa Fe time-series competition (Weigend and Gershenfeld, 1993).

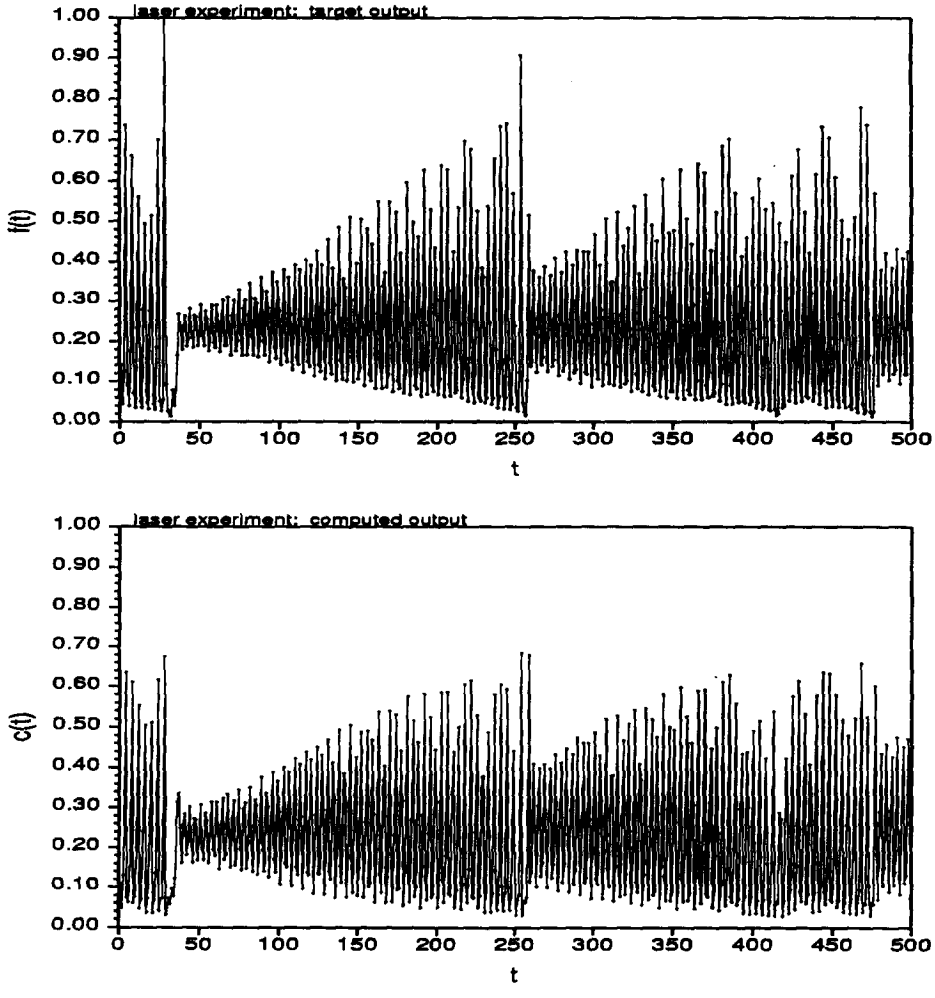

Fig. 4. Generalization performance for the Lorenz-like time series. (*top*) The time series used for test and (*bottom*) one-step ahead prediction for the test data.

The training examples were formed from the time series $\{x(t)\}$ by specifying groups with respect to a time index t. The input pattern was assigned $(x(t-1), x(t-2), x(t-3))$; the desired output was $x(t)$:

$$x(t)=f(x(t-1), x(t-2), x(t-3)). \quad (32)$$

Figure 3 shows the time series of 500 steps used for training (top) and the predicted result by the evolved neural tree (bottom). The next 500 steps for prediction are shown in Fig. 4, in which the top curve is the target and the bottom one is the one-step-ahead prediction result. It is remarkable that though some peaks of the time series were not completely fit during the learning phase, the generalization performance on the unseen time series is relatively good.

Figure 5 (top) plots the evolution of the fitness and error of the best and worst neural trees in each generation. The evolution of structural complexity in terms of the number of layers is shown in Fig. 5 (bottom). Comparing both figures, the tendency is observed that the tree size first grows and then shrinks, during which the error of the best individual steadily decreases. This is the desired effect of the Occam's Razor, implemented in the fitness function.

Figure 6 compares the generalization accuracy of two solutions obtained on different runs with the laser data. Shown in the figure is the prediction error d(t) at time t for

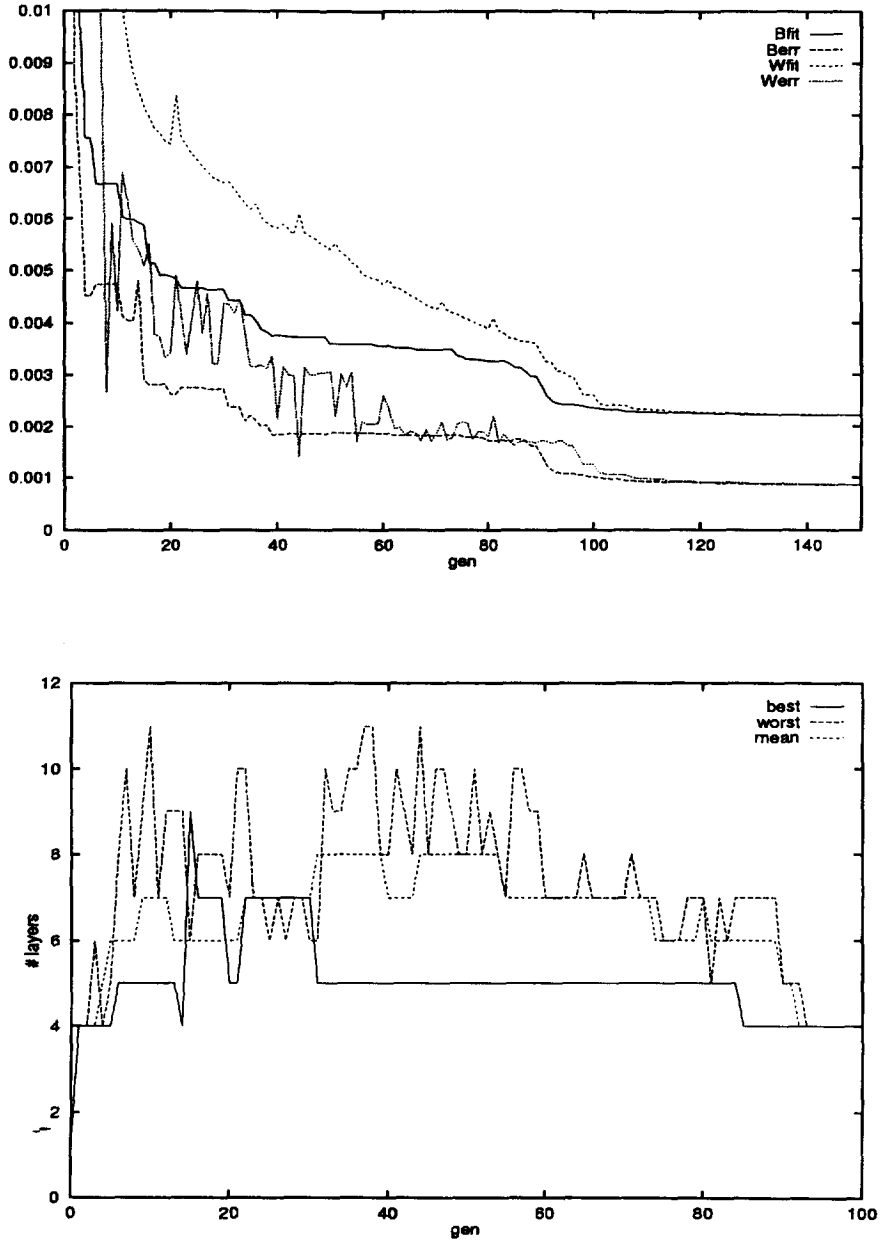

Fig. 5. Evolution of fitness and complexity for the laser data. (*top*) The upper two curves plot the fitness of the worst and best neural trees, and the lower two curves show the error of the worst and best, respectively. (*bottom*) The top and bottom curves show the size of worst and best neural trees, while the middle curve is the mean size of the population.

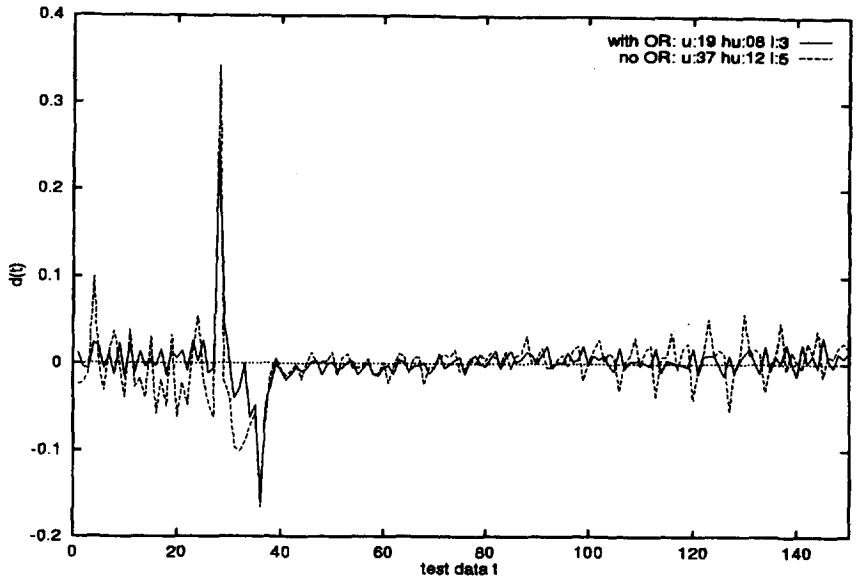

Fig. 6. Comparison of prediction errors of two solutions with differing complexity for the laser data. The smaller tree shows better predictive performance.

each solution. One run was made using Occam's Razor in its fitness function (MDL-based fitness measure), and resulted in a neural tree structure of 3 layers, containing 8 hidden units and 19 units in total. The other run was made without using the complexity penalty (but using a maximum complexity limit), resulting in a larger structure of 5 layers, containing 12 hidden units and a total of 37 units. This indicates that the use of the complexity penalty evolved a more parsimonious structure. The generalization performance of the smaller tree was better than that of the larger one.

6. MACKEY–GLASS DYNAMIC SYSTEMS

The Mackey–Glass differential equation is a dynamic system, created as a model for blood flow:

$$\frac{dx(t)}{dt} = \frac{ax(t-\tau)}{1+x^{10}(t-\tau)} - bx(t). \tag{33}$$

For $a=0.2$, $b=0.1$, and $\tau=17$, the trajectory is chaotic, and lies on an attractor of fractal dimension of approximately 2.1.

Two different instances of the MacKey–Glass time series were used here. One is to model the behavior of the system based on 9 contiguous data points in the past:

$$x(t)=f(x(t-1),x(t-2),\ldots,x(t-9)). \tag{34}$$

For this problem, 100 data points were used to evolve neural tree models, and the continuing 400 points were used to evaluate the prediction performance of the evolved models. Figure 7 (left) shows the learning accuracy of the training

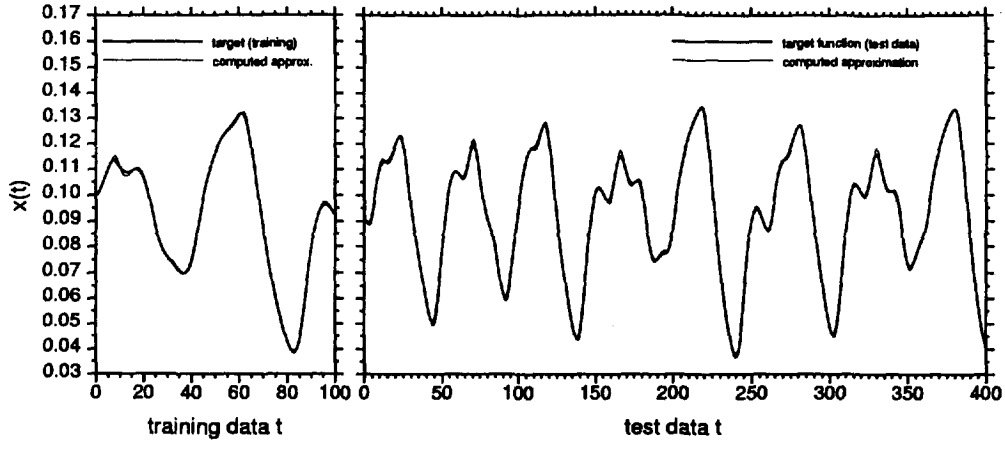

Fig. 7. Results for the Mackey–Glass dynamic system: (*left*) training performance and (*right*) one-step ahead prediction performance. The thick curves denote the true values and the thin curves the predicted ones.

data for the one-step-ahead prediction. Figure 7 (right) plots the one-step-ahead prediction for the test data. This run was made with a population size of $M=200$ for 80 generations. The neural tree structure evolved for the above performance contained 59 weights, with 13 hidden units. The mean-squared error of this solution is 0.6×10^{-6}. This error is eight times lower than the value 4.7×10^{-6} obtained by STROGANOFF (Iba *et al.*, 1993), a GMDH-based system-identification method.

It is interesting to observe in the resulting neural tree solutions that the inputs of the near past, such as $x(t-1)$, $x(t-2)$, $x(t-3)$, etc., appear more often than the inputs with greater distance in the series, such as $x(t-9)$, $x(t-8)$, $x(t-7)$, etc. In addition, $x(t-1)$ generally appeared more

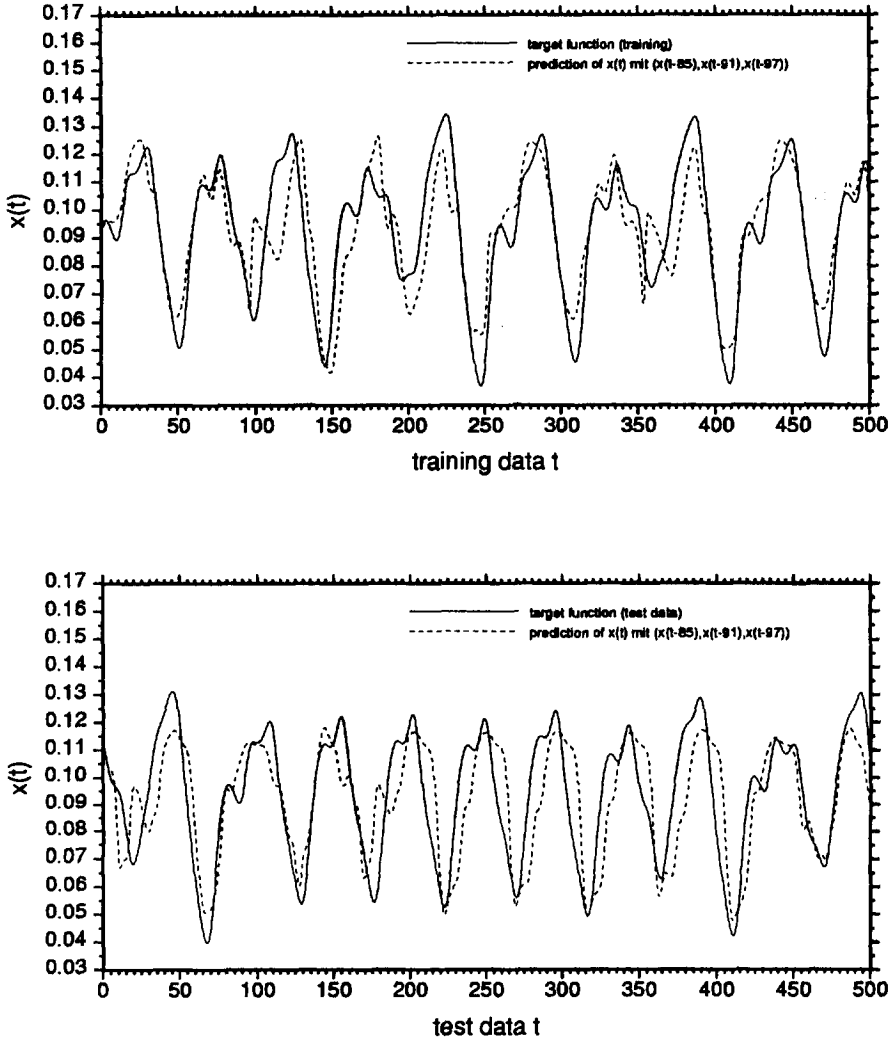

Fig. 8. Performance for the Mackey–Glass time series with $T=85$.

Table 1. Comparison of results for the Mackey–Glass problem with $T=85$. BP stands for backpropagation networks and NT stands for neural trees. The number in parenthesis for neural trees is the total number of units. The learning time is given in epochs for BP and in generations for NT

Method	Hidden units	Parameters	E_{rel} training	E_{rel} test	Time
BP	10–10	171	0.54	0.55	400 epo.
BP	20–20	541	0.54	0.55	400 epo.
BP	100	601	0.53	0.56	400 epo.
BP	300	1801	0.69	0.84	400 epo.
NT best	30 (124)	153	0.52	0.58	80 gen.
NT mean	26 (114)	139	0.55	0.61	80 gen.

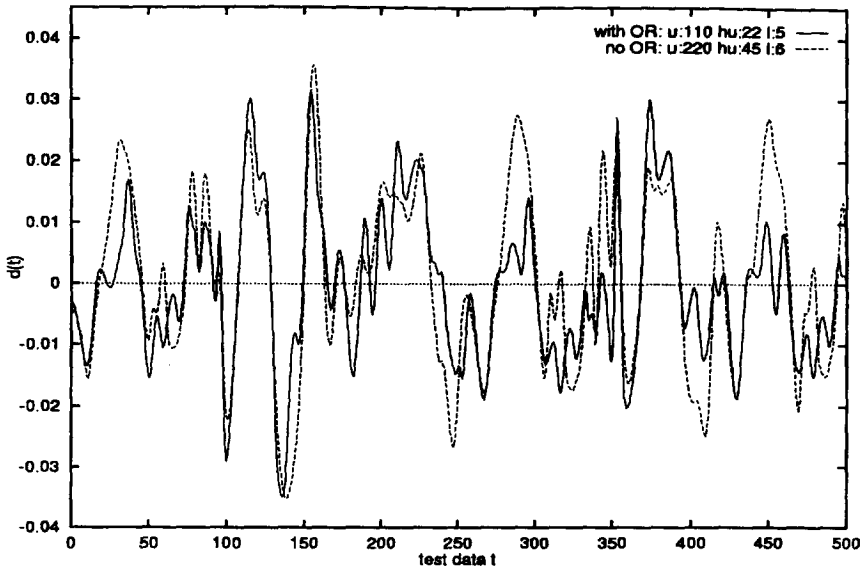

Fig. 9. Comparison of prediction errors of two solutions with differing complexity for the Mackey–Glass series with $T=85$. The smaller tree shows better predictive performance.

often than the other inputs in solutions evolved in most experiments. The evolutionary method seems to assign different credit to different inputs to find problem-specific solutions.

The second problem, and a more difficult one, is to learn to predict the following instance of the time series equation (33):

$$x(t+T)=f(x(t), x(t-d), x(t-2d), x(t-3d)) \text{ with } d=6 \text{ and } T=85. \quad (35)$$

The first 500 data points were used for evolving the trees, and the continuing 500 points were used for testing the prediction accuracy. Figure 8 shows the results for training (top) and test (bottom) for this problem. The mean-squared error of the prediction accuracy is 0.00016 after 80 generations, with population size $M=200$. This accuracy outperforms the mean-squared error of 0.0187 obtained by STROGANOFF.

For a comparison with other methods, the results obtained by backpropagation networks reported in (Hartman and Keeler, 1991) are shown in Table 1. The figures in the table indicate that the evolutionary neural trees converge robustly to parsimonious solutions, while the performance of backpropagation networks may be highly dependent on the network structure specified by the designer. The best performance of neural trees is comparable to that of the backpropagation network. The goodness measure used here is the relative error E_{rel}: the root of mean-squared error divided by the standard deviation. It should be noted that the data used by the different methods are not exactly the same, since the MacKey–Glass equation depends on the initial values, which are not reported in the literature.

Figure 9 compares the prediction errors of two neural tree solutions with differing complexity, showing a positive effect of Occam's Razor on predictive performance. The solid line shows the result of a tree consisting of 5 layers of 110 units (22 units of which are hidden units), which was obtained by using the Occam's Razor in the fitness function. The dotted line shows a bigger neural tree consisting of 6 layers of 220 units (45 hidden units), obtained by not using the complexity penalty.

7. CONCLUDING REMARKS

An evolutionary method has been presented, that builds models of complex systems using neural trees consisting of various neuron types. In contrast to previous approaches, this method is based on tree-structured representations of neural networks, which allows the dynamic adaptation of the neuron type, weight, and topology in a natural way using genetic operators.

The method was successfully applied to two representative time-series prediction problems. One primary difficulty with this approach is that the tree size may become very large, without any improvement in generalization performance. The use of a complexity penalty in fitness evaluation proved highly useful for solving this problem, as was indicated by the relatively good generalization performance.

The generality of the proposed approach is contrasted with various existing methods for system identification, which explore only restricted subsets rather than the complete class of functional structures. Considering the enormous size of the search space, it is interesting to understand why the evolutionary search finds a reasonable solution. It seems that there are some regularities that the evolutionary search can exploit, to explore the neural tree space efficiently, and to discover relevant structures and eliminate irrelevant substructures. This method seems especially effective in identifying the important variables and structures of the systems whose functional structures

are unknown or ill-defined. Other examples of promising application areas of the evolutionary neural trees include monitoring, fault-detection and diagnosis of complex systems, such as environmental processes (Zhang *et al.*, 1995a).

REFERENCES

Fogel, D. B. (1995) *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. IEEE Press, New York.

Goldberg, D. (1994) Genetic and evolutionary algorithms come of age. *Communications of the ACM*, **39**(3), 113–119.

Hartman, E. and Keeler, J. D. (1991) Predicting the future: advantages of semilocal units. *Neural Computation*, **3**, 566–578.

Hübner, U., Weiss, C. O., Abraham, N. B. and Tang, D. (1993) Lorenz-like chaos in NH_3–FIR laser. In *Time Series Prediction: Forecasting the Future and Understanding the Past*, eds A. S. Weigend and N. A. Gershenfeld, pp. 73–104. Addison–Wesley, Reading, MA.

Iba, H., Kurita, T., de Garis, H. and Sato, T. (1993) System identification using structured genetic algorithms. In *Proc. Fifth Int. Conf. Genetic Algorithms*, pp. 279–286. Morgan Kaufmann, San Mateo, CA.

Ivakhnenko, A. G. (1971) Polynomial theory of complex systems. *IEEE Transactions on Systems, Man, and Cybernetics*, **1**(4), 364–378.

Koza, J. R. (1992) *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA.

Mühlenbein, H. and Schlierkamp-Voosen, D. (1994) The science of breeding and its application to the breeder genetic algorithm. *Evolutionary Computation*, **1**(4), 335–360.

Rissanen, J. (1986) Stochastic complexity and modeling. *The Annals of Statistics*, **14**, 1080–1100.

Tenorio, M. F. and Lee, W.-T. (1990) Self-organizing network for optimum supervised learning. *IEEE Transactions on Neural Networks*, **1**(1), 100–110.

Weigend, A. S. and Gershenfeld, N. A. (eds) (1993) *Time Series Prediction: Forecasting the Future and Understanding the Past*. Addison–Wesley, Reading, MA.

Zhang, B. T. and Mühlenbein, H. (1993) Evolving optimal neural networks using genetic algorithms with Occam's razor. *Complex Systems*, **7**(3), 199–220.

Zhang, B. T. and Mühlenbein, H. (1995) MDL based fitness functions for learning parsimonious programs. In *Proc. AAAI 1995 Fall Symposium on Genetic Programming*, pp. 122–126. AAAI Press, New York.

Zhang, B. T., Ohm, P. and Mühlenbein, H. (1995a) Water pollution prediction with evolutionary neural trees. In *Proc. IJCAI-95 Workshop on AI and the Environment*, August. Montreal, Canada.

Zhang, B. T., Ohm, P. and Mühlenbein, H. (1995b) Learning to predict with evolutionary neural trees. In *Proc. World Congress on Neural Networks*, pp. 823–826. Washington, DC.

AUTHORS' BIOGRAPHIES

Byoung-Tak Zhang is an Assistant Professor in the Department of Computer Science and Engineering at Konkuk University, Seoul, Korea. He received his Ph.D. degree in Computer Science from the University of Bonn, Germany in 1992, and the M.S. and B.S. degrees in Computer Engineering from Seoul National University, in 1988 and 1986, respectively. From 1988 to 1992 he has been a recipient of the POSCO Scholarship. From 1992 to 1995 Dr Zhang has been a research fellow at the German National Research Center for Information Technology (GMD), Sankt Augustin, Germany. His research interests include evolutionary algorithms, neural networks, machine learning, artificial intelligence, and their application to real-world problems.

Peter Ohm is a Researcher at the German National Research Center for Information Technology (GMD), Sankt Augustin, Germany. He received his Masters degree in Computer Science from the University of Bonn, Germany in 1996. His research areas include genetic programming, evolutionary algorithms, combinatorial optimization, and their applications.

Heinz Mühlenbein is Research Manager at the German National Research Center for Information Technology (GMD) in Sankt Augustin, Germany. Since 1987, he has been head of the Adaptive Systems Research Group. The mission of the research group is to develop systems that can deal with incomplete information in the real world. Previously Dr Mühlenbein has done research in the areas of performance analysis of computer systems, computer networks and parallel processing. Dr Mühlenbein is on the editorial board of several journals, including *Evolutionary Computation, Heuristics and Scientific Computation*. His current research is focused on evolutionary algorithms, reflective learning, and learning robots.