Complex Systems **7** (1993) 199–220

Evolving Optimal Neural Networks Using Genetic Algorithms with Occam's Razor

Byoung-Tak Zhang*
Heinz Mühlenbein†
Artificial Intelligence Research Division
German National Research Center for Computer Science (GMD)
Schloss Birlinghoven, D-53757 Sankt Augustin, Germany

Abstract. Genetic algorithms have had two primary applications for neural networks: optimization of network architecture, and training weights of a fixed architecture. While most previous work focuses on one or the other of these options, this paper investigates an alternative evolutionary approach—breeder genetic programming (BGP)—in which the architecture and the weights are optimized simultaneously. In this method, the genotype of each network is represented as a tree whose depth and width are dynamically adapted to the particular application by specifically defined genetic operators. The weights are trained by a next-ascent hillclimbing search. A new fitness function is proposed that quantifies the principle of Occam's razor; it makes an optimal trade-off between the error fitting ability and the parsimony of the network. Simulation results on two benchmark problems of differing complexity suggest that the method finds minimal networks on clean data. The experiments on noisy data show that using Occam's razor not only improves the generalization performance, it also accelerates convergence.

1. Introduction

Constructing multilayer neural networks involves difficult optimization problems, such as finding a network architecture appropriate for the application at hand, and finding an optimal set of weight values for the network to solve the problem. Genetic algorithms (GAs) [8, 5, 20] have been used to solve each of these optimization problems [36]. In weight optimization, the set of weights is represented as a chromosome, and a genetic search is applied to

*Email: zhang@gmd.de
†Email: muehlen@gmd.de

© 1993 Complex Systems Publications, Inc.

the encoded representation to find a set of weights that best fits the training data. Some encouraging results have been reported that are comparable with conventional learning algorithms [17]. In architecture optimization, the topology of the network is encoded as a chromosome, and genetic operators are applied to find an architecture that best fits the specified task (according to explicit design criteria).

The optimization of neural network architectures and the discovery of minimal networks for particular applications are important processes, because the speed and accuracy of learning and performance are dependent on network complexity (i.e., the type and number of units and connections, and the connectivity of units). For example, a network having a large number of adjustable connections tends to converge quickly, but it usually leads to overfitting of the training data. On the other hand, a small network will achieve a good generalization if it converges, but it needs, in general, a large amount of training time [1, 32]. Therefore, the size of a network should be as small as possible, yet sufficiently large to ensure an accurate fitting of the training set.

A general method for evolving genetic neural networks was suggested by Mühlenbein and Kindermann [24]. Recent works, however, have used GAs separately in each optimization problem, primarily focusing on optimizing network topology. Harp et al. [7] and Miller et al. [15] have described representation schemes in which the anatomical properties of the network structure are encoded as bit-strings. A similar representation has been used by Whitley et al. [36] to prune unnecessary connections. Kitano [11] and Gruau [6] have suggested encoding schemes in which a network configuration is indirectly specified by a graph-generation grammar that is evolved by GAs. All of these methods use the backpropagation algorithm [29], a gradient-descent method, to train the weights of the network. Koza [12] provides an alternative approach to the representation of neural networks, under the framework of genetic programming (GP), which enables modification of not only the weights of a neural network, but the architecture as well. However, this method provides neither a general method for representing an arbitrary feedforward network, nor a mechanism for finding a network of minimal complexity.

In this paper we describe a new genetic programming method, which we call breeder genetic programming, that employs Occam's razor in the fitness function. This method makes an optimal trade-off between the error fitting ability and the parsimony of the network; it prefers a simple network architecture to a complex one, given a choice of networks having the same fitting errors. The weights are not trained by backpropagation, but by a next-ascent hillclimbing search.

The organization of the paper is as follows. Section 2 presents a grammar for representing multilayer feedforward neural networks. Section 3 describes the genetic operators and the control algorithm for adapting the architectures and the weights. Section 4 derives the fitness function for the genetic search of minimal-complexity solutions. The experimental results are given

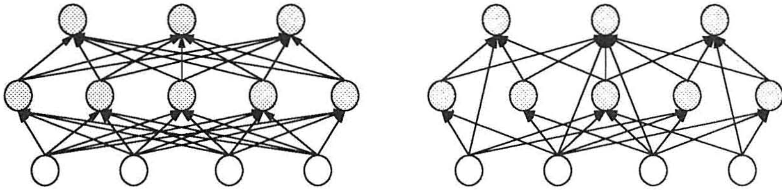

Figure 1: Architectures of multilayer perceptrons. While a commonly used architecture adopts full connectivity between neighboring layers only (left), the architecture used in this paper allows local receptive fields and direct connections between non-neighboring layers (right).

in section 5, followed by an analysis of fitness landscapes in section 6, and discussion in section 7.

2. Representing neural networks as trees

Multilayer feedforward neural networks (or multilayer perceptrons) [28, 16, 29] are networks of simple processing elements, called neurons or units, organized in layers. The external inputs are presented to the input layer and are fed forward via one or more layers of hidden units to the output layer. There are no connections between units in the same layer. A commonly adopted architecture allows full connectivity between neighboring layers only. We allow both partial and direct connections between nonneighboring layers, because this is important for finding a parsimonious architecture. Specifically, this allows for some input units to be connected directly to output units. Figure 1 compares a typical multilayer perceptron to a more general architecture as adopted in this work. There are many types of neural units; we confine ourselves to McCulloch-Pitts neurons [14], although the method we describe can be extended easily to other types of neurons.

The McCulloch-Pitts neuron is a binary device. Each neuron has a threshold. The neuron can receive inputs from excitatory and/or inhibitory synapses. Given an input vector x, the net input of the ith unit, I_i, is computed by

$$I_i = \sum_{j \in R(i)} w_{ij} x_j \tag{1}$$

where w_{ij} is the connection weight from unit j to unit i, and $R(i)$ denotes the receptive field of unit i.

The neuron becomes active if the sum of weighted inputs exceeds its threshold. If it does not, the neuron remains inactive. Formally, the units are activated by the threshold activation function

$$f_i(I_i) = \begin{cases} 1 & \text{if } I_i \geq \theta_i \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

where θ_i denotes the threshold value for unit i. The θ_i is usually considered as a weight w_{i0} in (1), connected to an extra unit whose activation value is

$$
\begin{array}{lcl}
\text{NN} & \longrightarrow & (\ Y_1\ Y_2\ ...\ Y_m\ ) \\
Y & \longrightarrow & (\ \text{`Y'}\ r\ \theta\ W_1\ W_2\ ...\ W_r\ ) \\
W & \longrightarrow & (\ \text{`W'}\ w\ \{Y \mid X\}\ ) \\
X & \longrightarrow & \text{`X'}\ i \\
\theta & \longrightarrow & \Theta_{bin} \mid \Theta_{int} \\
\Theta_{bin} & \longrightarrow & -1 \mid +1 \\
\Theta_{int} & \longrightarrow & -r \mid ... \mid 0 \mid ... \mid +r \\
w & \longrightarrow & \Omega_{bin} \mid \Omega_{int} \\
\Omega_{bin} & \longrightarrow & -1 \mid +1 \\
\Omega_{int} & \longrightarrow & 0 \mid \pm 1 \mid \pm 2 \mid \pm 3 \mid ... \\
r & \longrightarrow & 1 \mid 2 \mid 3 \mid ... \\
i & \longrightarrow & 1 \mid 2 \mid 3 \mid ... \mid n
\end{array}
$$

Figure 2: Grammar for generating the genotype of a feedforward network of McCulloch-Pitts neurons. A network is represented as a set of m trees, each having an arbitrary number of subtrees. Each leaf of the trees indexes one of the n external input units.

always 1. Despite their simplicity, McCulloch-Pitts neurons are very powerful. In fact, it can be shown that any finite logical expression can be realized by them [14].

In the case of a two layer (one hidden layer) architecture, the ith output of the network, y_i, is expressed as a function of inputs x and weights w:

$$y_i = f_i\left(\sum_{j \in R(i)} w_{ij} f_j \left(\sum_{k \in R(j)} w_{jk} x_k\right)\right) \tag{3}$$

where i, j, and k denote output, hidden, and input units, respectively. Note that $R(i)$ can include input as well as hidden units, because direct connections between input and output units are possible (when they are present, the f_j is an identity function).

For genetic optimization, we represent a feedforward network as a set of m trees, each corresponding to one output unit. Figure 2 shows the grammar for generating a feedforward network of n input and m output units. The nonterminal symbol Y is used to represent neural units (some of which are output units) having a threshold of θ and r weights. The integer r indicates the receptive-field width of the unit. Each connection weight is represented as a nonterminal node W consisting of a symbol 'W' and a weight value w, followed by a nonterminal symbol indicating recursively another neural unit Y or an external input unit X. An external input is described by a symbol 'X' followed by an integer i denoting the index of the input unit.

In our simulations, we used binary thresholds. McCulloch-Pitts neurons allow integer thresholds; however, networks with binary thresholds can realize networks with integer thresholds by using additional neurons. Similarly, integer weights can be realized by neurons using binary weights. The number

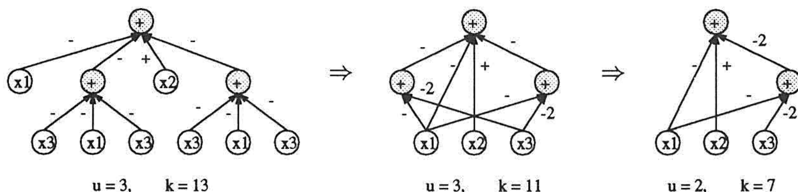

Figure 3: Conversion of a tree into networks. The tree representation allows fine-tuning of the network structure. Integer weights of a network are represented in a tree by means of multiple binary weights.

of weights and units is typically reduced if the genotype is transformed into a network of integer values. This is illustrated in figure 3, in which u and k denote the number of units and adjustable weights, respectively.

Binary weights are useful because they can be trained by a simple hill-climbing search, instead of an expensive gradient-based method. A possible disadvantage of binary weight representation is that it requires a larger chromosome. However, the use of integer rather than binary weight representation does not reduce the search space and, therefore, does not automatically accelerate convergence. Another advantage of the binary weight representation is that it functions as a regularizing factor, by avoiding arbitrary growth of chromosome size.

3. Genetic breeding of neural networks

3.1 Breeder genetic programming (BGP)

For the evolution of optimal neural networks, we use the concepts based on the breeder genetic algorithm (BGA) of Mühlenbein et al. [25]. While genetic algorithms typically model a natural evolution, the BGA models a rational selection performed by human breeders. The BGA can be considered as a recombination of evolution strategies [27, 30] and GAs [8, 5]. The BGA uses truncation selection as performed by breeders. This selection scheme is similar to the (μ, λ)–strategy in [30]. The search process of the BGA is mainly driven by recombination. Our approach in this paper differs from the BGA in that we use variable-size chromosomes, a characteristic of GP [12]. Thus, we call our method breeder genetic programming (BGP). Whereas GP uses fitness-proportionate selection combined with crossover as the main operator, BGP uses truncation selection combined with crossover plus local hillclimbing. As we show, ranking-based selection makes it easier to balance the accuracy and parsimony of solutions.

The BGP evolutionary learning algorithm is summarized in figure 4. The algorithm maintains a population $\mathcal{A}$ consisting of M individuals A_i of variable size. Each individual represents a neural network. The networks of the initial population, $\mathcal{A}(0)$, are generated with a random number of layers. The receptive field and width of each neural unit are also chosen randomly.

1. Generate initial population $\mathcal{A}(0)$ of M networks at random. Set current generation $g \leftarrow 0$.
2. Evaluate fitness values $F_i(g)$ of networks using the training set of N examples.
3. If the termination condition is satisfied, then stop the evolution. Otherwise, continue with step 4.
4. Select upper τM networks of gth population into the mating pool $\mathcal{B}(g)$.
5. Each network in $\mathcal{B}(g)$ undergoes a local hillclimbing, resulting in revised mating pool $\mathcal{B}(g)$.
6. Create $(g+1)$th population $\mathcal{A}(t+1)$ of size M, by applying genetic operators to randomly chosen parent networks in $\mathcal{B}(g)$.
7. Replace the worst fit network in $\mathcal{A}(t+1)$ by the best in $\mathcal{A}(t)$.
8. Set $g \leftarrow g+1$ and return to step 2.

Figure 4: Summary of the BGP algorithm.

The $(g+1)$th population, $\mathcal{A}(g+1)$, is created from $\mathcal{A}(g)$ in three steps: selection, hillclimbing, and mating.

In the selection step, the most fit individuals τM in $\mathcal{A}(g)$ are accepted into the mating pool $\mathcal{B}(g)$. The parameter τ determines the selection intensity, and has a value from the interval $(0, 1]$. A fitness function that balances the error fitting ability and the parsimony of the networks is derived in section 4. After selection, each individual in $\mathcal{B}(g)$ undergoes a hillclimbing search in which the weights of the network are adapted by mutation. This results in the revised mating pool $\mathcal{B}(g)$. The mating phase repeatedly selects two random parent individuals in $\mathcal{B}(g)$ to mate and generate two offspring in the new population $\mathcal{A}(g+1)$ by applying crossover operators, until the population size amounts to M. Note that the size of individuals in one population may differ, $|A_i(g)| \neq |A_j(g)|, i \neq j$ and $i, j \in \{1, ..., M\}$, and the size of the same individual in a subsequent population also may differ, $|A_i(g+1)| \neq |A_i(g)|$, $i \in \{1, ..., M\}$.

A new population is generated repeatedly until an acceptable solution is found, or the variance of the fitness $V(g)$ falls below a specified limit value V_{min}; that is, the procedure terminates if

$$V(g) = \frac{1}{M} \sum_{i=1}^{M} \left(F_i(g) - \bar{F}(g) \right)^2 \leq V_{min} \tag{4}$$

where $\bar{F}(g)$ is the average fitness of the individuals in $\mathcal{A}(g)$. In addition, the algorithm terminates if a specified number of generations, g_{max}, is carried out.

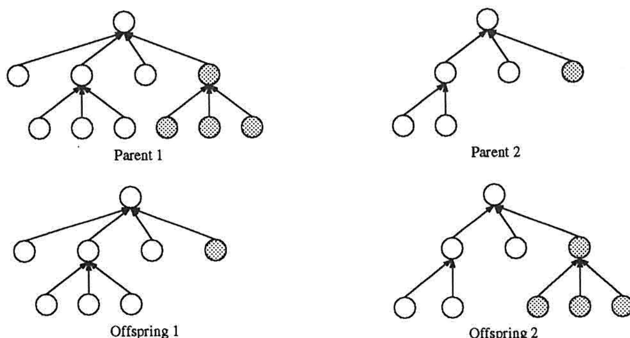

Figure 5: Crossover operation. Parent 1 and parent 2 mate by crossing over, and produce two new individuals (offspring 1 and offspring 2). In this example, the first network shrank, whereas the second grew. Guided by an appropriate selection mechanism, the network architecture is adapted in this way to the specific application.

3.2 Genetic operators

The weights of a network are trained by applying a hillclimbing search to each of the individuals accepted by truncation selection. Given a chromosome s_i, the next-ascent hillclimbing procedure finds a better chromosome s_i^{new} by repeatedly applying the mutation operator, until no weight configuration that has better fitness can be found in each sweep through the individual. The sequence of mutation is defined as the depth-first search order.

Each mutation operation is performed by replacing the value of a node u_i by another; that is, by finding the class U_k of u_i and replacing u_i by another member $u_j, j \neq i$ in the set U_k. First, the class U_k must be found, because some values (nodes) cannot be mutated to arbitrary values. For example, a weight value must be drawn from the set $\{+1, -1\}$. The biases are mutated in the same way as the weights. The index for the input units can be mutated by another input index.

Unlike mutation, the crossover operator adapts the size and shape of the network architecture. A crossover operation starts by randomly choosing two parent individuals from the mating pool $\mathcal{B}(g)$. The actual crossover of two individuals i and j is performed on their genotypical representations s_i and s_j. The nodes in the tree are numbered according to the depth-first search order, and crossover sites c_i and c_j are chosen at random, with the conditions

$$1 \leq c_i \leq \mathit{Size}(s_i) \qquad \text{and} \qquad 1 \leq c_j \leq \mathit{Size}(s_j),$$

where the length $\mathit{Size}(s_i)$ of an individual is defined as the total number of units and weights.

Given the crossover points, the subtrees of two parent individuals s_i and s_j are exchanged to form two offspring s_i' and s_j' (see figure 5). The label of the nodes c_i and c_j must belong to the same class, that is, they both must be

either Y-type or W-type nodes. The number of arguments of each operator plays no role, because the syntactically correct subtree under the node c_i and c_j is completely replaced by another syntactically correct subtree.

4. Fitness function with Occam's razor

Occam's razor states that simpler models should be preferred to unnecessarily complex ones [13, 33]. This section complies with Occam's razor by giving a quantitative method for using GAs to construct neural networks of minimal complexity.

In defining criteria for minimality, it is important that the network be able to approximate at least the training set to a specified performance level. The algorithm should prefer a smaller network to a larger network *only if* it achieves a comparable or better performance. Otherwise, the algorithm would not reduce the approximation error, preferring smaller networks that cannot be powerful enough to solve the task. Thus, the first term of the fitness function of an individual network should be the error function. The error function commonly used for the data set $D = \{(x_i, y_i) \mid i = 1, ..., N\}$ of N examples is the sum of the squared errors between the desired and actual outputs:

$$E(D \mid W, A) = \sum_{i=1}^{N} E(y_i \mid x_i, W, A) \tag{5}$$

with

$$E(y_i \mid x_i, W, A) = \sum_{j=1}^{m} (y_{ij} - o_j(x_i; W, A))^2, \tag{6}$$

where y_{ij} denotes the jth component of the ith desired output vector y_i, and $o_j(x_i; W, A)$ denotes the jth actual output of the network with architecture A and set of weights W for the ith training input vector x_i.

The complexity of a neural network architecture is dependent upon the task to be learned and can be defined in various ways, depending on the application. In general, the number of free parameters (or adjustable weights) of the network should be minimal, since this is one of the most important factors determining the speed and accuracy of learning. In addition, large weights generally should be penalized, in the hope of achieving a smoother or simpler mapping—this technique is called regularization [26, 13]. We define the complexity C of a network as

$$C(W \mid A) = \sum_{k=1}^{K} w_k^2 \tag{7}$$

where K is the number of free parameters. Note that K can be arbitrarily large, because we fit the architectures as well. In the case of binary weights, C reduces to the number of synaptic connections. This measure of complexity

might be extended by additional cost terms (such as the number of layers) when the application requires a fast execution of the trained network.

The combined fitness function that we try to *minimize* is defined as

$$F(D \mid W, A) = \alpha C(W \mid A) + \beta E(D \mid W, A) \tag{8}$$

where α and β are constants for the trade-off between error fitting and complexity reduction. This fitness function has an elegant probabilistic interpretation for the learning process: according to the Bayesian framework, minimizing F is identical to finding the most probable network with architecture A and weights W.

To show this, let us define the following. Let D be the training data set for the function $\gamma : X \rightarrow Y$, that is,

$$D = \{(x, y) \mid x \in X,\ y \in Y,\ y = \gamma(x)\}. \tag{9}$$

A model $\mathcal{M}$ of the function γ is an assignment to each possible pair (x, y) of a number $P(y \mid x)$ representing the hypothetical probability of y given x. That is, a network with specified architecture A and weights W is viewed as a model $\mathcal{M} = \{A, W\}$ that predicts the outputs y as a function of input x, in accordance with the probability distribution

$$P(y \mid x, W, A) = \frac{\exp(-\beta E(y \mid x, W, A))}{Z(\beta)}, \tag{10}$$

where β is a positive constant which determines the sensitivity of the probability to the error value, and

$$Z(\beta) = \int \exp(-\beta E(y \mid x, W, A))dy \tag{11}$$

is a normalizing constant (see [35]). Under the assumption of the Gaussian error model (i.e., if the true output is expected to include additive Gaussian noise with standard deviation σ), we have

$$P(y \mid x, W, A) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{E(y \mid x, W, A)}{2\sigma^2}\right) \tag{12}$$

with $\beta = 1/(2\sigma^2)$ and $Z(\beta) = \sqrt{2\pi}\sigma$.

A prior probability is assigned to the alternative network model written in the form

$$P(W \mid A) = \frac{\exp(-\alpha C(W \mid A))}{Z(\alpha)}, \tag{13}$$

where

$$Z(\alpha) = \int \exp(-\alpha C(W \mid A))d^K W \tag{14}$$

is a measure of the characteristic network complexity. The posterior probability of the network model is then

$$P(W \mid D, A) = \frac{\exp(-\alpha C(W \mid A) - \beta E(D \mid W, A))}{Z(\alpha, \beta)} \tag{15}$$

with

$$Z(\alpha, \beta) = \int \exp(-\alpha C(W \mid A) - \beta E(D \mid W, A)) d^K W. \tag{16}$$

Let $-I(\mathcal{M})$ be the log of the prior probability of the model $\mathcal{M}$, that is,

$$I(\mathcal{M}) = -\log P(W \mid A). \tag{17}$$

Let $-I(D \mid \mathcal{M})$ be the log probability of D according to $\mathcal{M}$:

$$I(D \mid \mathcal{M}) = -\sum_{i=1}^{N} \log P(y \mid x, W, A). \tag{18}$$

Then the probability that both $\mathcal{M}$ is true and D occurs is

$$p(\mathcal{M}) = \exp(-I(D, \mathcal{M})), \tag{19}$$

where

$$I(D, \mathcal{M}) = I(\mathcal{M}) + I(D \mid \mathcal{M}). \tag{20}$$

It is well known that this p results as the posterior probability of $\mathcal{M}$, and the model which maximizes $p(\mathcal{M})$ would be the best fit. For most real applications, $I(D, \mathcal{M})$ cannot be computed exactly because the probabilities involved are not known. But it is evident that minimization of the fitness function (8) approximates maximization of $p(\mathcal{M})$ under the assumption (12).

5. Simulation results

The convergence and generalization properties of the BGP method were studied on two classes of problems, of differing complexity: majority and parity. The majority function of n inputs (n odd) returns a 1 if more than half of the input units have a 1, otherwise it returns a 0. The parity function outputs a 1 if the number of 1's in the input pattern of size n is odd, otherwise it outputs a 0. These tasks were chosen because they have often been used to test neural net learning algorithms, and results can be compared with the standard solutions. It is important to observe that the genetic search is performed in a variable d-dimensional space, and the minimal d is usually much larger than the input size n, depending on the task.

In our experiments, we used the fitness function

$$F(D \mid W, A) = E'(D \mid W, A) + \frac{1}{N} C'(W \mid A) \tag{21}$$

	n	popsize	layers	units	weights	generations
majority	3	100	1	1	4	2
	5	100	2	2	15	8
	7	500	2	4	21	11
	9	1000	2	6	31	13
parity	2	100	2	3	9	2
	4	1000	2	6	49	9
	6	1000	3	12	105	31
	8	1000	3	15	136	83

Table 1: Network complexities for discovered minimal solutions, in terms of the number of layers, units, and weights. Also shown is the number of generations required to obtain the solution. An elitist selection strategy with top 20% truncation was used.

where E' is a normalized version of equation (5),

$$E'(D \mid W, A) = \frac{E(D \mid W, A)}{m \cdot N}, \tag{22}$$

where m denotes the number of output units and N the size of the training set. Note that the error term satisfies $0 \leq E'(D \mid W, A) \leq 1$. C' is a revised measure of network complexity, defined as

$$C'(W \mid A) = \frac{C(W \mid A) + L(A) + U(A)}{C_{max}}, \tag{23}$$

where $L(A)$ and $U(A)$ denote the number of layers and units, respectively. C_{max} is a normalization factor used for the complexity term to satisfy $0 < C'(W \mid A) \leq 1$.

In all experiments we set $C_{max} = 1000$, assuming that the problems can be solved by $C(W \mid A) + L(A) + U(A) \leq 1000$. The $L(A)$ term penalizes a deep architecture which requires a large execution time after training. The $U(A)$ term penalizes a large number of units whose realization is more expensive than weights. The normalization of the functions does not hinder the probabilistic interpretation of network learning, because we are using a ranking-based selection strategy, not proportionate selection: for survival, only the ranking is important. Note that in (21) the complexity term $C'(W \mid A)$ is divided by N, the number of training examples, so that the error term plays a major role in determining the total fitness value of the network.

We performed two separate series of experiments. In the first, we were interested to see whether the BGP method would be able to find minimal or subminimal solutions at all and, if so, how the method would scale with problems of increasing complexity. In these experiments, the entire set of $N = 2^n$ examples was used to evaluate the fitness of the individual networks,

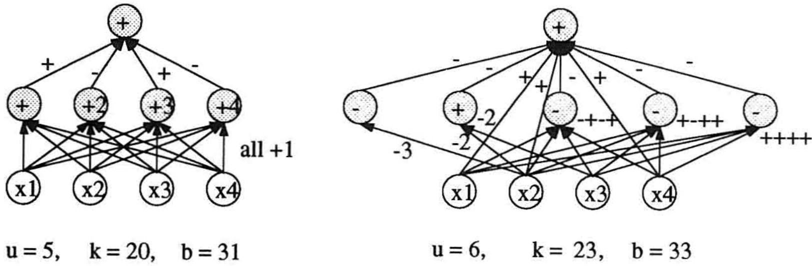

Figure 6: Solutions for the 4-input parity problem. Compared with the known minimal solution (left), the typical solution found by BGP (right) contains one more unit (u), three additional connection weights (k), and two more binary-valued connections (b) than the minimal solution.

and the examples were noise-free. For the second series of experiments, we tested the performance of BGP on noisy data. The generalization performance and learning speed of different strategies were compared, to study the effect of Occam's razor.

The results for the first experiments are summarized in table 1, which shows the complexity of minimal solutions discovered and the time required (in generations). The number of weights is given in the table in terms of the number of connections and thresholds with binary values. For all experiments, the top 20% of the population was selected for mating. The most fit individual was always retained in the new generation (truncation selection with an elitist strategy).

The network counterpart of most of the solutions was found to be minimal or subminimal in comparison to the standard solutions. This is illustrated in figure 6, which depicts a solution for the 4-input parity problem as found by this method. For comparison, the minimal solution for this problem is also depicted. Whereas the fitness value of the solution found by our method is $F_{found} = 0.0026$, that of the minimal solution is $F_{min} = E' + (Weights + Layers + Units)/(2^4 \cdot 1000) = 0.0024$. No general learning method is yet known to find such a solution (in terms of both architecture and weight values). Most existing search methods, including iterated hillclimbing methods [4, 18, 31], simulated annealing [10], backpropagation [29], and even other GAs [2], work on a search space of fixed size, while our search space is of variable size. This difference of ability, combined with the difference in parameters used in each algorithm, makes the comparison of learning speeds difficult.

The fitness function worked well in balancing the ability to solve the problem with the parsimony of the solution. A typical evolution of network complexity is shown in figure 7. Globally, the complexity of the network grows during evolution; locally, growth and pruning is repeated to fit errors on the one hand and to minimize the complexity of the network on the other. The corresponding evolution of the fitness values of the best individuals in each generation is depicted in figure 8. It is interesting to note that the global

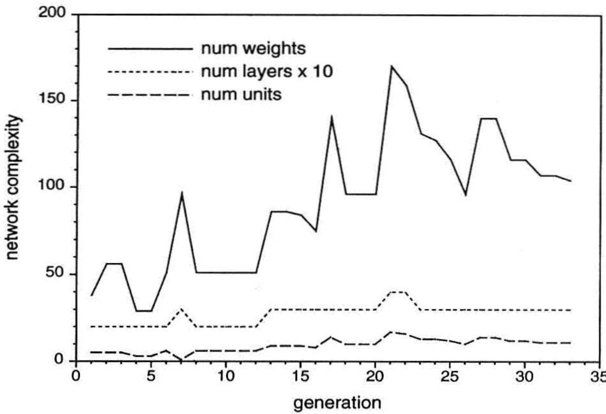

Figure 7: The evolution of network complexity, in terms of the number of weights C, layers L, and units U for the most fit individual in each generation. Growth and pruning are repeated to find an optimal complexity that is parsimonious yet large enough to solve the problem.

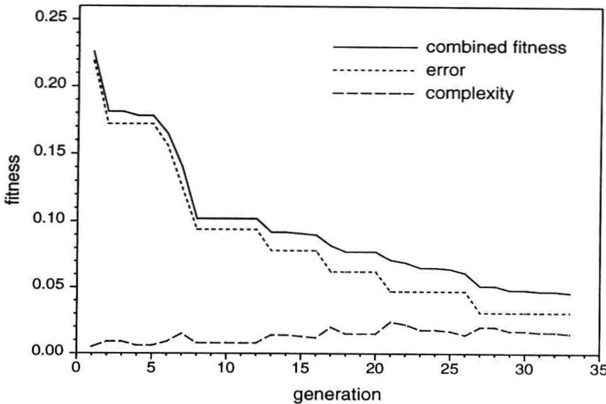

Figure 8: The evolution of the network fitness F, decomposed into the normalized error E' and the extended complexity C'. In spite of a fixed factor motivated by Occam's razor, the relative importance of the complexity term increases as evolution proceeds.

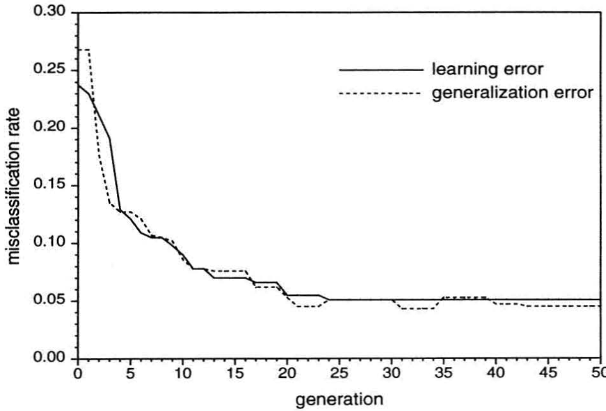

Figure 9: The evolution of network performance, for noisy data, of the nine-input majority function. Also shown is the generalization performance on the complete test set of noise-free examples.

method	layers	units	weights
$F = E$	9.7 ± 1.9	153.0 ± 76.4	1026.0 ± 526.1
$F = E + C$	3.7 ± 0.3	19.7 ± 1.1	123.1 ± 10.3

Table 2: Network complexity with and without Occam's razor

behavior of this optimization method is comparable with that of the group method of data handling (GMDH) in which additional terms are incrementally added to the existing polynomial approximator to achieve a minimal description length model of a complex system [9, 34].

The performance of the BGP method on noisy data was tested using the majority problem with nine inputs. In each run, we used a training set of 256 examples with 5% noise (which means that, on average, 12 or 13 examples out of 256 have false output value). Population size was 1000, and the top 20% most fit individuals were selected to mate. Figure 9 shows a typical evolution of the fitness value of most fit individuals until the 50th generation. For comparison we also depict the generalization performance on the complete test set, consisting of 512 noise-free examples. Although the test set was not used for selection, the training error and the generalization error correspond well.

The performance of the BGP method using the fitness function of (21) was compared with a method that uses the error term alone as the fitness measure, that is, $F(D \mid W, A) = E'(D \mid W, A)$. Both mothods used the same noisy data from the 9-majority problem. For each method, 10 runs were executed until the 50th generation to observe the training and generalization

method	learning	generalization	learning time
$F = E$	$95.2 \pm 0.7\%$	$92.4 \pm 1.4\%$	20294.7 ± 3019.4
$F = E + C$	$92.9 \pm 2.1\%$	$92.7 \pm 1.6\%$	5607.2 ± 67.2

Table 3: Comparison of performance with and without Occam's razor

performance of the solutions. Table 2 shows the average network size found at the 50th generation. The corresponding performance and learning time are shown in table 3. The learning time is measured in millions of evaluations of arithmetic operations associated with calculating activation values of neural units. The results show that using Occam's razor leads to decreased performance on the training set, but eventually results in an improved generalization performance. We attribute this effect to the tendency of Occam's razor to avoid overfitting to noisy data. Another advantage of Occam's razor is accelerated convergence. In our experiments, the proposed fitness function decreased the network size by an order of magnitude, and learning was accelerated by a factor of approximately 4.

In general, the method evolved a subminimal architecture that was, in most cases, an optimal solution (in terms of the parameters chosen for balancing the error fitting ability with the complexity of the solution). For some classes of large problems, however, the convergence was very slow. For a reasonably large class of binary functions of size n, there exists no simple optimization method that performs better than any other. To be effective, every sophisticated optimization method must be tuned to the application [22]. In order to assess the complexity of an optimization problem, and to speed up the genetic search, an investigation of its fitness landscapes is necessary.

6. Analysis of fitness landscapes

Fitness landscapes have been analyzed for Boolean N-K networks by Kauffman [3], for random traveling salesman problems (TSPs) by Kirkpatrick et al. [10], and for Euclidean TSPs by Mühlenbein [21]. The general characterization of a fitness landscape is very difficult. The number of local optima, their distribution, and the basins of attraction are some of the important parameters which describe a fitness landscape. For the evaluation of search strategies, more specific questions must be answered:

- What is the distribution of local optima if the error term alone is used for the fitness function?
- How does the distribution of local optima change if the search space is enlarged?

These two questions are first steps towards a statement of the general problem:

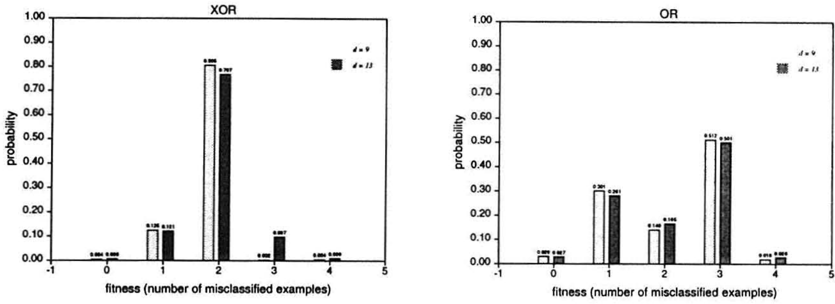

Figure 10: Fitness distribution.

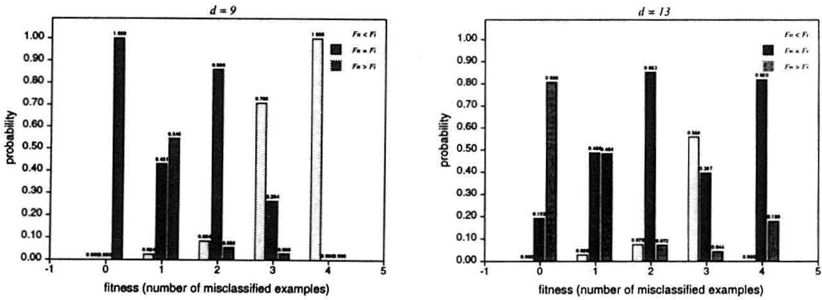

Figure 11: Fitness distribution of neighbors for each fitness value (XOR).

- Does the fitness function of (21) make the fitness landscape simpler or more complex when compared to an error-based fitness function with a fixed minimal network architecture?

These questions have been studied in the context of two problems: XOR and OR functions of two inputs. For each problem, we analyzed two search spaces of different dimension. One was a feedforward network of 2-2-1 architecture that had nine free parameters (six binary weights plus three binary thresholds). The other search space was a 2-3-1 architecture having thirteen free parameters (nine binary weights plus four binary thresholds). In describing these landscapes, we must focus on their statistical characteristics, because the spaces are too large to list all the details. For this analysis, the fitness function consisted of the error term only; the coefficient α in (8) was set to zero, and $\beta = 1$.

The fitness distributions are shown as bargraphs in figure 10. Note that each of the XOR and OR networks had two binary inputs, resulting in four input-output pairs. Hence, a specific network could have one of only five fitness values (0 if all four examples are classified correctly, 1 if one example is classified incorrectly, and so on). The analysis shows that the XOR-9 network

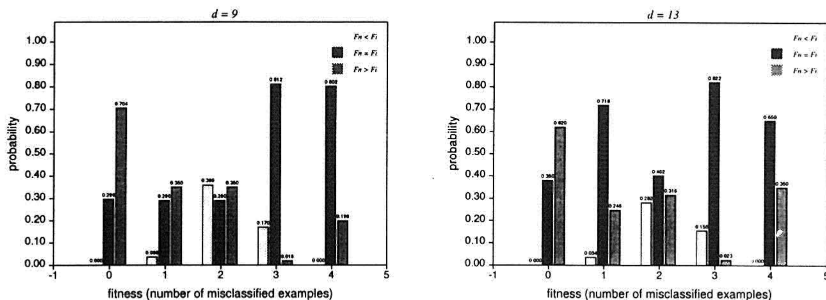

Figure 12: Fitness distribution of neighbors for each fitness value (OR)

had only two (= 0.4%) isolated global optima, while the OR-9 net had fifteen (= 2.9%) optima. Growth of the dimension from 9 to 13 increased the proportion of optima of XOR by 0.2%, but reduced that of OR by 0.2%. The bargraphs also show that the fitness of OR-9 was more uniformly distributed than that of XOR-9, suggesting that a search step in the OR network space would get more information than a step in the XOR space.

To see how the local optima vary, we computed the probability of an individual i finding a better, same, and worse fit neighbor n by a *single* mutation (figures 11 and 12). By a better fit neighbor n of i, we mean that F_n is smaller than F_i, since we attempt to minimize the fitness function. We see, for instance, that the probability of finding a better neighbor for XOR-9 is only 8.4% if the fitness of the individual is 0.5. For OR, the corresponding probability is 36.0%. A very important result can be concluded from the bargraphs for fitness value 0 in figures 11 and 12. For XOR with a minimal network architecture ($d = 9$), all global minima are isolated; no neighbors are global optima. However, for the enlarged search space ($d = 13$), there is a 19.2% chance that another global optimum can be reached by one-bit mutation. The same behavior can be observed for the OR problem. This analysis suggests that the increase of the dimensionality of the search space from 9 to 13 leads to a change in the fitness distributions and landscapes, which in turn can make it easier to train the weights.

We also computed the probability of a configuration finding a better fit neighbor by steepest-descent hillclimbing, that is, by looking at *all* of its neighbors at Hamming distance 1. Not surprisingly for this kind of landscape, there is for XOR a less than 50% chance of finding a better configuration. For OR, the probability is about 70%. This means steepest-descent hillclimbing would be effective for OR, but not for XOR. This explains in part why our experiments showed a better scaling property for the majority function (a kind of OR) in comparison to the parity problem (whose smallest size is XOR).

7. Discussion and conclusions

We have presented an evolutionary method for learning both the architecture and the weights of a network at the same time. This method uses trees to represent a feedforward network whose size and topology are dynamically adapted by genetic operators. A new fitness function that uses Occam's razor has been proposed, which proved to work well for the class of problems studied. Simulation results indicate that, given enough resources, the method finds minimal-complexity networks. Experiments on noisy data show that using Occam's razor not only improves the generalization performance, but accelerates the convergence of genetic programming as well. Extensions and refinements are expected in the following areas.

Information about the fitness landscape can be used to accelerate convergence. As we have shown, the fitness landscapes are characterized by large plateaus. The basin of attraction of the global optimum is fairly small. We have also seen that the fitness landscapes are changed by modifying the architectures. It is expected that fitness landscapes will generally have large plateaus as the network complexity approaches a minimum, which makes it difficult for a hillclimber to reach the minimum. A possible method of accelerating the convergence speed would be to start with networks that are supposed to be larger than minimal, and to prune the network with the Occam factor. This is supported by the results of the landscape analysis; the increase of the dimensionality of the search space leads to a larger chance of finding better solutions in the near of global optima.

Another factor that merits future study is the effect of the training set on the convergence speed and generalization performance of the algorithm. GP involves a time-consuming process of evaluating training examples. The fitness evaluation time can be shortened enormously if we have an efficient method for selecting examples critical to specific tasks [38, 37, 40]. The integration of active data selection to GP should improve the efficiency and scaling property of the method we have described.

Whereas we have used a simple next-ascent hillclimbing for adjustment of discrete weights, other traditional search methods might well be used for this purpose. Examples include iterated hillclimbing procedures developed in symbolic artificial intelligence [4, 18, 31]. The discrete-valued weights may be extended to more general real-valued weights. In this extension, it will be necessary to modify or replace the discrete hillclimbing search by a continuous parameter optimization method, which could be GAs [25, 30] or conventional gradient-based search methods [29]. Note that this adaptation does not change the top-level structure of the BGP method described in figure 4.

As opposed to conventional learning algorithms for neural networks, the GP method makes relatively few assumptions about network types. Thus, the same method also can be used to breed other network architectures, such as networks of radial basis functions, sigma-pi units, or any mixture, instead of the threshold or sigmoid units. The potential for evolving neural

architectures that are customized for specific applications is one of the most interesting properties of GAs. On the other hand, neural net optimization provides a very interesting problem worthy of further theoretical study from the GA point of view. For example, the problem we have discussed involved chromosomes of variable lengths, through which the fitness landscape was modified during evolution. This kind of optimization problem can be contrasted to the usual applications of GAs, in which the search space is fixed.

The ultimate usefulness of the BGP method must be tested by implementing it in systems that solve real-world problems, such as pattern recognition or time-series prediction. To this end, we may need extensions to the current implementation. We believe, however, that the general framework and the fitness function provided in this paper are of value, because the problem of balancing the accuracy with the complexity of a solution is fundamental in both neural networks and genetic programming.

Acknowledgments

This research was supported in part by the Real-World Computing Program, under the project SIFOGA (Statistical Inference as a Foundation of Genetic Algorithms). The authors thank Jürgen Bendisch, Frank Śmieja, Dirk Schlierkamp-Voosen, and the other members of the learning systems research group of the GMD Institute for Applied Information Technology for their valuable discussions and suggestions. We also wish to thank the anonymous reviewers whose comments helped to improve the clarity of the paper.

References

[1] Y. S. Abu-Mostafa, "The Vapnik-Chervonenkis Dimension: Information versus Complexity in Learning," *Neural Computation*, **1** (1989) 312–317.

[2] T. Bäck and H.-P. Schwefel, "An Overview of Evolutionary Algorithms for Parameter Optimization," *Evolutionary Computation*, **1** (1993) 1–23.

[3] S. Kauffman and S. Levin, "Towards a General Theory of Adaptive Walks on Rugged Landscapes," *Journal of Theoretical Biology*, **128** (1987) 11–45.

[4] I. P. Gent and T. Walsh, "Towards an Understanding of Hill-climbing Procedures for SAT," in *Proceedings of the 11th National Conference on Artificial Intelligence (AAAI-93)*, 28–33, (Cambridge, MIT Press, 1993).

[5] D. E. Goldberg, *Genetic Algorithms in Search, Optimization & Machine Learning* (Reading, MA, Addison Wesley, 1989).

[6] F. Gruau, "Genetic Synthesis of Boolean Neural Networks with a Cell Rewriting Developmental Process," Tech. Rep., Laboratoire de l'Informatique du Parallélisme (1992).

[7] S. A. Harp, T. Samad, and A. Guha, "Towards the Genetic Synthesis of Neural Networks," in *Proceedings of the Third International Conference on Genetic Algorithms (ICGA-89)*, 360–369, (San Mateo, CA, Morgan Kaufmann, 1989).

[8] J. H. Holland, *Adaptation in Natural and Artificial Systems* (Ann Arbor, University of Michigan Press, 1975).

[9] A. G. Ivakhnenko, "Polynomial Theory of Complex Systems," *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-**1** (1971) 364–378.

[10] S. Kirkpatrick, C. D. Gelett, and M. P. Vecchi, "Optimization by Simulated Annealing," *Science*, **220** (1985) 621–630.

[11] H. Kitano, "Designing Neural Networks Using Genetic Algorithms with Graph Generation System," *Complex Systems*, **4** (1990) 461–476.

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

[13] D. J. C. MacKay, "Bayesian Methods for Adaptive Models," Ph.D. thesis, California Institute of Technology, Pasadena, CA. (1992).

[14] W. S. McCulloch and W. Pitts, "A Logical Calculus of the Ideas Immanent in Nervous Activity," *Bulletin of Mathematical Biophysics*, **5** (1943) 115–133.

[15] G. F. Miller, P. M. Todd, and S. U. Hegde, "Designing Neural Networks Using Genetic Algorithms," in *Proceedings of the Third International Conference on Genetic Algorithms (ICGA-89)*, 379–384 (San Mateo, CA, Morgan Kaufmann, 1989).

[16] M. Minsky and S. Papert, *Perceptrons: An Introduction to Computational Geometry* (Cambridge, MIT Press, 1988).

[17] D. Montana and L. Davis, "Training Feedforward Neural Networks Using Genetic Algorithms," in *Proceedings of the International Joint Conference on Artificial Intelligence* (1989).

[18] P. Morris, "The Breakout Method for Escaping from Local Minima," in *Proceedings of the 11th National Conference on Artificial Intelligence (AAAI-93)*, 40–45, (Cambridge, MIT Press, 1993).

[19] H. Mühlenbein, "Darwin's Continental Cycle and Its Simulation by the Prisoner's Dilemma," *Complex Systems*, **5** (1991) 459–478.

[20] H. Mühlenbein, "Evolution in Time and Space—The Parallel Genetic Algorithm," in *Foundations of Genetic Algorithms*, 316–338, edited by G. Rawlins (San Mateo, CA, Morgan Kaufmann, 1991).

[21] H. Mühlenbein, "Parallel Genetic Algorithms in Combinatorial Optimization," in *Computer Science and Operations Research*, 441–456, edited by G. Balci, R. Sharda, and S. A. Zenios (Oxford, Pergamon, 1992).

[22] H. Mühlenbein, "Evolutionary Algorithms: Theory and Applications," in *Local Search in Combinatorial Optimization*, edited by E. H. L. Aarts and J. K. Lenstra (New York, Wiley, 1993).

[23] H. Mühlenbein, M. Gorges-Schleuter, and O. Krämer, "New Solutions to the Mapping Problem of Parallel Systems—The Evolution Approach," *Parallel Computing*, **4** (1987) 269–279.

[24] H. Mühlenbein and J. Kindermann, "The Dynamics of Evolution and Learning—Towards Genetic Neural Networks," in *Connectionism in Perspective*, 173–197, edited by R. Pfeifer *et al.*, (New York, Elsevier, 1989).

[25] H. Mühlenbein and D. Schlierkamp-Voosen, "Predictive Models for the Breeder Genetic Algorithm I: Continuous Parameter Optimization," *Evolutionary Computation*, **1** (1993) 25–49.

[26] T. Poggio and F. Girosi, "Networks for Approximation and Learning," *Proceedings of the IEEE*, **78** (1990) 1481–1497.

[27] I. Rechenberg, *Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution* (Stuttgart, Frommann-Holzboog, 1973).

[28] F. Rosenblatt, *Principles of Neurodynamics* (Washington D.C., Spartan Books, 1962).

[29] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Internal Representations by Error-Propagation," in *Parallel Distributed Processing*, Vol. I, 318–362, edited by D. E. Rumelhart and J. L. McClelland (Cambridge, MIT Press, 1986).

[30] H.-P. Schwefel, *Numerical Optimization of Computer Models* (Chichester, Wiley, 1981).

[31] B. Selman and H. A. Kautz, "An Empirical Study of Greedy Local Search for Satisfiability Testing," in *Proceedings of the 11th National Conference on Artificial Intelligence (AAAI-93)*, 46–51, (Cambridge, MIT Press, 1993).

[32] F. Śmieja, "Neural Network Constructive Algorithms: Trading Generalization for Learning Efficiency?" *Circuits, Systems, and Signal Processing*, **12** (1993) 331–374.

[33] R. Sorkin, "A Quantitative Occam's Razor," *International Journal of Theoretical Physics*, **22** (1983) 1091–1104.

[34] M. F. Tenorio and W.-T. Lee, "Self-Organizing Network for Optimum Supervised Learning," *IEEE Transactions on Neural Networks*, **1** (1990) 100–110.

[35] N. Tishby, E. Levin, and S. A. Solla, "Consistent Inference of Probabilities in Layered Networks: Predictions and Generalization," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN-89)*, Vol. II, 403–409 (New York, IEEE, 1989).

[36] D. Whitley, T. Starkweather, and C. Bogart, "Genetic Algorithms and Neural Networks: Optimizing Connections and Connectivity," *Parallel Computing*, **14** (1990) 347–361.

[37] B. T. Zhang, *Learning by Genetic Neural Evolution*, (in German), ISBN 3-929037-16-5, (Sankt Augustin, Germany, Infix-Verlag, 1992). Also available as Informatik Berichte No. 93, Institut für Informatik I, Universität Bonn (Bonn, Germany, July 1992).

[38] B. T. Zhang, "Accelerated Learning by Active Example Selection," forthcoming in *International Journal of Neural Systems* (1993).

[39] B. T. Zhang and H. Mühlenbein, "Genetic Programming of Minimal Neural Nets Using Occam's Razor," in *Proceedings of the Fifth International Conference on Genetic Algorithms (ICGA-93)*, 342-349, edited by S. Forrest (San Mateo, CA, Morgan Kaufmann, 1993).

[40] B. T. Zhang and G. Veenker, "Focused Incremental Learning for Improved Generalization with Reduced Training Sets," in *Artificial Neural Networks: Proceedings of the International Conference on Artificial Neural Networks (ICANN-91)*, Vol. I, 227–232, edited by T. Kohonen et al. (New York, Elsevier, 1991).

[41] B. T. Zhang and G. Veenker, "Neural Networks That Teach Themselves through Genetic Discovery of Novel Examples," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN-91)*, Vol. I, 690–695 (New York, IEEE, 1991).