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
e-mail: zhang@gmd.de, muehlen@gmd.de

Abstract

Genetic algorithms have been used for neural networks in two main ways: to optimize the network architecture and to train the weights of a fixed architecture. While most previous work focuses on only one of these two options, this paper investigates an alternative evolutionary approach called Breeder Genetic Programming (BGP) in which the architecture and the weights are optimized simultaneously. 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 size networks on clean data. The experiments on noisy data show that using Occam's razor not only improves the generalization performance, it also accelerates the convergence speed of evolution.

*Published in *Complex Systems*, 7(3): 199-220, 1993

1 Introduction

Constructing multilayer neural networks involves difficult optimization problems, i.e. 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 [8, 5, 20] have been used for solving both optimization problems [36]. In weight optimization, the set of weights is represented as a chromosome and a genetic search is applied on the encoded representation to find a set of weights that best fits the training data. Some encouraging results have been reported which are comparable with conventional learning algorithms [17]. In architecture optimization, the topology of the networks is encoded as a chromosome and some genetic operators are applied to find an architecture which fits best the specified task according to some explicit design criteria.

Optimization of neural network architectures or finding a minimal network for particular applications is important because the speed and accuracy of learning and performance are dependent on the 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 fast, 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, it needs, however, generally a large amount of training time [1, 32]. Therefore, the size of the network should be as small as possible, but sufficiently large to ensure an accurate fitting of the training set.

A general way of evolving genetic neural networks was suggested by Mühlenbein and Kindermann in [24]. Recent works, however, have focused on using genetic algorithms separately in each optimization problem, mainly in optimizing the network topology. Harp *et al.* [7] and Miller [15] have described representation schemes in which the anatomical properties of the network structure are encoded as bit-strings. Similar representation has also 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 which is evolved by genetic algorithms. All 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 representing neural networks, under the framework of so-called genetic programming, which en-

ables modification not only of the weights but also of the architecture for a neural network. 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, called breeder genetic programming (BGP), that employs an Occam's razor in the fitness function. The method makes an optimal trade-off between the error fitting ability and the parsimony of the network by preferring a simple network architecture to a complex one, given a choice of networks having the same fitting errors. The weights are trained not by backpropagation, but by a next-ascent hillclimbing search.

The organization of the paper is as follows. In Section 2, a grammar for representing multilayer feedforward neural networks is presented. 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 in Section 5, which is followed by an analysis of fitness landscapes in Section 6, and discussions 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 is no connection between units in the same layer. A commonly adopted architecture involves full connectivity between neighboring layers only. We allow both partial connectivity and direct connections between non-neighboring layers, since this is important for finding a parsimonious architecture. Specifically, this architecture allows for some of input units to be connected directly to output units. Figure 1 compares an usual multilayer perceptron and a more general architecture adopted in this work. There are also many options in the type of neural units. We will confine ourselves to McCulloch-Pitts neurons [14], although the method described below can be easily extended to employ other types of neurons.

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

The McCulloch-Pitts neuron is a binary device, i.e. it can be in only one of two possible states. Each neuron has a threshold. The neuron can receive inputs from excitatory and/or from 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 is 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 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].

For 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 label output, hidden and input units, respectively. Note that $R(i)$ can include input units as well as hidden units since direct connections between input and output units are possible, in which case the f_j is an identity function.

For the 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 a neural unit (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', 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 the simulations we used binary thresholds. McCulloch-Pitts neurons allow integer thresholds. Networks with binary thresholds can realize networks with integer thresholds by using additional neurons. Similarly, integer weights can also be realized by neurons using binary weights. The number of weights and units is usually 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 than a representation using integer weights directly. This does not mean, however, that the convergence will be accelerated automatically, because the search space is not reduced by using integers instead of binary weights. Another advantage of the binary over the integer weights is that it automatically functions as a regularizing factor by avoiding arbitrary growing of chromosome size.

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.

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 the usual genetic algorithms model a natural evolution, the BGA models a rational selection performed by human breeders. The BGA can be considered as a recombination between evolution strategies (ES) [27, 30] and genetic algorithms (GA) [8, 5]. The BGA uses truncation selection as performed by breeders. This selection scheme is similar to the (μ, λ)–strategy in ES [30]. The search process of the BGA is mainly driven by recombination, making the BGA a genetic algorithm. Our approach differs from the BGA in that we use variable size of chromosomes, a characteristic of genetic programming (GP) [12]. Thus we call the method Breeder Genetic Programming (BGP). BGP also differs from usual GP. While GP uses proportional selection combined with crossover as main operator, BGP uses truncation selection combined with crossover plus local hillclimbing. As will be shown later, 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 of each neural unit and its width are also chosen randomly. The $(g+1)$-st 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 τM individuals 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 will be derived in the next section which balances the error fitting ability and the parsimony of the networks. After selection, each individual in $\mathcal{B}(g)$ undergoes a hillclimbing search where the weights of the network are adapted by mutation. This results in the revised mate set $\mathcal{B}(g)$. 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. Notice that not only the size of individuals in one population may be different, $|A_i(g)| \neq |A_j(g)|$, $i \neq j$ and $i,\ j \in \{1, ..., M\}$,

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 $(g+1)$-st 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

but the size of same individual of subsequent population may also be different, $|A_i(g+1)| \neq |A_i(g)|, \;\; i \in \{1, ..., M\}$.

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}, i.e. the procedure terminates if

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

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

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 chromo-

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

some s_i of the network, the next-ascent hillclimbing procedure finds a better chromosome s_i^{new} by repeatedly applying the mutation operator until there is no weight configuration found having better fitness 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, of the tree by another, i.e. 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. Here the class U_k must first be found because not every value (node) can be mutated to arbitrary values. For example, a weight value must be drawn from the set $\{+1, -1\}$. The biases are mutated the same way as the weights. The index for the input units can be mutated by another input index.

Unlike the mutation, the crossover operator adapts the size and shape of the network architecture. A crossover operation starts by choosing at random two parent individuals from the mating pool $\mathcal{B}(g)$. Actual crossover of two individuals, i and j, is done 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 following conditions:

$$1 \leq c_i \leq Size(s_i) \quad \text{and} \quad 1 \leq c_j \leq Size(s_j).$$

Here, the length of an individual, $Size(s_i)$, 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' (Figure 5). The label of the nodes c_i and c_j must belong to the same class, i.e. either both Y-type or both 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 an Occam's razor

Occam's razor states that unnecessarily complex models should not be preferred to simpler ones [13, 33]. This section gives a quantitative Occam's razor for constructing minimal complexity neural networks by genetic algorithms.

In defining minimality, it is important that the network be able to approximate at least the training set to a specified performance level. A small network should be preferred to a large network *only if* both of them achieve a comparable performance. Otherwise, the algorithm would not reduce the approximation error, preferring smaller networks which can not be powerful enough to solve the task. So 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 squared errors between the desired and actual outputs:

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

with

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

Here 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 the architecture A and the set of weights W for the ith training input vector x_i.

The complexity of a neural network architecture is dependent on 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 the learning. Additionally, large weights should in general 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

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

where K is the number of free parameters. Notice that K can be arbitrarily large, because we fit the architectures too. In the case of binary weights, C reduces to the number of synaptic connections. This complexity measure 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 which we try to *minimize* is defined as

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

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 see this, let us define the following. Let D be the training data set for the function $\gamma : X \rightarrow Y$, i.e.

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

Then a model $\mathcal{M}$ of the function γ is an assignment to each possible pair (x, y) of a number $P(y|x)$ representing the hypothetical probability of y given x. That is, a network with specified architecture A and weights W is viewed as a model $\mathcal{M} = \{A, W\}$ predicting the outputs y as a function of input x in accordance with the probability distribution [35]:

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

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

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

is a normalizing constant. 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

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

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

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

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

where

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

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

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

with

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

Now let $-I(\mathcal{M})$ be the log of the prior probability of the model $\mathcal{M}$, i.e.

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

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

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

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

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

where

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

It is well known 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, $I(D, \mathcal{M})$ can not be computed exactly because the involved probabilities are not known. But it is easily seen 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 with different 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 the 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 the experiments, we used the fitness function

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

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

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

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

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

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 $0 < C'(W|A) \leq 1$.

In all experiments we set $C_{max} = 1000$, assuming that the problems can be solved by $C(W|A) + L(A) + U(A) \leq 1000$. 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 the network learning, because we are using a ranking-based selection strategy, not proportionate selection: for the survival only the ranking is of importance. Notice in Eqn. (21) that the complexity term $C'(W|A)$ is divided by N, the number of training examples, to have the error term play a major role in determining the total fitness value of the network. This ensures a small network be preferred to a large network only if both of them achieve a comparable performance.

We performed two kinds of experiments separately. In the first, we are interested in whether the BGP method is able to find minimal or subminimal solutions at all and, if yes, how the method scales 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. The examples were noise-free. For the second series of experiments, we tested the performance of BGP on noisy data. The generalization performance and the learning speed of different strategies are compared to study the effect of Occam's razor.

The results for the first experiments are summarized in Table 1. It shows the complexity of discovered minimal solutions and the required time in generations. The number of weights given in the table is 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 elitest strategy). For most of the solutions, their network counterpart was found to be minimal or subminimal in comparison to the known standard solutions. This is illustrated in Figure 6 which depicts a solution for the 4-input parity problem found by the method.

For comparison, the minimal solution for this problem is also depicted. Whereas the fitness value of the minimal solution is $F_{min} = E' + (Weights + Layers + Units)/(2^4 \cdot 1000) = 0.0024$, that of the found solution is $F_{found} = 0.0026$. Note that the standard minimal solution is shown for illustration

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

purposes. No general learning methods are yet known to find such a solution (architecture plus weight values). Most existing search methods, including iterated hillclimbing methods [4, 18, 31], simulated annealing [10], Backpropagation [29] and even other genetic algorithms [2], work on a search space of fixed size, while our search space is of variable size. This difference of ability combined with different parameters used in each algorithm make the comparison of learning speed difficult.

The fitness function worked well in balancing the ability to solve the problem and 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, while locally growth and pruning is repeated to fit errors on one hand and to minimize the complexity of the network on the other hand. The corresponding evolution of the fitness values of the best individuals in each generation is depicted in Figure 8. It is interesting to notice that the global behavior of this optimization method is comparable with 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 with the majority problem of 9 inputs. Unlike in the previous experiments where all possible examples are used without noise insertion, we used in each run a training set of 256 examples with 5% noise. This means, on average, 12 or 13 examples out of 256 have false output value. Population size was 1000 and upper 20% best individuals were selected to mate. Figure 9 shows a typical evolution of the fitness value of best individuals until the 50th generation. For comparison we also depict the generalization performance on the complete test set consisting of 512 noise-free examples. Notice that 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 (21) was compared with a method that uses just the error term as the fitness measure, i.e. $F(D|W,A) = E'(D|W,A)$. Both mothods used the same noisy data of the 9-majority problem. For each method, 10 runs were executed until the 50th generation to observe the training and generalization 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

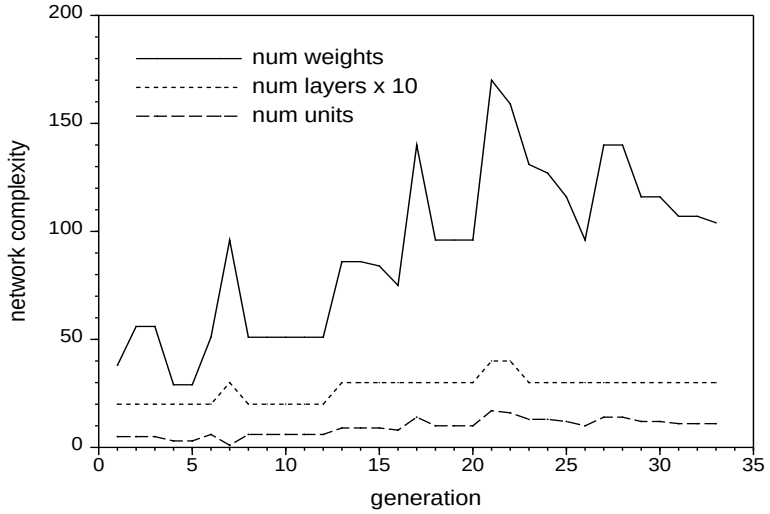

Figure 7: The evolution of network complexity in terms of the number of weights, layers, and units for the best individual in each generation. Growth and pruning is repeated to find an optimal complexity which is parsimonious but large enough to solve the problem.

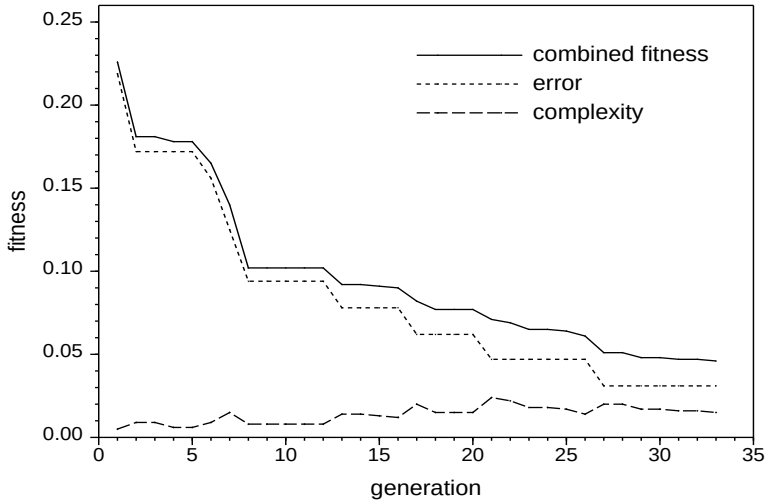

Figure 8: The evolution of the network fitness F decomposed into the normalized error E' and the extended complexity C'. In spite of a fixed Occam factor, the relative importance of the complexity term increases as evolution proceeds.

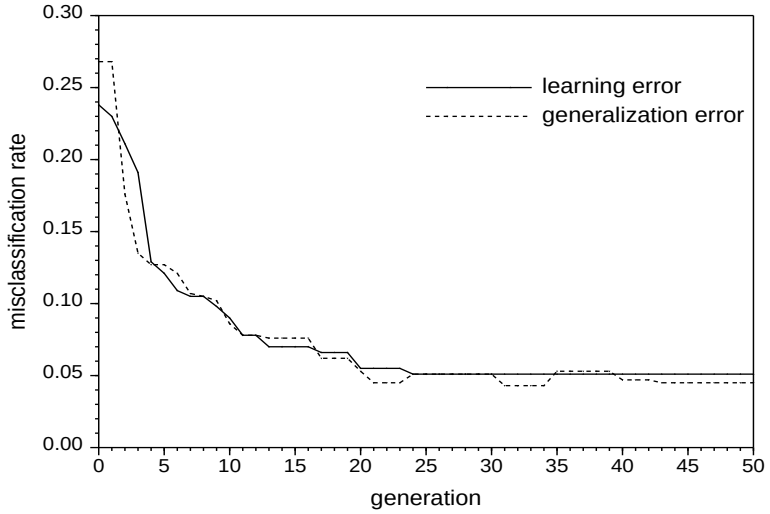

Figure 9: The evolution of the network performance for noisy data of the 9-input majority function. Also shown is the generalization performance on the complete test set of noise-free examples.

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

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

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

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. This is supposed to be the effect of Occam's razor for avoiding overfitting to noisy data. Another advantage of using Occam's razor is the accelerated convergence. In the above experiments, the proposed fitness function decreased the network size by an order of magnitude and the speed-up factor of learning was approximately four.

In general, the method evolved a subminimal architecture which is in most cases an optimal solution in terms of the parameters chosen for balancing the error fitting ability and the complexity of the solution. For some classes of large problems, however, the convergence was very slow. A simple optimization method does not exist which performs better than any other optimization method for a reasonable large class of binary functions of size n. To be effective, every sophisticated optimization method has to be tuned to the application [22]. In order to assess the complexity of an optimization problem and to speed up the genetic search further, 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 (TSP) problems by Kirkpatrick *et al.* [10], and for Euclidean TSP problems 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 have to be answered:

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

These two questions are first steps towards the general problem

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

The questions have been studied in the context of two problems: XOR and OR function of two inputs. For each problem we analyzed two search spaces of different dimension. One was a feedforward network of 2-2-1 architecture which has 9 free parameters (6 binary weights plus 3 binary thresholds). The other search space was a 2-3-1 architecture having 13 free parameters (9 binary weights plus 4 binary thresholds). In describing the landscapes, we have to focus on the statistical characteristics of them because the spaces are too large to list all the details. For the 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 in Figure 10 as bargraphs. Notice that each of the XOR and OR networks has two binary inputs, resulting in four input-output pairs. Hence a specific network can have only one of five fitness values (0 in case of all four examples are classified correctly, 1 if one example is classified incorrectly, and so on). The analysis shows that the XOR-9 network has only two (0.4%) isolated global optima, while the OR-9 net has fifteen (2.9%) optima. Growth of the dimension from 9 to 13 increases the proportion of optima of XOR by 0.2%, but reduced that of OR by 0.2%. The bargraphs also shows that the fitness of OR-9 is 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, respectively (Figure 11 and 12). Here, a better fit neighbor n of i means F_n is smaller than F_i, since we attempt to minimize the fitness function. The shows, for instance, that for XOR-9 the probability of finding a better neighbor 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 the 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 a global optimum. But for the enlarged search space ($d = 13$), there is a chance of 19.2% 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

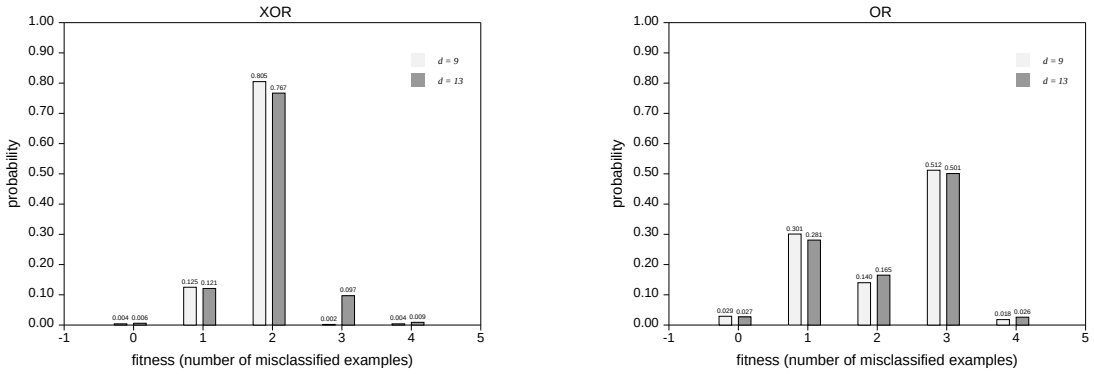

Figure 10: Fitness distribution

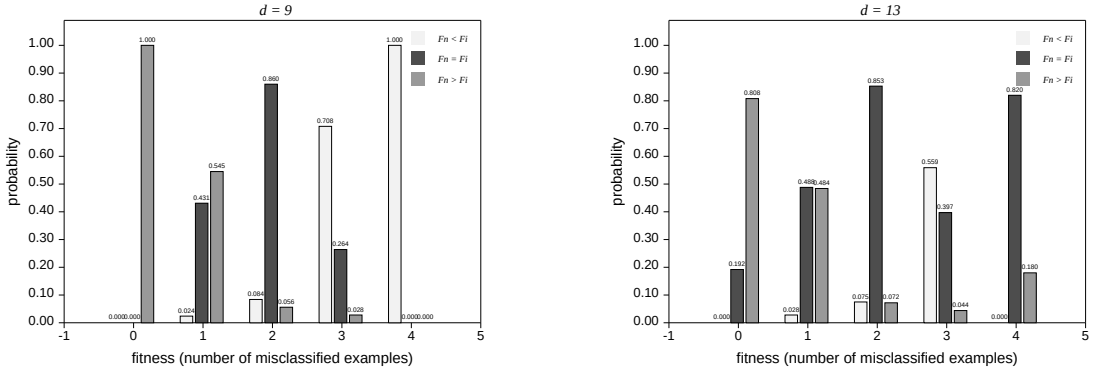

Figure 11: Fitness distribution of neighbors for each fitness value (XOR)

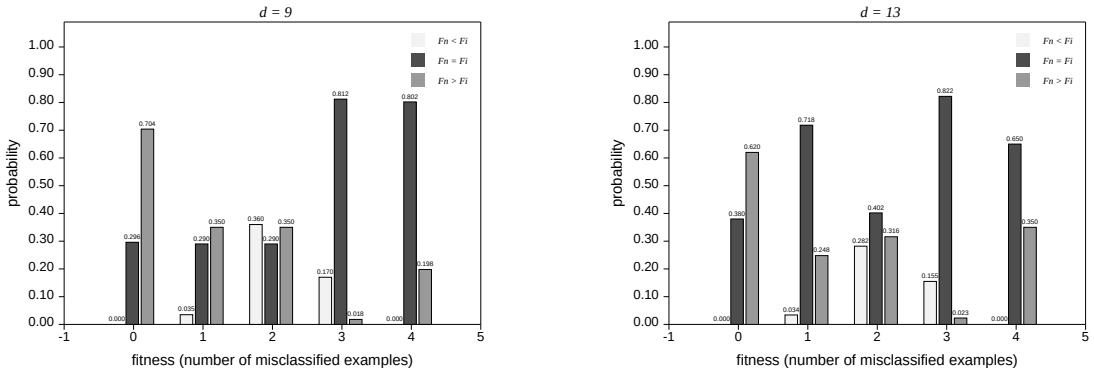

Figure 12: Fitness distribution of neighbors for each fitness value (OR)

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, i.e. by looking at *all* its neighbors at Hamming distance 1. Not surprisingly for this kind of landscape, one has 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 good 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 called breeder genetic programming (BGP) for learning both the network architecture and the weights at the same time. The method uses trees to represent a feedforward network whose size and topology are dynamically adapted by genetic operators. A new fitness function with an 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. The experiments on noisy data show that using Occam's razor not only improves the generalization performance, but it accelerates the convergence of genetic programming as well. Extensions and refinements are expected in the following areas.

The information about the fitness landscape can be used to speed up convergence. As was 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 larger networks (than are supposed to be minimal) and to let the network be pruned by 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 future work concerns the study of other factors, for instance the effect of training set, on convergence speed and generalization performance of the algorithm. The genetic programming involves a time-consuming process of evaluating training examples. The fitness evaluation time can be saved enormously, if we have an efficient method for selecting examples critical to specific tasks [38, 37, 40]. The integration of active data selection to the genetic programming should improve the efficiency and scaling property of the method described above.

While we have used a simple next-ascent hillclimbing for adjustment of discrete weights, other traditional search methods might as well have been 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 may be again genetic algorithms [25, 30] or conventional gradient-based search methods [29]. Notice that this adaptation does not change the top-level structure of the breeder genetic programming method described in Figure 4.

As opposed to conventional learning algorithms for neural networks, the genetic programming method makes relatively few assumptions about the network types. Thus the same method can also be used to breed other network architectures, e.g. networks of radial basis functions, sigma-pi units, or any mixture of them, 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 genetic algorithms. On the other hand, neural net optimization provides a very interesting problem worthy of further theoretical study from the genetic algorithm point of view. For example, the problem we discussed had to handle variable length of chromosomes through which the fitness landscape is modified during evolution. This kind of optimization problem is contrasted with usual applications of genetic algorithms 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 some further extensions to the current implementation. We believe, however, that the general framework and the fitness function provided in this paper are of value since the problem of balancing the accuracy and the complexity of the solution is

fundamental in both neural networks and genetic programming.

Acknowledgements

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, (MIT Press, 1993).

[5] D. E. Goldberg, *Genetic Algorithms in Search, Optimization & Machine Learning* (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, (Morgan Kaufmann, 1989).

[8] J. H. Holland, *Adaptation in Natural and Artificial Systems*, (University of Michigan Press, Ann Arbor, 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* (MIT Press, 1992).

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

[14] W. S. McCulloch and W. Pitts, "A Logical Calculus of the Ideas Immanent in Nervous Activity," *Bull. Math. 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 (Morgan Kaufmann, 1989).

[16] M. Minsky and S. Papert, *Perceptrons: An Introduction to Computational Geometry* (MIT Press, 1969, 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, (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 (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 (Pergamon, Oxford, 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 (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.*, (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* (Spartan Books, Washington D.C., 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 (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, (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 (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, Infix-Verlag, Sankt Augustin (1992). Also available as Informatik Berichte No. 93, Institut für Informatik I, Universität Bonn (July 1992).

[38] B. T. Zhang, "Accelerated Learning by Active Example Selection," to appear 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 (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.* (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 (IEEE, 1991).