

A New Evolutionary System for Evolving Artificial Neural Networks

Xin Yao, *Senior Member, IEEE*, and Yong Liu

Abstract—This paper presents a new evolutionary system, i.e., EPNet, for evolving artificial neural networks (ANN's). The evolutionary algorithm used in EPNet is based on Fogel's evolutionary programming (EP). Unlike most previous studies on evolving ANN's, this paper puts its emphasis on evolving ANN's behaviors. This is one of the primary reasons why EP is adopted. Five mutation operators proposed in EPNet reflect such an emphasis on evolving behaviors. Close behavioral links between parents and their offspring are maintained by various mutations, such as partial training and node splitting. EPNet evolves ANN's architectures and connection weights (including biases) simultaneously in order to reduce the noise in fitness evaluation. The parsimony of evolved ANN's is encouraged by preferring node/connection deletion to addition. EPNet has been tested on a number of benchmark problems in machine learning and ANN's, such as the parity problem, the medical diagnosis problems (breast cancer, diabetes, heart disease, and thyroid), the Australian credit card assessment problem, and the Mackey–Glass time series prediction problem. The experimental results show that EPNet can produce very compact ANN's with good generalization ability in comparison with other algorithms.

Index Terms—Evolution, evolutionary programming, evolution of behaviors, generalization, learning, neural-network design, parsimony.

I. INTRODUCTION

ARTIFICIAL neural networks (ANN's) have been used widely in many application areas in recent years. Most applications use feedforward ANN's and the backpropagation (BP) training algorithm. There are numerous variants of the classical BP algorithm and other training algorithms. All these training algorithms assume a fixed ANN architecture. They only train weights in the fixed architecture that includes both connectivity and node transfer functions.¹ The problem of designing a near optimal ANN architecture for an application remains unsolved. However, this is an important issue because there are strong biological and engineering evidences to support that the function, i.e., the information processing capability of an ANN is determined by its architecture.

There have been many attempts in designing ANN architectures (especially connectivity²) automatically, such as various

constructive and pruning algorithms [5]–[9]. Roughly speaking, a constructive algorithm starts with a minimal network (i.e., a network with a minimal number of hidden layers, nodes, and connections) and adds new layers, nodes, and connections if necessary during training, while a pruning algorithm does the opposite, i.e., deletes unnecessary layers, nodes, and connections during training. However, as indicated by Angeline *et al.* [10], “Such structural hill climbing methods are susceptible to becoming trapped at structural local optima.” In addition, they “only investigate restricted topological subsets rather than the complete class of network architectures.”

Design of a near optimal ANN architecture can be formulated as a search problem in the architecture space where each point represents an architecture. Given some performance (optimality) criteria, e.g., minimum error, fastest learning, lowest complexity, etc., about architectures, the performance level of all architectures forms a surface in the space. The optimal architecture design is equivalent to finding the highest point on this surface. There are several characteristics with such a surface, as indicated by Miller *et al.* [11], which make evolutionary algorithms better candidates for searching the surface than those constructive and pruning algorithms mentioned above.

This paper describes a new evolutionary system, i.e., EPNet, for evolving feedforward ANN's. It combines the architectural evolution with the weight learning. The evolutionary algorithm used to evolve ANN's is based on Fogel's evolutionary programming (EP) [1]–[3]. It is argued in this paper that EP is a better candidate than genetic algorithms (GA's) for evolving ANN's. EP's emphasis on the behavioral link between parents and offspring can increase the efficiency of ANN's evolution.

EPNet is different from previous work on evolving ANN's on a number of aspects. First, EPNet emphasises the evolution of ANN behaviors by EP and uses a number of techniques, such as partial training after each architectural mutation and node splitting, to maintain the behavioral link between a parent and its offspring effectively. While some of previous EP systems [3], [10], [12]–[15], acknowledged the importance of evolving behaviors, few techniques have been developed to maintain the behavioral link between parents and their offspring. The common practice in architectural mutations was to add or delete hidden nodes or connections uniformly at random. In particular, a hidden node was usually added to a hidden layer with full connections. Random initial weights were attached to these connections. Such an approach tends to destroy the behavior already learned by the parent and create poor behavioral link between the parent and its offspring.

Manuscript received January 6, 1996; revised August 12, 1996 and November 12, 1996. This work was supported by the Australian Research Council through its small grant scheme.

The authors are with the Computational Intelligence Group, School of Computer Science, University College, The University of New South Wales, Australian Defence Force Academy, Canberra, ACT, Australia 2600.

Publisher Item Identifier S 1045-9227(97)02758-6.

¹Weights in this paper indicate both connection weights and biases.

²This paper is only concerned with connectivity and will use architecture and connectivity interchangeably. The work on evolving both connectivity and node transfer functions was reported elsewhere [4].

Second, EPNet encourages parsimony of evolved ANN's by attempting different mutations sequentially. That is, node or connection deletion is always attempted before addition. If a deletion is "successful," no other mutations will be made. Hence, a parsimonious ANN is always preferred. This approach is quite different from existing ones which add a network complexity (regularization) term in the fitness function to penalize large ANN's (i.e., the fitness function would look like $f = f_{\text{error}} + \alpha f_{\text{complexity}}$). The difficulty in using such a function in practice lies in the selection of suitable coefficient α , which often involves tedious trial-and-error experiments. Evolving parsimonious ANN's by sequentially applying different mutations provides a novel and simple alternative which avoids the problem. The effectiveness of the approach has been demonstrated by the experimental results presented in this paper.

Third, EPNet has been tested on a number of benchmark problems, including the parity problem of various sizes, the Australian credit card assessment problem, four medical diagnosis problems (breast cancer, diabetes, heart disease, and thyroid), and the Mackey–Glass time series prediction problem. It was also tested on the two-spiral problem [16]. Few evolutionary systems have been tested on a similar range of benchmark problems. The experimental results obtained by EPNet are better than those obtained by other systems in terms of generalization and the size of ANN's.

The rest of this paper is organized as follows. Section II discusses different approaches to evolving ANN architectures and indicates potential problems with the existing approaches, Section III describes EPNet in detail and gives motivations and ideas behind various design choices, Section IV presents experimental results on EPNet and some discussions, and finally Section V concludes with a summary of the paper and a few remarks.

II. EVOLVING ANN ARCHITECTURES

There are two major approaches to evolving ANN architectures. One is the evolution of "pure" architectures (i.e., architectures without weights). Connection weights will be trained after a near optimal architecture has been found. The other is the simultaneous evolution of both architectures and weights. Schaffer *et al.* [17] and Yao [18]–[21] have provided a comprehensive review on various aspects of evolutionary artificial neural networks (EANN's).

A. The Evolution of Pure Architectures

One major issue in evolving pure architectures is to decide how much information about an architecture should be encoded into a chromosome (genotype). At one extreme, all the detail, i.e., every connection and node of an architecture can be specified by the genotype, e.g., by some binary bits. This kind of representation schemes is called the direct encoding scheme or the strong specification scheme. At the other extreme, only the most important parameters of an architecture, such as the number of hidden layers and hidden nodes in each layer are encoded. Other detail about the architecture is either predefined or left to the training process to decide. This kind of

representation schemes is called the indirect encoding scheme or the weak specification scheme. Fig. 1 [20], [21] shows the evolution of pure architectures under either a direct or an indirect encoding scheme.

It is worth pointing out that genotypes in Fig. 1 do not contain any weight information. In order to evaluate them, they have to be trained from a random set of initial weights using a training algorithm like BP. Unfortunately, such fitness evaluation of the genotypes is very noisy because a phenotype's fitness is used to represent the genotype's fitness. There are two major sources of noise.

- 1) The first source is the random initialization of the weights. Different random initial weights may produce different training results. Hence, the same genotype may have quite different fitness due to different random initial weights used by the phenotypes.
- 2) The second source is the training algorithm. Different training algorithms may produce different training results even from the same set of initial weights. This is especially true for multimodal error functions. For example, a BP may reduce an ANN's error to 0.05 through training, but an EP could reduce the error to 0.001 due to its global search capability.

Such noise can mislead the evolution because of the fact that the fitness of a phenotype generated from genotype G_1 is higher than that generated from genotype G_2 does not mean that G_1 has higher fitness than G_2 . In order to reduce such noise, an architecture usually has to be trained many times from different random initial weights. The average results will then be used to estimate the genotype's fitness. This method increases the computation time for fitness evaluation dramatically. It is one of the major reasons why only small ANN's were evolved in previous studies [22]–[24].

In essence, the noise identified in this paper is caused by the one to many mapping from genotypes to phenotypes. Angeline *et al.* [10] and Fogel [3], [25] have provided a more general discussion on the mapping between genotypes and phenotypes. It is clear that the evolution of pure architectures has difficulties in evaluating fitness accurately. As a result, the evolution would be very inefficient.

B. The Simultaneous Evolution of Both Architectures and Weights

One way to alleviate the noisy fitness evaluation problem is to have a one to one mapping between genotypes and phenotypes. That is, both architecture and weight information are encoded in individuals and are evolved simultaneously. Although the idea of evolving both architectures and weights is not new [3], [10], [13], [26], few have explained why it is important in terms of accurate fitness evaluation. The simultaneous evolution of both architectures and weights can be summarized by Fig. 2.

The evolution of ANN architectures in general suffers from the permutation problem [27], [28] or called competing conventions problem [17]. It is caused by the many to one mapping from genotypes to phenotypes since two ANN's which order their hidden nodes differently may have different

1. Decode each individual (i.e., chromosome) in the current generation into an architecture. If the indirect encoding scheme is used, further detail of the architecture is specified by some developmental rules or a training process.
2. Train each neural network with the decoded architecture by a pre-defined learning rule/algorithm (some parameters of the learning rule could be learned during training) starting from different sets of random initial weights and, if any, learning parameters.
3. Define the fitness of each individual (encoded architecture) according to the above training result and other performance criteria such as the complexity of the architecture.
4. Reproduce a number of children for each individual in the current generation based on its fitness.
5. Apply genetic operators to the children generated above and obtain the next generation.

Fig. 1. A typical cycle of the evolution of architectures.

1. Evaluate each individual based on its error and/or other performance criteria such as its complexity.
2. Select individuals for reproduction and genetic operation.
3. Apply genetic operators, such as crossover and mutation, to the ANN's architectures and weights, and obtain the next generation.

Fig. 2. A typical cycle of the evolution of both architectures and weights. The word "genetic" used above is rather loose and should not be interpreted in the strict biological sense. Genetic operators are just search operators.

genotypes but are behaviorally (i.e., phenotypically) equivalent. This problem not only makes the evolution inefficient, but also makes crossover operators more difficult to produce highly fit offspring. It is unclear what building blocks actually are in this situation. For example, ANN's shown in Figs. 3(a) and 4(a) are equivalent, but they have different genotypic representations as shown by Figs. 3(b) and 4(b) using a direct encoding scheme. In general, any permutation of the hidden nodes will produce behaviorally equivalent ANN's but with different genotypic representations. This is also true for indirect encoding schemes.

C. Some Related Work

There is some related work to evolving ANN architectures. For example, Smalz and Conrad [29] proposed a novel approach to assigning credits and fitness to neurons (i.e.,

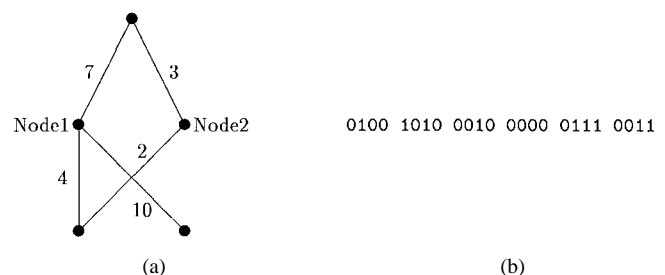


Fig. 3. (a) An ANN and (b) its genotypic representation, assuming that each weight is represented by four binary bits. Zero weight implies no connection.

nodes) in an ANN, rather than the ANN itself. This is quite different from all other methods which only evaluate a complete ANN without going inside it. The idea is to identify those neurons which "are most compatible with all of the network contexts associated with the best performance

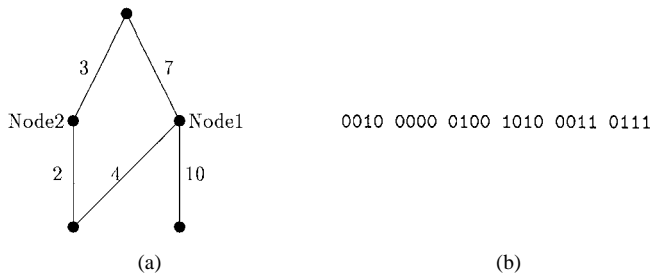


Fig. 4. (a) An ANN which is equivalent to that given in Fig. 3(a) and (b) its genotypic representation.

on any of the inputs” [29]. Starting from a population of redundant, identically structured networks that vary only with respect to individual neuron parameters, their evolutionary method first evaluates neurons and then copies with mutation the parameters of those neurons that have high fitness values to other neurons in the same class. In other words, it tries to put all fit neurons together to generate a hopefully fit network. However, Smalz and Conrad’s evolutionary method does not change the network architecture, which is fixed [29]. The appropriateness of assigning credit/fitness to individual neurons also needs further investigation. It is well known that ANN’s use distributed representation. It is difficult to identify a single neuron for the good or poor performance of a network. Putting a group of “good” neurons from different ANN’s together may not produce a better ANN unless a local representation is used. It appears that Smalz and Conrad’s method [29] is best suited to ANN’s such as radial basis function (RBF) networks.

Odri *et al.* [30] proposed a nonpopulation-based learning algorithm which could change ANN architectures. It uses the idea of evolutionary development. The algorithm is based on BP. During training, a new neuron may be added to the existing ANN through “cell division” if an existing neuron generates a nonzero error [30]. A connection may be deleted if it does not change very much in previous training steps. A neuron is deleted only when all of its incoming or all of its outgoing connections have been deleted. There is no obvious way to add a single connection [30]. The algorithm was only tested on the XOR problem to illustrate its ideas [30]. One major disadvantage of this algorithm is its tendency to generate larger-than-necessary ANN and overfit training data. It can only deal with strictly layered ANN’s.

III. EPNET

In order to reduce the detrimental effect of the permutation problem, an EP algorithm, which does not use crossover, is adopted in EPNet. EP’s emphasis on the behavioral link between parents and their offspring also matches well with the emphasis on evolving ANN behaviors, not just circuitry. In its current implementation, EPNet is used to evolve feedforward ANN’s with sigmoid transfer functions. However, this is not an inherent constraint. In fact, EPNet has minimal constraint on the type of ANN’s which may be evolved. The feedforward ANN’s do not have to be strictly layered or fully connected

between adjacent layers. They may also contain hidden nodes with different transfer functions [4].

The major steps of EPNet can be described by Fig. 5, which are explained further as follows [16], [31]–[34].

- 1) Generate an initial population of M networks at random. The number of hidden nodes and the initial connection density for each network are uniformly generated at random within certain ranges. The random initial weights are uniformly distributed inside a small range.
- 2) Partially train each network in the population on the training set for a certain number of epochs using a modified BP (MBP) with adaptive learning rates. The number of epochs, K_0 , is specified by the user. The error value E of each network on the validation set is checked after partial training. If E has not been significantly reduced, then the assumption is that the network is trapped in a local minimum and the network is marked with “failure.” Otherwise the network is marked with “success.”
- 3) Rank the networks in the population according to their error values, from the best to the worst.
- 4) If the best network found is acceptable or the maximum number of generations has been reached, stop the evolutionary process and go to Step 11). Otherwise continue.
- 5) Use the rank-based selection to choose one parent network from the population. If its mark is “success,” go to Step 6), or else go to Step 7).
- 6) Partially train the parent network for K_1 epochs using the MBP to obtain an offspring network and mark it in the same way as in Step 2), where K_1 is a user specified parameter. Replace the parent network with the offspring in the current population and go to Step 3).
- 7) Train the parent network with a simulated annealing (SA) algorithm to obtain an offspring network. If the SA algorithm reduces the error E of the parent network significantly, mark the offspring with “success,” replace its parent by it in the current population, and then go to Step 3). Otherwise discard this offspring and go to Step 8).
- 8) First decide the number of hidden nodes N_{hidden} to be deleted by generating a uniformly distributed random number between one and a user-specified maximum number. N_{hidden} is normally very small in the experiments, no more than three in most cases. Then delete N_{hidden} hidden nodes from the parent network uniformly at random. Partially train the pruned network by the MBP to obtain an offspring network. If the offspring network is better than the worst network in the current population, replace the worst by the offspring and go to Step 3). Otherwise discard this offspring and go to Step 9).
- 9) Calculate the approximate importance of each connection in the parent network using the nonconvergent method. Decide the number of connections to be deleted in the same way as that described in Step 8). Randomly delete the connections from the parent network according to the calculated importance. Partially train the pruned network by the MBP to obtain an offspring network. If the offspring network is better than the worst network in the current population, replace the worst by the offspring and go to Step 3). Otherwise discard this offspring and go to Step 10).

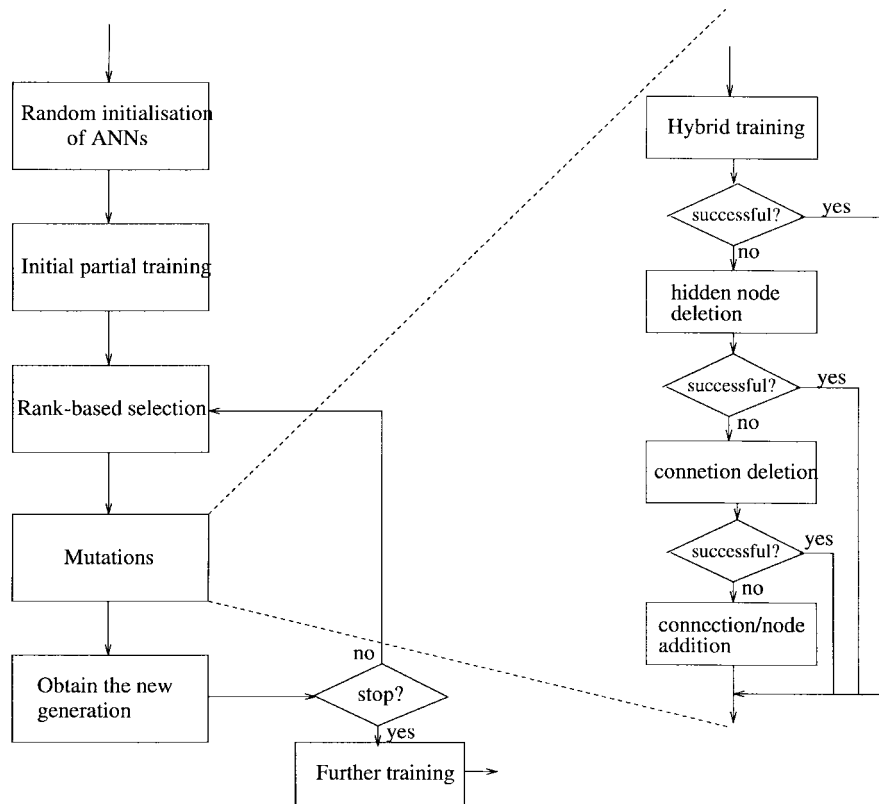


Fig. 5. Major steps of EPNet.

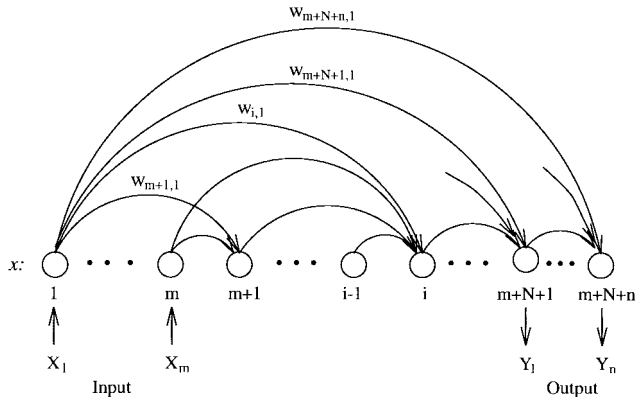


Fig. 6. A fully connected feedforward ANN [35, p. 273].

- 10) Decide the number of connections and nodes to be added in the same way as that described in Step 8). Calculate the approximate importance of each virtual connection with zero weight. Randomly add the connections to the parent network to obtain *Offspring 1* according to their importance. Addition of each node is implemented by splitting a randomly selected hidden node in the parent network. The new grown network after adding all nodes is *Offspring 2*. Partially train *Offspring 1* and *Offspring 2* by the MBP to obtain a survival offspring. Replace the worst network in the current population by the offspring and go to Step 3).
- 11) After the evolutionary process, train the best network further on the combined training and validation set until it “converges.”

The above evolutionary process appears to be rather complex, but its essence is an EP algorithm with five mutations: hybrid training, node deletion, connection deletion, connection addition, and node addition. Details about each component of EPNet are given in the following sections.

A. Encoding Scheme for Feedforward ANN's

The feedforward ANN's considered by EPNet are generalized multilayer perceptrons [35, pp. 272–273]. The architecture of such networks is shown in Fig. 6, where \mathbf{X} and \mathbf{Y} are inputs and outputs, respectively,

$$\begin{aligned}
 x_i &= X_i, & 1 \leq i \leq m \\
 \text{net}_i &= \sum_{j=1}^{i-1} w_{ij}x_j, & m < i \leq m + N + n \\
 x_j &= f(\text{net}_j), & m < j \leq m + N + n \\
 Y_i &= x_{i+m+N}, & 1 \leq i \leq n
 \end{aligned}$$

where f is the following sigmoid function:

$$f(z) = \frac{1}{1 + e^{-z}}$$

m and n are the number of inputs and outputs, respectively, N is the number of hidden nodes.

In Fig. 6, there are $m + N + n$ circles, representing all of the nodes in the network, including the input nodes. The first m circles are really just copies of the inputs X_1, \dots, X_m . Every other node in the network, such as node number i , which calculates net_i and x_i , takes inputs from every node

that precedes it in the network. Even the last output node (the $(m + N + n)$ th), which generates Y_n , takes input from other output nodes, such as the one which outputs Y_{n-1} .

The direct encoding scheme is used in EPNet to represent ANN architectures and connection weights (including biases). This is necessary because EPNet evolves ANN architectures and weights simultaneously and needs information about every connection in an ANN. Two equal size matrices and one vector are used to specify an ANN in EPNet. The dimension of the vector is determined by a user-specified upper limit N , which is the maximum number of hidden nodes allowable in the ANN. The size of the two matrices is $(m + N + n) \times (m + N + n)$, where m and n are the number of input and output nodes, respectively. One matrix is the connectivity matrix of the ANN, whose entries can only be zero or one. The other is the corresponding weight matrix whose entries are real numbers. Using two matrices rather than one is purely implementation-driven. The entries in the hidden node vector can be either one, i.e., the node exists, or zero, i.e., the node does not exist.

Since this paper is only concerned with feedforward ANN's, only the upper triangle will be considered in the two matrices. There will be no connections among input nodes. Architectural mutations can be implemented easily under such a representation scheme. Node deletion and addition involve flipping a bit in the hidden node vector. A zero bit disables all the connections to and from the node in the connectivity matrix. Connection deletion and addition involve flipping a bit in the connectivity matrix. A zero bit automatically disables the corresponding weight entry in the weight matrix. The weights are updated by a hybrid algorithm described later.

B. Fitness Evaluation and Selection Mechanism

The fitness of each individual in EPNet is solely determined by the inverse of an error value defined by (1) [36] over a validation set containing T patterns

$$E = 100 \cdot \frac{o_{\max} - o_{\min}}{T \cdot n} \sum_{t=1}^T \sum_{i=1}^n (Y_i(t) - Z_i(t))^2 \quad (1)$$

where o_{\max} and o_{\min} are the maximum and minimum values of output coefficients in the problem representation, n is the number of output nodes, $Y_i(t)$ and $Z_i(t)$ are actual and desired outputs of node i for pattern t .

Equation (1) was suggested by Prechelt [36] to make the error measure less dependent on the size of the validation set and the number of output nodes. Hence a mean squared error percentage was adopted. o_{\max} and o_{\min} were the maximum and minimum values of outputs [36].

The fitness evaluation in EPNet is different from previous work in EANN's since it is determined through a validation set which does not overlap with the training set. Such use of a validation set in an evolutionary learning system improves the generalization ability of evolved ANN's and introduces little overhead in computation time.

The selection mechanism used in EPNet is rank based. Let M sorted individuals be numbered as $0, 1, \dots, M - 1$, with

the zeroth being the fittest. Then the $(M - j)$ th individual is selected with probability [37]

$$p(M - j) = \frac{j}{\sum_{k=1}^M k}.$$

The selected individual is then modified by the five mutations. In EPNet, error E is used to sort individuals directly rather than to compute $f = 1/E$ and use f to sort them.

C. Replacement Strategy and Generation Gap

The replacement strategy used in EPNet reflects the emphasis on evolving ANN behaviors and maintaining behavioral links between parents and their offspring. It also reflects that EPNet actually emulates a kind of Lamarckian rather than Darwinian evolution. There is an on-going debate on whether Lamarckian evolution or Baldwin effect is more efficient in simulated evolution [38], [39]. Ackley and Littman [38] have presented a case for Lamarckian evolution. The experimental results of EPNet seem to support their view.

In EPNet, if an offspring is obtained through further BP partial training, it always replaces its parent. If an offspring is obtained through SA training, it replaces its parent only when it reduces its error significantly. If an offspring is obtained through deleting nodes/connections, it replaces the worst individual in the population only when it is better than the worst. If an offspring is obtained through adding nodes/connections, it always replaces the worst individual in the population since an ANN with more nodes/connections is more powerful although its current performance may not be very good due to incomplete training.

The generation gap in EPNet is minimal. That is, a new generation starts immediately after the above replacement. This is very similar to the steady-state GA [40], [41] and continuous EP [42], although the replacement strategy used in EPNet is different. It has been shown that the steady-state GA and continuous EP outperform their classical counterparts in terms of speed and the quality of solutions [40]–[42].

The replacement strategy and generation gap used in EPNet also facilitate population-based incremental learning. Vavak and Forgarty [43] have recently shown that the steady-state GA outperformed the generational GA in tracking “environmental changes which are relatively small and occur with low frequency.”

D. Hybrid Training

The only mutation for modifying ANN's weights in EPNet is implemented by a hybrid training algorithm consisting of an MBP and an SA algorithm. It could be regarded as two mutations driven by the BP and SA algorithm separately. They are treated as one in this paper for convenience sake.

The classical BP algorithm [44] is notorious for its slow convergence and convergence to local minima. Hence it is modified in order to alleviate these two problems. A simple heuristic is used to adjust the learning rate for each ANN in the population. Different ANN's may have different learning rates. During BP training, the error E is checked after every

k epochs, where k is a parameter determined by the user. If E decreases, the learning rate is increased by a predefined amount. Otherwise, the learning rate is reduced. In the latter case the new weights and error are discarded.

In order to deal with the local optimum problem suffered by the classical BP algorithm, an extra training stage is introduced when BP training cannot improve an ANN anymore. The extra training is performed by an SA algorithm. When the SA algorithm also fails to improve the ANN, the four mutations will be used to change the ANN architecture. It is important in EPNet to train an ANN first without modifying its architecture. This reflects the emphasis on a close behavioral link between the parent and its offspring.

The hybrid training algorithm used in EPNet is not a critical choice in the whole system. Its main purpose is to discourage architectural mutations if training, which often introduces smaller behavioral changes in comparison with architectural mutations, can produce a satisfactory ANN. Other training algorithms which are faster and can avoid poor local minima can also be used in EPNet. For example, recently proposed new algorithms, such as guided evolutionary simulated annealing [45], NOVEL [46] and fast evolutionary programming [47], can all be used in EPNet. The investigation of the best training algorithm is outside the scope of this paper and would be the topic of a separate paper.

E. Architecture Mutations

In EPNet, only when the hybrid training fails to reduce the error of an ANN will architectural mutations take place. For architectural mutations, node or connection deletions are always attempted before connection or node additions in order to encourage the evolution of small ANN's. Connection or node additions will be tried only after node or connection deletions fail to produce a good offspring. Using the order of mutations to encourage parsimony of evolved ANN's represents a dramatically different approach from using a complexity (regularization) term in the fitness function. It avoids the time-consuming trial-and-error process of selecting a suitable coefficient for the regularization term.

Hidden Node Deletion: Certain hidden nodes are first deleted uniformly at random from a parent ANN. The maximum number of hidden nodes that can be deleted is set by a user-specified parameter. Then the mutated ANN is partially trained by the MBP. This extra training process can reduce the sudden behavioral change caused by the node deletion. If this trained ANN is better than the worst ANN in the population, the worst ANN will be replaced by the trained one and no further mutation will take place. Otherwise connection deletion will be attempted.

Connection Deletion: Certain connections are selected probabilistically for deletion according to their importance. The maximum number of connections that can be deleted is set by a user-specified parameter. The importance is defined by a significance test for the weight's deviation from zero in the weight update process [48]. Denote the weight update $\Delta w_{ij}(w) = -\eta[\partial L_t / \partial w_{ij}]$ by the local gradient of the linear error function L ($L = \sum_{t=1}^T \sum_{i=1}^n |Y_i(t) - Z_i(t)|$) with

respect to example t and weight w_{ij} , the significance of the deviation of w_{ij} from zero is defined by the test variable [48]

$$\text{test}(w_{ij}) = \frac{\sum_{t=1}^T \xi_{ij}^t}{\sqrt{\sum_{t=1}^T (\xi_{ij}^t - \bar{\xi}_{ij})^2}} \quad (2)$$

where $\xi_{ij}^t = w_{ij} + \Delta w_{ij}^t(w)$, $\bar{\xi}_{ij}$ denotes the average over the set ξ_{ij}^t , $t = 1, \dots, T$. A large value of test variable $\text{test}(w_{ij})$ indicates higher importance of the connection with weight w_{ij} .

The advantage of the above nonconvergent method [48] over others is that it does not require the training process to converge in order to test connections. It does not require any extra parameters either. For example, Odri *et al.*'s method needs to "guess" values for four additional parameters. The idea behind the test variable (2) is to test the significance of the deviation of w_{ij} from zero [48]. Equation (2) can also be used for connections whose weights are zero, and thus can be used to determine which connections should be added in the addition phase.

Similar to the case of node deletion, the ANN will be partially trained by the MBP after certain connections have been deleted from it. If the trained ANN is better than the the worst ANN in the population, the worst ANN will be replaced by the trained one and no further mutation will take place. Otherwise node/connection addition will be attempted.

Connection and Node Addition: As mentioned before, certain connections are added to a parent network probabilistically according to (2). They are selected from those connections with zero weights. The added connections are initialized with small random weights. The new ANN will be partially trained by the MBP and denoted as Offspring 1.

Node addition is implemented through splitting an existing hidden node, a process called "cell division" by Odri *et al.* [30]. In addition to reasons given by Odri *et al.* [30], growing an ANN by splitting existing ones can preserve the behavioral link between the parent and its offspring better than by adding random nodes. The nodes for splitting are selected uniformly at random among all hidden nodes. Two nodes obtained by splitting an existing node i have the same connections as the existing node. The weights of these new nodes have the following values [30]:

$$\begin{aligned} w_{ij}^1 &= w_{ij}^2 = w_{ij}, & i \geq j \\ w_{ki}^1 &= (1 + \alpha)w_{ki}, & i < k \\ w_{ki}^2 &= -\alpha w_{ki}, & i < k \end{aligned}$$

where \mathbf{w} is the weight vector of the existing node i , \mathbf{w}^1 and \mathbf{w}^2 are the weight vectors of the new nodes, and α is a mutation parameter which may take either a fixed or random value. The split weights imply that the offspring maintains a strong behavioral link with the parent. For training examples which were learned correctly by the parent, the offspring needs little adjustment of its inherited weights during partial training.

The new ANN produced by node splitting is denoted as Offspring 2. After it is generated, it will also be partially trained by the MBP. Then it has to compete with Offspring 1 for survival. The survived one will replace the worst ANN in the population.

TABLE I
THE PARAMETERS USED IN THE EXPERIMENTS WITH THE N PARITY PROBLEM

Population size	20	# epochs for each MBP partial training	100
Initial number of hidden nodes	$2-N$	# mutated hidden nodes	1-2
Initial connection density	0.75	# mutated connections	1-3
Initial learning rate	0.5	# temperatures in SA	5
Range of learning rate	0.1-0.6	# iterations at each temperature	100

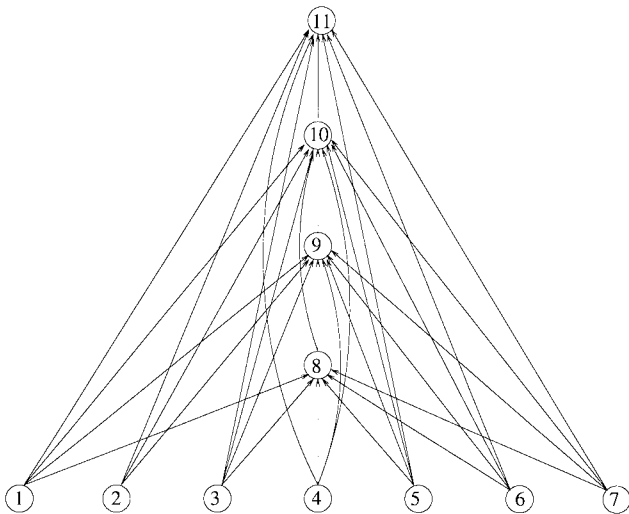


Fig. 7. The best network evolved by EPNet for the seven-parity problem.

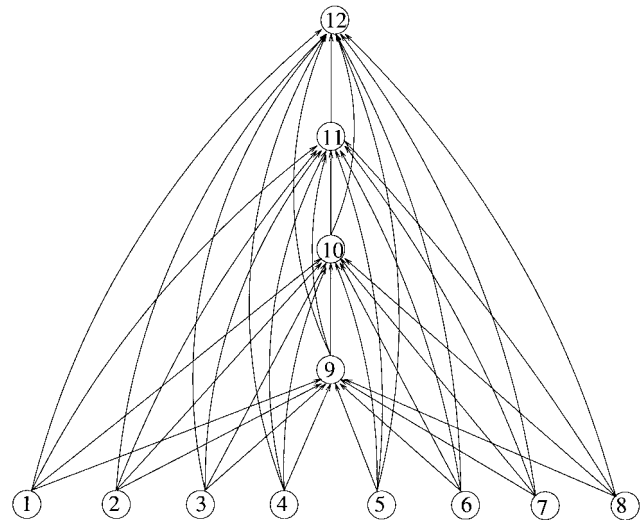


Fig. 8. The best network evolved by EPNet for the eight-parity problem.

F. Further Training After Evolution

One of the most important goal for ANN's is to have a good generalization ability. In EPNet, a training set is used for the MBP and a validation set for fitness evaluation in the evolutionary process. After the simulated evolution, the best evolved ANN is further trained using the MBP on the combined training and validation set. Then this further trained ANN is tested on an unseen testing set to evaluate its performance.

Alternatively, all the ANN's in the final population can be trained using the MBP and the one which has the best performance on a second validation set is selected as EPNet's final output. This method is more time-consuming, but it considers all the information in the final population rather than just the best individual. The importance of making use of the information in a population has recently been demonstrated by evolving both ANN's [49], [50] and rule-based systems [50], [51]. The use of a second validation set also helps to prevent ANN's from overfitting the combined training and the first validation set. Experiments using either one or two validation sets will be described in the following section.

IV. EXPERIMENTAL STUDIES

A. The Parity Problems

EPNet was first tested on the N parity problem where $N = 4-8$ [34]. All 2^N patterns were used in training. No validation

sets were used. The parameters used in the experiments are given in Table I.

Ten runs were conducted for each N value from four to eight for the N parity problem. The results are summarized in Table II, where "number of epochs" indicates the total number of epochs taken by EPNet when the best network is obtained.

The results obtained by EPNet are quite competitive in comparison with those obtained by other algorithms. Table III compares EPNet's best results with those of cascade-correlation algorithm (CCA) [5], the perceptron cascade algorithm (PCA) [7], the tower algorithm (TA) [6], and the FNNCA [8]. All these algorithms except for the FNNCA can produce networks with short cut connections. Two observations can be made from this table. First, EPNet can evolve very compact networks. In fact, it generated the smallest ANN among the five algorithms compared here. Second, the size of the network evolved by EPNet seems to grow slower than that produced by other algorithms when the size of the problem (i.e., N) increases. That is, EPNet seems to perform even better for large problems in terms of the number of hidden nodes. Since CCA, PCA, and TA are all fully connected, the number of connections in EPNet-evolved ANN's is smaller as well.

Figs. 7 and 8 show the best networks evolved by EPNet for the seven- and eight-parity problem, respectively. Tables IV and V give their weights. It is rather surprising that a three-hidden-node network can be found by EPNet for the eight-

TABLE II
SUMMARY OF THE RESULTS PRODUCED BY EPNet ON THE N PARITY PROBLEM. ALL RESULTS WERE AVERAGED OVER TEN RUNS

Problem Instance		Parity-4	Parity-5	Parity-6	Parity-7	Parity-8
Number of hidden nodes	Min	2	2	3	3	3
	Max	3	4	4	5	6
	Mean	2.3	2.5	3.2	3.4	4.6
	SD	0.483	0.707	0.422	0.699	1.188
Number of connections	Min	13	17	28	31	38
	Max	16	27	37	48	76
	Mean	14.5	20.6	30.3	34.7	55.0
	SD	1.080	3.718	3.368	5.122	14.262
Number of epochs	Min	4950	17100	62050	103850	97150
	Max	27250	38200	312150	283700	401850
	Mean	14625	30245	132525	177417	249625
	SD	7332	7416	82728	72834	127562
Error of networks	Min	8.3×10^{-6}	1.1×10^{-2}	1.5×10^{-3}	4.2×10^{-4}	3.9×10^{-4}
	Max	1.4×10^{-3}	5.0×10^{-2}	6.1×10^{-2}	3.2×10^{-2}	2.1×10^{-2}
	Mean	5.0×10^{-4}	1.4×10^{-2}	1.2×10^{-2}	8.9×10^{-3}	5.2×10^{-3}
	SD	3.5×10^{-3}	1.6×10^{-2}	1.8×10^{-2}	9.5×10^{-3}	7.1×10^{-3}

TABLE III
COMPARISON BETWEEN EPNet AND OTHER ALGORITHMS IN TERMS OF THE MINIMAL NUMBER OF HIDDEN NODES IN THE BEST NETWORK GENERATED. THE FIVE-TUPLES IN THE TABLE REPRESENT THE NUMBER OF HIDDEN NODES FOR THE FOUR-, FIVE-, SIX-, SEVEN-, AND EIGHT-PARITY PROBLEM, RESPECTIVELY. "-" MEANS NO RESULT IS AVAILABLE

Algorithm	EPNet	CCA	PCA	TA	FNNCA
Number of hidden nodes	2,2,3,3,3	2,2,3,4,5	2,2,3,3,4	2,2,3,3,4	3,4,5,5,-

parity problem. This demonstrates an important point made by many evolutionary algorithm researchers—an evolutionary algorithm can often discover novel solutions which are very difficult to find by human beings. However, EPNet might take a long time to find a solution to a large parity problem. Some of the runs did not finish within the user-specified maximum number of generations.

Although there is a report on a two-hidden-node ANN which can solve the N parity problem [52], their network was hand-crafted and used a very special node transfer function, rather than the usual sigmoid one.

B. The Medical Diagnosis Problems

Since the training set was the same as the testing set in the experiments with the N parity problem, EPNet was only tested for its ability to evolve ANN's that learn well but not necessarily generalize well. In order to evaluate EPNet's ability in evolving ANN's that generalize well, EPNet was applied to four real-world problems in the medical domain, i.e., the breast cancer problem, the diabetes problem, the heart disease problem, and the thyroid problem. All data sets were obtained from the UCI machine learning benchmark repos-

itory. These medical diagnosis problems have the following common characteristics [36].

- The input attributes used are similar to those a human expert would use in order to solve the same problem.
- The outputs represent either the classification of a number of understandable classes or the prediction of a set of understandable quantities.
- In practice, all these problems are solved by human experts.
- Examples are expensive to get. This has the consequence that the training sets are not very large.
- There are missing attribute values in the data sets.

These data sets represent some of the most challenging problems in the ANN and machine learning field. They have a small sample size of noisy data.

The Breast Cancer Data Set: The breast cancer data set was originally obtained from W. H. Wolberg at the University of Wisconsin Hospitals, Madison. The purpose of the data set is to classify a tumour as either benign or malignant based on cell descriptions gathered by microscopic examination. The data set contains nine attributes and 699 examples of which 458 are benign examples and 241 are malignant examples.

TABLE IV
CONNECTION WEIGHTS AND BIASES (REPRESENTED
BY T) FOR THE NETWORK IN FIG. 7

T	1	2	3	4	5	
8	-42.8	-32.4	0	-5.6	-23.6	-33.6
9	-75.3	-32.0	43.2	-41.1	-34.5	-34.8
10	-85.0	-28.1	28.6	-28.0	-28.0	-28.2
11	59.9	12.9	-13.5	13.0	13.0	13.0

	6	7	8	9	10
8	-33.6	41.6	0	0	0
9	-34.8	39.8	-58.9	0	0
10	-28.2	29.3	-47.6	-41.3	0
11	13.0	-13.4	0	0	81.8

The Diabetes Data Set: This data set was originally donated by Vincent Sigillito from Johns Hopkins University and was constructed by constrained selection from a larger database held by the National Institute of Diabetes and Digestive and Kidney Diseases. All patients represented in this data set are females of at least 21 years old and of Pima Indian heritage living near Phoenix, AZ.

The problem posed here is to predict whether a patient would test positive for diabetes according to World Health Organization criteria given a number of physiological measurements and medical test results.

This is a two class problem with class value one interpreted as "tested positive for diabetes." There are 500 examples of class 1 and 268 of class 2. There are eight attributes for each example. The data set is rather difficult to classify. The so-called "class" value is really a binarised form of another attribute which is itself highly indicative of certain types of diabetes but does not have a one to one correspondence with the medical condition of being diabetic.

The Heart Disease Data Set: This data set comes from the Cleveland Clinic Foundation and was supplied by Robert Detrano of the V.A. Medical Center, Long Beach, CA. The purpose of the data set is to predict the presence or absence of heart disease given the results of various medical tests carried out on a patient. This database contains 13 attributes, which have been extracted from a larger set of 75. The database originally contained 303 examples but six of these contained missing class values and so were discarded leaving 297. Twenty seven of these were retained in case of dispute, leaving a final total of 270. There are two classes: presence and absence (of heart disease). This is a reduction of the number of classes in the original data set in which there were four different degrees of heart disease.

The Thyroid Data Set: This data set comes from the "ann" version of the "thyroid disease" data set from the UCI machine learning repository. Two files were provided. "ann-train.data" contains 3772 learning examples. "ann-test.data" contains 3428 testing examples. There are 21 attributes for each example.

TABLE V
CONNECTION WEIGHTS AND BIASES (REPRESENTED
BY T) FOR THE NETWORK IN FIG. 8

T	1	2	3	4	5	
9	-12.4	25.2	27.7	-29.4	-28.9	-29.7
10	-40.4	19.6	18.9	-18.1	-19.1	-18.5
11	-48.1	16.0	16.1	-15.9	-16.3	-15.8
12	45.7	-10.0	-11.0	10.0	9.9	9.4

	6	7	8	9	10	11
9	-25.4	-28.5	27.8	0	0	0
10	-17.3	-18.8	20.4	-67.6	0	0
11	-15.9	-15.8	16.7	-55.0	-26.7	0
12	10.0	9.6	-11.4	6.8	2.3	76.3

The purpose of the data set is to determine whether a patient referred to the clinic is hypothyroid. Therefore three classes are built: normal (not hypothyroid), hyperfunction and subnormal functioning. Because 92 percent of the patients are not hyperthyroid, a good classifier must be significantly better than 92%.

Experimental Setup: All the data sets used by EPNet were partitioned into three sets: a training set, a validation set, and a testing set. The training set was used to train ANN's by MBP, the validation set was used to evaluate the fitness of the ANN's. The best ANN evolved by EPNet was further trained on the combined training and validation set before it was applied to the testing set.

As indicated by Prechelt [36], [53], it is insufficient to indicate only the number of examples for each set in the above partition, because the experimental results may vary significantly for different partitions even when the numbers in each set are the same. An imprecise specification of the partition of a known data set into the three sets is one of the most frequent obstacles to reproduce and compare published neural-network learning results. In the following experiments, each data set was partitioned as follows.

- For the breast cancer data set, the first 349 examples were used for the training set, the following 175 examples for the validation set, and the final 175 examples for the testing set.
- For the diabetes data set, the first 384 examples were used for the training set, the following 192 examples for the validation set, the final 192 examples for the testing set.
- For the heart disease data set, the first 134 examples were used for the training set, the following 68 examples for the validation set, and the final 68 examples for the testing set.
- For the thyroid data set, the first 2514 examples in "ann-train.data" were used for the training set, the rest in "ann-train.data" for the validation set, and the whole "ann-test.data" for the testing set.

The input attributes of the diabetes data set and heart disease data set were rescaled to between 0.0 and 1.0 by

a linear function. The output attributes of all the problems were encoded using a 1-of- m output representation for m classes. The winner-takes-all method was used in EPNet, i.e., the output with the highest activation designates the class.

There are some control parameters in EPNet which need to be specified by the user. It is, however, unnecessary to tune all these parameters for each problem because EPNet is not very sensitive to them. Most parameters used in the experiments were set to be the same: the population size (20), the initial connection density (1.0), the initial learning rate (0.25), the range of learning rate (0.1 to 0.75), the number of epochs for the learning rate adaptation (5), the number of mutated hidden nodes (1), the number of mutated connections (one to three), the number of temperatures in SA (5), and the number of iterations at each temperature (100). The different parameters were the number of hidden nodes of each individual in the initial population and the number of epochs for MBP's partial training. The number of hidden nodes for each individual in the initial population was chosen from a uniform distribution within certain ranges: one to three hidden nodes for the breast cancer problem; two to eight for the diabetes problem; three to five for the heart disease problem; and six to 15 for the thyroid problem.

The number of epochs (K_0) for training each individual in the initial population is determined by two user-specified parameters: the "stage" size and the number of stages. A stage includes a certain number of epochs for MBP's training. The two parameters mean that an ANN is first trained for one stage. If the error of the network reduces, then another stage is executed, or else the training finishes. This step can repeat up to *the-number-of-stages* times. This simple method balances fairly well between the training time and the accuracy. For the breast cancer problem and the diabetes problem, the two parameters were 400 and two. For the heart disease problem, they were 500 and two. For the thyroid problem, they were 350 and three.

The number of epochs for each partial training during evolution (i.e., K_1) was determined in the same way as the above. The two parameters were 50 and three for the thyroid problem, 100 and two for the other problems. The number of epochs for training the best individual on the combined training and testing data set was set to be the same (1000) for all four problems. A run of EPNet was terminated if the average error of the population had not decreased by more than a threshold value ϵ after consecutive G_0 generations or a maximum number of generations was reached. The same maximum number of generations (500) and the same G_0 (10) were used for all four problems. The threshold value ϵ was set to 0.1 for the thyroid problem, and 0.01 for the other three. These parameters were chosen after some limited preliminary experiments. They were not meant to be optimal.

Experimental Results: Tables VI and VII show EPNet's results over 30 runs. The error in the tables refers to the error defined by (1). The error rate refers to the percentage of wrong classifications produced by the evolved ANN's.

It is clear from the two tables that the evolved ANN's have very small sizes, i.e., a small number of hidden nodes and connections, as well as low error rates. For example,

TABLE VI
ARCHITECTURES OF EVOLVED ARTIFICIAL NEURAL NETWORKS

		Number of connections	Number of hidden nodes	Number of generations
Breast	Mean	41.0	2.0	137.3
Cancer	SD	14.7	1.1	37.7
Data	Min	15	0	100
Set	Max	84	5	240
Diabetes	Mean	52.3	3.4	132.2
Data	SD	16.1	1.3	48.7
Set	Min	27	1	100
	Max	87	6	280
Heart	Mean	92.6	4.1	193.3
Disease	SD	40.8	2.1	60.3
Data	Min	34	1	120
Set	Max	213	10	320
Thyroid	Mean	219.6	5.9	45.0
Data	SD	74.36	2.4	12.5
Set	Min	128	3	10
	Max	417	12	70

an evolved ANN with just one hidden node can achieve an error rate of 19.794% on the testing set for the diabetes problem. Another evolved ANN with just three hidden nodes can achieve an error rate of 1.925% on the testing set for the thyroid problem.

In order to observe the evolutionary process in EPNet, Figs. 9–12 show the evolution of the mean of average numbers of connections and the mean of average classification accuracy of ANN's over 30 runs for the four medical diagnosis problems. The evolutionary processes are quite interesting. The number of connections in ANN's decreases in the beginning of the evolution. After certain number of generations, the number starts increasing in some cases, e.g., Fig. 9. This phenomenon illustrates the effectiveness of the ordering of different mutations in EPNet. There is an obvious bias toward parsimonious ANN's.

In the beginning stage of the evolution, very few ANN's will be fully trained and thus most of them will have high errors. Deleting a few connections from an ANN will not affect its high error very much. After each deletion, further training is always performed, which is likely to reduce the high error. Hence deletion will be successful and the number of connections will be reduced. After certain number of generations, ANN's in the population will have fewer connections and lower errors than before. They have reached such a level that further deletion of connections will increase their errors in spite of further training due to the insufficient capacity of the ANN. Hence deletion is likely to fail and addition is likely to be attempted. Since further training after adding

TABLE VII
ACCURACIES OF EVOLVED ARTIFICIAL NEURAL NETWORKS

		Training set		Validation set		Test set	
		error	error rate	error	error rate	error	error rate
Breast	Mean	3.246	0.03773	0.644	0.00590	1.421	0.01376
Cancer	SD	0.589	0.00694	0.213	0.00236	0.729	0.00938
Data	Min	1.544	0.01719	0.056	0.00000	0.192	0.00000
Set	Max	3.890	0.04585	1.058	0.01143	3.608	0.04000
Diabetes	Mean	16.674	0.24054	13.308	0.18854	15.330	0.22379
Data	SD	0.294	0.00009	0.437	0.00008	0.300	0.00014
Set	Min	16.092	0.21875	12.574	0.16667	14.704	0.19271
	Max	17.160	0.26042	14.151	0.20313	15.837	0.25000
Heart	Mean	10.708	0.13632	13.348	0.17304	12.270	0.16765
Disease	SD	0.748	0.01517	0.595	0.01995	0.724	0.02029
Data	Min	8.848	0.09702	12.388	0.13235	10.795	0.13235
Set	Max	12.344	0.16418	14.540	0.20588	14.139	0.19118
Thyroid	Mean	0.470	0.00823	0.689	0.01174	1.157	0.02115
Data	SD	0.091	0.00146	0.127	0.00235	0.098	0.00220
Set	Min	0.336	0.00517	0.469	0.00636	0.887	0.01634
	Max	0.706	0.01154	1.066	0.01749	1.328	0.02625

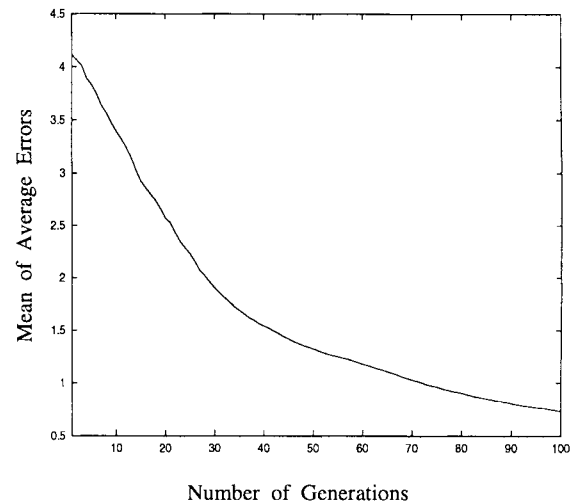
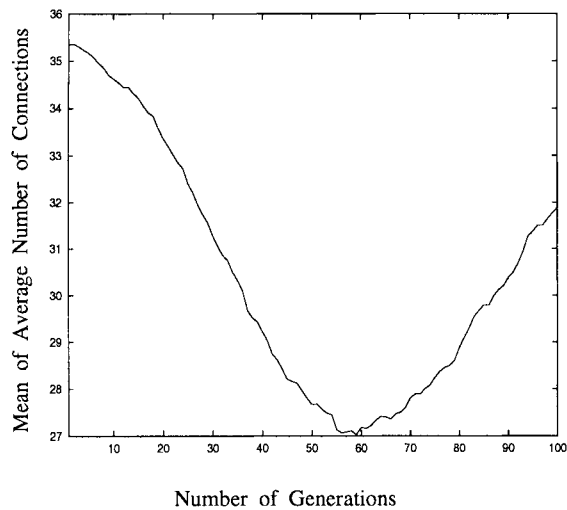


Fig. 9. Evolution of ANN's connections and accuracy for the breast cancer problem.

extra connections to an ANN often reduces its error because of a more powerful ANN, addition is likely to succeed. Hence the number of connections increases gradually while the error keeps reducing. Such trend is not very clear in Figs. 11 and 12, but it is expected to appear if more generations were allowed for the experiments. The heart disease and thyroid problems are larger than the breast cancer and diabetes problems. They would need more time to reach the lowest point for the number of connections.

Comparisons with Other Work: Direct comparison with other evolutionary approaches to designing ANN's is very difficult due to the lack of such results. Instead, the best and latest results available in the literature, regardless of whether the algorithm used was an evolutionary, a BP or a statistical one, were used in the comparison. It is possible that some papers which should have been compared with were overlooked. However, the aim of this paper is not to compare EPNet exhaustively with all other algorithms.

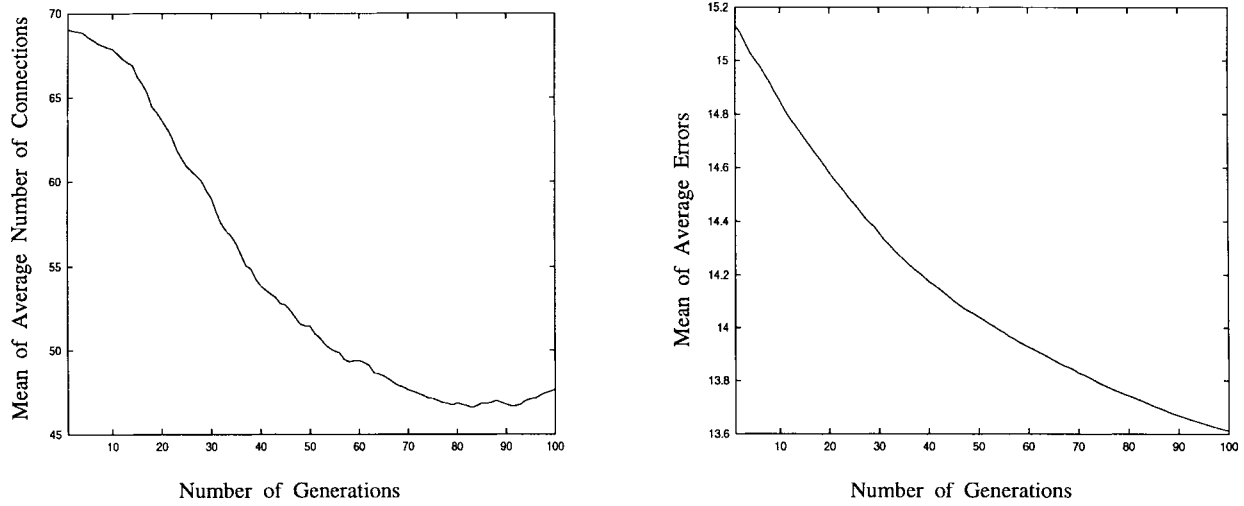


Fig. 10. Evolution of ANN's connections and accuracy for the diabetes problem.

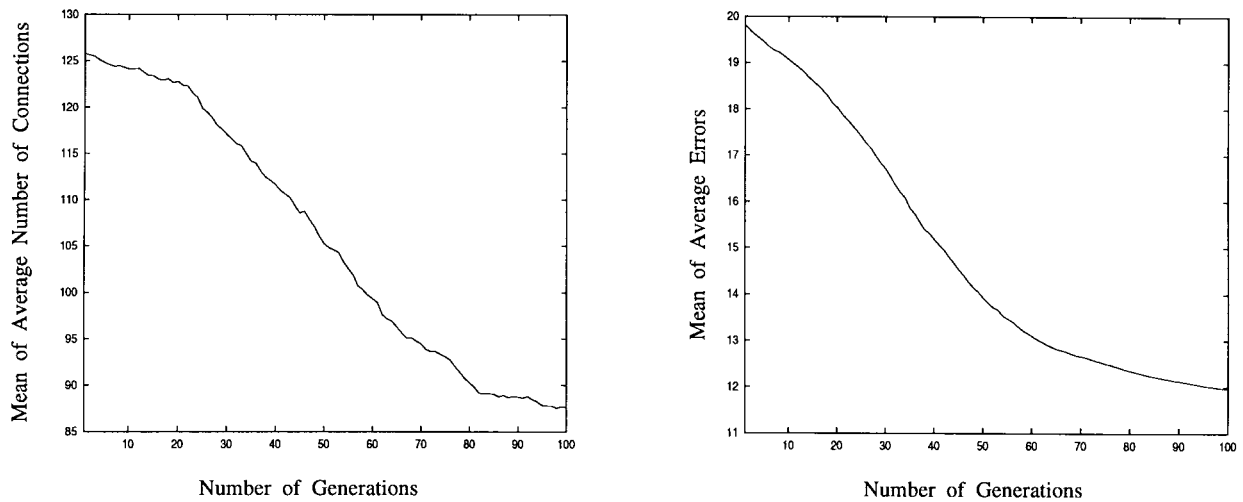


Fig. 11. Evolution of ANN's connections and accuracy for the heart disease problem.

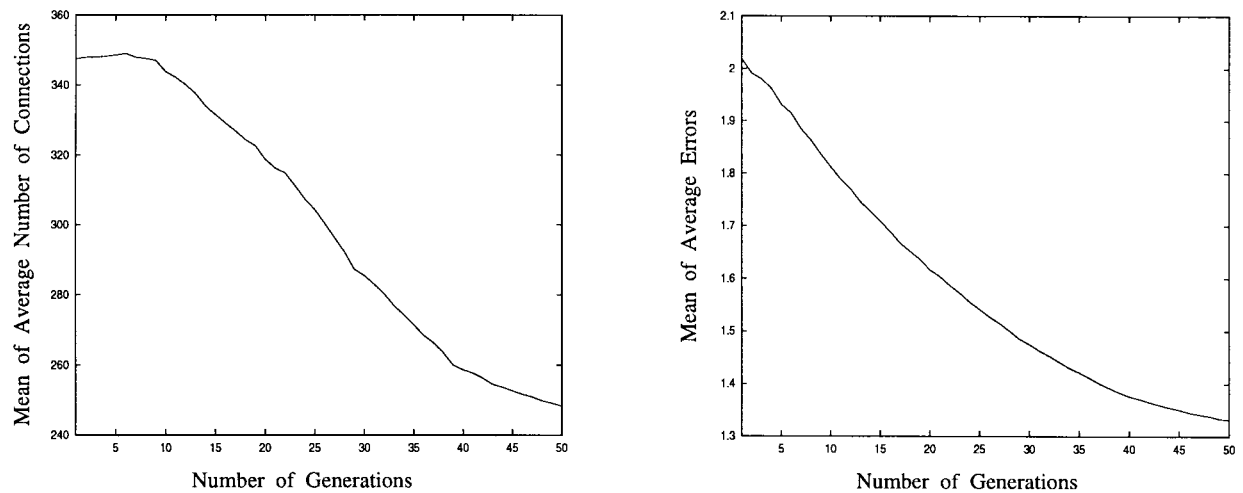


Fig. 12. Evolution of ANN's connections and accuracy for the thyroid problem. All 30 runs took less than 100 generations to finish. Some of them took less than 50 generations to finish. In those cases, the average number of connections and accuracy between the last generation and the 50th one were set to be the same as those at the last generation in order to draw this figure.

TABLE VIII

COMPARISON AMONG FNNCA [8], A HAND-DESIGNED ANN [36], AND EPNet ON THE BREAST CANCER PROBLEM. ANN'S DESIGNED MANUALLY AND BY FNNCA HAVE MORE CONNECTIONS THAN THOSE EVOLVED BY EPNet, EVEN WHEN THE NUMBER OF HIDDEN NODES IS THE SAME SINCE EPNet CAN GENERATE SPARSELY CONNECTED ANN'S. ONLY THE AVERAGE RESULTS FROM EPNet ARE SHOWN HERE. EPNet'S BEST RESULTS ARE CLEARLY SUPERIOR, AS INDICATED BY TABLE VII

	Best Results by FNNCA in 50 Runs	Best Results by HDANNS by Trial-and-Error	Average Results by EPNet Over 30 Runs
# Hidden Nodes	2	6	2.0
Testing Error Rate	0.0145 [†]	0.01149	0.01376

TABLE IX

COMPARISON BETWEEN EPNet AND OTHERS [54] IN TERMS OF THE AVERAGE TESTING ERROR RATE ON THE DIABETES PROBLEM

Algorithm	EPNet	Logdisc	DIPOL92	Discrim	SMART	RBF
Testing Error Rate	0.224	0.223	0.224	0.225	0.232	0.243
Algorithm	ITrule	BP	Cal5	CART	CASTLE	Quadisc
Testing Error Rate	0.245	0.248	0.250	0.255	0.258	0.262

TABLE X

COMPARISON AMONG MSM1 [56], A HAND-DESIGNED ANN [36], AND EPNet ON THE HEART DISEASE PROBLEM. THE SMALLEST ERROR RATE ACHIEVED BY EPNet WAS 0.13235. "-" IN THE TABLE MEANS "NOT AVAILABLE"

	Results by MSM1	Best Results by HDANNS by Trial-and-Error	Average Results by EPNet Over 30 Runs
# Hidden Nodes	-	4	4.1
Testing Error Rate	0.1653	0.1478	0.16767

The breast cancer problem: Setiono and Hui [8] have recently published a new ANN constructive algorithm called FNNCA. Prechelt [36] also reported results on manually constructed ANN's. He tested a number of different ANN architectures for the breast cancer problem. The best results produced by FNNCA [8] and by hand-designed ANN's (denoted as HDANN's) [36] are compared to the average results produced by EPNet in Table VIII.

Although EPNet can evolve very compact ANN's which generalize well, they come with the cost of additional computation time in order to perform search. The total time used by EPNet could be estimated by adding the initial training time ($400 \times 20 = 8000$ epochs), the evolving time (approximately 200 epochs per generation for maximally 500 generations), and the final training time (1000 epochs) together. That is, it could require roughly 109 000 epochs for a single run. The actual time was less since few runs reached the maximal number of generations. Similar estimations can be applied to other problems tested in this paper. For many applications, the training time is less important than generalization. Section I has explained why the evolutionary approach is necessary and better than constructive algorithms for such applications.

The diabetes problem: The diabetes problem is one of the most challenging problems in ANN and machine learning due to its relatively small data set and high noise level. In the medical domain, data are often very costly to obtain. It would

be unreasonable if an algorithm relies on more training data to improve its generalization.

Table IX compares EPNet's result with those produced by a number of other algorithms [54]. It is worth pointing out that the other results were obtained by 12-fold cross validation [54]. They represented the best 11 out of 23 algorithms tested [54].

In terms of best results produced, Prechelt [36] tried different ANN's manually for the problem and found an eight-hidden-node ANN which achieved the testing error rate of 0.2135 (21.35%), while EPNet achieved the testing error rate of 0.1927 (19.27%). The largest ANN evolved by EPNet among 30 runs had only six hidden nodes. The average was 3.4.

The heart disease problem: Table X shows results from EPNet and other neural and nonneural algorithms. The GM algorithm [55] is used to construct RBF networks. It produced a RBF network of 24 Gaussians with 18.18% testing error. Bennet and Mangasarian [56] reported a testing error rate of 16.53% with their MSM1 method, 25.92% with their MSM method, and about 25% with BP, which is much worse than the worst ANN evolved by EPNet. The best manually designed ANN achieved 14.78% testing error [36], which is worse than the best result of EPNet, 13.235%.

The thyroid problem: Schiffmann *et al.* [57] tried this problem using a 21-20-3 network. They found that several thousand learning passes were necessary to achieve a testing

TABLE XI
COMPARISON AMONG SCHIFFMANN *ET AL.*'S BEST RESULTS [56], A HAND-DESIGNED ANN [36], AND EPNet ON THE THYROID PROBLEM. THE SMALLEST ERROR RATE ACHIEVED BY EPNet WAS 0.01634

	Results by Schiffmann	Best Results by HDANNS by Trial-and-Error	Average Results by EPNet Over 30 Runs
# Hidden Nodes	50	12	5.9
Testing Error Rate	0.025	0.01278	0.02115

TABLE XII
ARCHITECTURES OF EVOLVED ANN'S FOR THE AUSTRALIAN CREDIT CARD DATA SET

	Mean	Std Dev	Min	Max
Number of connections	88.03	24.70	43	127
Number of hidden nodes	4.83	1.62	2	7

error rate of 2.6% for this network. They also used their genetic algorithm to train multilayer ANN's on the reduced training data set containing 713 examples. They obtained a network with 50 hidden nodes and 278 connections, which had testing error rate 2.5%. These results are even worse than those generated by the worst ANN evolved by EPNet. However, the best manually designed ANN [36] has a testing error rate of 1.278%, which is better than EPNet's best result, 1.634%. This is the only case where the best manually designed ANN [36] outperforms EPNet's best. Table XI summarizes the above results.

C. The Australian Credit Card Assessment Problem

One of things which have often been overlooked in evolutionary algorithms is the information contained in the final population of the evolution. Most people just use the best individual in the population without thinking of exploring possible useful information in the rest of the population. In EPNet, simulated evolution is driven by an EP algorithm without any recombination operator. Due to the many to many mapping between genotypes and phenotypes [3], [25], different individuals in the final population may have similar error rates but quite different architectures and weights. Some of them may have overfitted the training and/or validation set, and some may not. In order to avoid overfitting and achieve better generalization, a second validation set has been proposed to stop training in the last step of EPNet.

In the following experiment, the original validation set was divided into two equal subsets; the first (V-set 1) was used in the fitness evaluation and the second (V-set 2) was used in the last step of EPNet. In the last step, all individuals in the final population were first trained by the MBP on the combined training set and V-set 1. Then the one which produced the minimum error rate on V-set 2 was chosen as the final output from EPNet and tested on the testing set. Ties were broken in favor of the network with the minimum number of connections. If there was still a tie, it was broken at random.

The effectiveness of using a second validation set in EPNet

was tested on another difficult problem—the Australian credit card assessment problem. The problem is to assess applications for credit cards based on a number of attributes. There are 690 cases in total. The output has two classes. The 14 attributes include six numeric values and eight discrete ones, the latter having from two to 14 possible values. This data set was also obtained from the UCI machine learning repository. The input attributes used for ANN's are rescaled to between 0.0 and 1.0 by a linear function.

Experimental Results and Comparisons: The whole data set was first randomly partitioned into training data (518 cases) and testing data (172 cases). The training data was then further partitioned into three subsets: (1) the training subset (346 cases); (2) validation set 1 (86 cases); and (3) validation set 2 (86 cases).

The experiments used the same parameters as those for the diabetes problem except for the maximum number of generations which was set at 100. The average results over 30 runs are summarized in Tables XII-XIII. Very good results have been achieved by EPNet. For example, an ANN with only two hidden nodes and 43 connections could achieve an error rate of 10.47% on the testing set. Table XIV compares EPNet's results with those produced by other algorithms [54]. It is worth pointing out that the other results were obtained by ten-fold cross validation [54]. They represented the best 11 out of 23 algorithms tested [54]. It is clear that EPNet performed much better than others even though they used ten-fold cross validation.

The evolution of the mean of average numbers of connections and the mean of average classification accuracy by evolved ANN's over 30 runs for the Australian credit card assessment problem is shown in Fig. 13.

D. The MacKey–Glass Chaotic Time Series Prediction Problem

This section describes EPNet's application to a time series prediction problem. The problem is different from previous ones in that its output is continuous. It is not a classification problem. This problem is used to illustrate that EPNet is applicable to a wide range of problems since it does not assume any *a priori* knowledge of the problem domain. The only part of EPNet which needs changing in order to deal with the continuous output is the fitness evaluation module.

The MacKey–Glass time series investigated here is generated by the following differential equation:

$$\dot{x}(t) = \beta x(t) + \frac{\alpha x(t - \tau)}{1 + x^{10}(t - \tau)} \quad (3)$$

TABLE XIII
ACCURACY OF EVOLVED ANN'S FOR THE AUSTRALIAN CREDIT CARD DATA SET

	Training set		Validation set 1		Validation set 2		Test set	
	error	error rate	error	error rate	error	error rate	error	error rate
Mean	9.69	0.111	6.84	0.074	9.84	0.091	10.17	0.115
SD	2.05	0.021	1.47	0.022	2.31	0.024	1.55	0.019
Min	6.86	0.081	4.01	0.035	5.83	0.047	7.73	0.081
Max	15.63	0.173	9.94	0.105	14.46	0.140	14.08	0.157

TABLE XIV
COMPARISON BETWEEN EPN_{et} AND OTHERS [54] IN TERMS OF THE AVERAGE TESTING ERROR RATE

Algorithm	EPNet	Cal5	ITrule	DIPOL92	Discrim	Logdisc
Testing Error Rate	0.115	0.131	0.137	0.141	0.141	0.141
Algorithm	CART	RBF	CASTLE	NaiveBay	IndCART	BP
Testing Error Rate	0.145	0.145	0.148	0.151	0.152	0.154

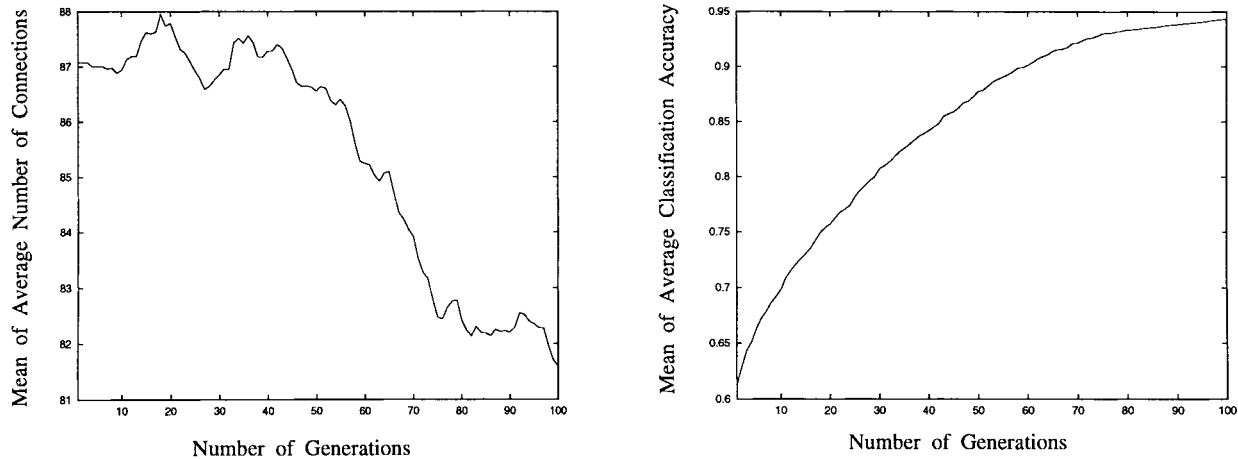


Fig. 13. Evolution of ANN's connections and accuracy for the Australian credit card assessment problem.

where $\alpha = 0.2$, $\beta = -0.1$, $\tau = 17$ [58], [59]. As mentioned by Martinetz *et al.* [60], $x(t)$ is quasiperiodic and chaotic with a fractal attractor dimension 2.1 for the above parameters.

The input to an ANN consists of four past data points, $x(t)$, $x(t-6)$, $x(t-12)$ and $x(t-18)$. The output is $x(t+6)$. In order to make multiple step prediction (i.e., $\Delta t = 90$) during testing, iterative predictions of $x(t+6)$, $x(t+12)$, \dots , $x(t+90)$ will be made. During training, the true value of $x(t+6)$ was used as the target value. Such experimental setup is the same as that used by Martinetz *et al.* [60].

In the following experiments, the data for the MacKey–Glass time series was obtained by applying the fourth-order Runge–Kutta method to (3) with initial condition $x(0) = 1.2$, $x(t-\tau) = 0$ for $0 \leq t < \tau$, and the time step is one. The training data consisted of point 118 to 617 (i.e., 500 training patterns). The following 500 data points (starting from point 618) were used as testing data. The values of training and testing data were rescaled linearly

to between 0.1 and 0.9. No validation sets were used in the experiments. Such experimental setup was adopted in order to facilitate comparison with other existing work.

The normalized root-mean-square (RMS) error E was used to evaluate the performance of EPN_{et}, which is determined by the RMS value of the absolute prediction error for $\Delta t = 6$, divided by the standard deviation of $x(t)$ [58], [60]

$$E = \frac{\langle [x_{\text{pred}}(t, \Delta t) - x(t + \Delta t)]^2 \rangle^{\frac{1}{2}}}{\langle (x - \langle x \rangle)^2 \rangle^{\frac{1}{2}}} \quad (4)$$

where $x_{\text{pred}}(t, \Delta t)$ is the prediction of $x(t + \Delta t)$ from the current state $x(t)$ and $\langle x \rangle$ represents the expectation of x . As indicated by Farmer and Sidorowich [58], “If $E = 0$, the predictions are perfect; $E = 1$ indicates that the performance is no better than a constant predictor $x_{\text{pred}}(t, \Delta t) = \langle x \rangle$ ”.

The following parameters were used in the experiments: the maximum number of generations (200), the number of hidden nodes for each individual in the initial population (eight

TABLE XV
THE AVERAGE RESULTS PRODUCED BY EPNet OVER 30 RUNS
FOR THE MACKEY–GLASS TIME-SERIES PREDICTION PROBLEM

	Mean	Std Dev	Min	Max
Number of Connections	103.33	24.63	66	149
Number of Hidden Nodes	10.87	1.78	8	14
Error on Training Set	0.0188	0.0024	0.0142	0.0237
Error on Testing Set ($\Delta t = 6$)	0.0205	0.0028	0.0152	0.0265
Error on Testing Set ($\Delta t = 90$)	0.0646	0.0103	0.0487	0.0921

to 16), the initial learning rate (0.1), the range of learning rate (0.1 to 0.75), the number of mutated hidden nodes (1), the two parameters for training each individual in the initial population (1000 and five), and the two parameters for each partial training during evolution (200 and five). All other parameters were the same as those for the medical diagnosis problems.

Experimental Results and Comparisons: Table XV shows the average results of EPNet over 30 runs. The error in the table refers to the error defined by (4). Table XVI compares EPNet's results with those produced by BP and the CC learning [61]. EPNet evolved much compact ANN's than the cascade-correlation networks, which are more than six times larger than the EPNet-evolved ANN's. EPNet-evolved ANN's also generalize better than the cascade-correlation networks. Compared with the networks produced by BP, the EPNet-evolved ANN's used only 103 connections (the median size) and achieved comparable results.

For a large time span $\Delta t = 90$, EPNet's results also compare favorably with those produced by Martinetz *et al.* [60] which had been shown to be better than Moody and Darken [62]. The average number of connections (weights) in an EPNet-evolved ANN is 103.33, while the smallest "neural-gas" network has about 200 connections (weights) [60], which is almost twice as large as the average size of EPNet-evolved ANN. To achieve a prediction error of 0.05, a "neural-gas" network had to use 1000 training data points and a size about 500 connections (weights) [60]. The smallest prediction error among 30 EPNet runs was 0.049, while the average prediction error was 0.065. For the same training set size of 500 data points, the smallest prediction error achieved by "neural-gas" networks was about 0.06. The network achieving the smallest prediction error had 1800 connections (200 hidden nodes), which is more than ten times larger than the largest EPNet-evolved ANN.

Further Discussions: In order to observe the evolutionary process of EPNet, Figs. 14 shows the evolution of the mean of average numbers of connections and hidden nodes, the mean of average normalized RMS errors, and the average numbers of five mutations used over 30 runs for the Mackey–Glass time-series prediction problem. Several observations can be made from these results.

First, EPNet is capable of finding a near optimal ANN through the evolutionary process. It can grow and prune ANN's dynamically during evolution, depending on whether the current networks are larger or smaller than necessary.

TABLE XVI
GENERALIZATION RESULTS COMPARISON AMONG EPNet, BP, AND CC
LEARNING FOR THE MACKEY–GLASS TIME-SERIES PREDICTION PROBLEM

Method	Number of Connections	Testing Error	
		$\Delta t = 6$	$\Delta t = 84$
EPNet	103	0.02	0.06
BP	540	0.02	0.05
CC Learning	693	0.06	0.32

For example, the initial number of hidden nodes for the MacKey–Glass problem was generated uniformly at random between eight and 16, which was relatively large for this problem. EPNet was able to prune ANN's significantly (see Fig. 14(b)) in the first part of the evolution because there was a bias in mutations to favor compact ANN's. The deletion mutations were successful most of the time during this period because the pruned ANN's were able to reduce the training error. When the ANN's were pruned to a certain size, further reduction in the network size would cause significant deterioration of the network performance (training error). Hence the deletion mutations were unlikely to be successful. The addition mutations would be used. That is why the number of connections and nodes increased again in the second part of the evolution in order to reduce the training error.

Second, ordering of the five mutations in EPNet has resulted in very compact ANN's due to the bias toward connection and node deletions. The effectiveness of such ordering in encouraging parsimony was illustrated by the above discussion and Fig. 14(c).

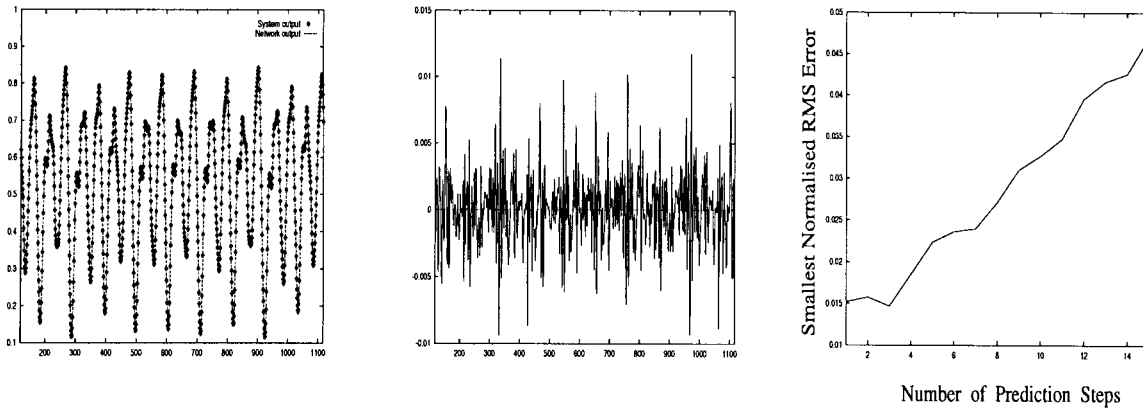
Third, connections were modified more frequently than hidden nodes because connection mutations caused less disruptions to ANN's behavior than node mutations. This is exactly what EPNet prefers to do due to its emphasis on behavioral evolution.

Fourth, EPNet was able to produce ANN's with good generalization ability. It is worth noting that all comparisons carried out in this section used EPNet's average results over 30 runs, not the best results. It is also worth mentioning that the conventional BP algorithm has a lot of difficulties in dealing with compact ANN's. It either trains slowly or cannot find a near global minimum of the error function at all. EPNet, on the other hand, can achieve very low error for a very compact ANN.

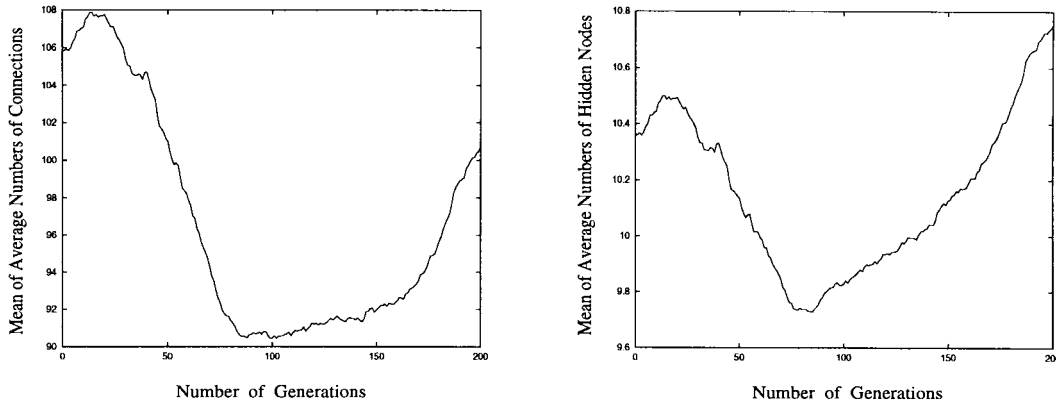
V. CONCLUSION

This paper describes a new EP based system, EPNet, for evolving feedforward ANN's. The idea behind EPNet is to put more emphasis on evolving ANN behaviors, rather than just its circuitry. EP [1], [2], [3] is better suited for evolving behaviors due to its emphasis on maintaining behavioral links between a parent and its offspring. EP also helps to avoid the permutation problem suffered by many EANN systems.

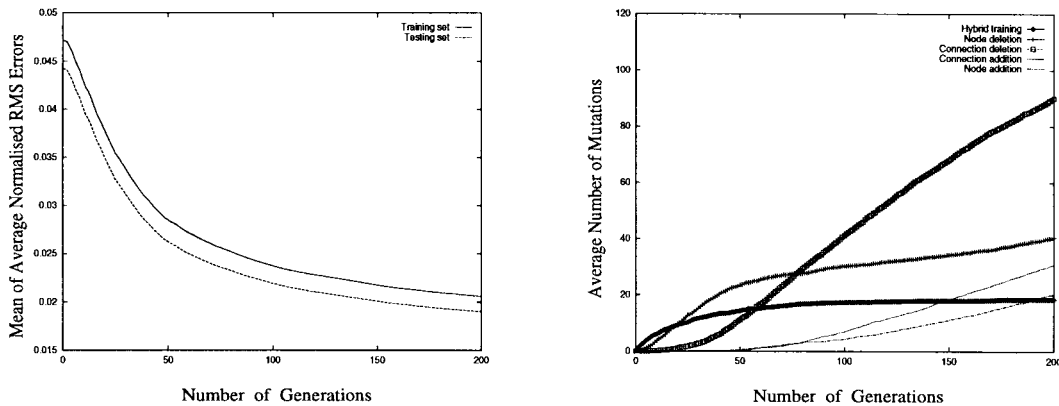
A number of techniques have been adopted in EPNet to maintain a close behavioral link between parents and their offspring. For example, partial training is always employed



(a)



(b)



(c)

Fig. 14. The Mackey–Glass time-series prediction problem: (a) The system’s and the best ANN’s outputs (left). The error between the system’s and the best ANN’s outputs. The time span is $\Delta t = 6$ (middle). The best ANN’s prediction error for the increased number of prediction steps (right), (b) Evolution of ANN’s connections and hidden nodes, (c) Evolution of ANN’s performance and mutations. The average results over 30 runs are shown in (b) and (c).

after each architectural mutation in order to reduce the behavioral disruption to an individual. The training mutation is always attempted first before any architectural mutation since it causes less behavioral disruption. A hidden node is not added to an existing ANN at random, but through splitting an existing node.

In order to reduce the noise in fitness evaluation, EPNet evolves ANN architectures and weights simultaneously. Each individual in a population evolved by EPNet is an ANN with

weights. The evolution simulated by EPNet is closer to the Lamarckian evolution than to the Darwinian one. Learned weights and architectures in one generation are inherited by the next generation. This is quite different from most genetic approaches where only architectures not weights are passed to the next generation.

EPNet encourages parsimony of evolved ANN’s by ordering its mutations, rather than using a complexity (regularization) term in the fitness function. It avoids the tedious trial-and-

error process to determine the coefficient for the complexity term. The effectiveness of the method has been shown by the compact ANN's evolved by EPNet, which have very good generalization ability.

EPNet has been tested on a number of benchmark problems, including the N parity problem, the two-spiral problem [16], the four medical diagnosis problems, the Australian credit card assessment problem, and the Mackey–Glass time series prediction problem. Very competitive results have been produced by EPNet in comparison with other algorithms. EPNet imposes very few constraints on feasible ANN architectures, and thus faces a huge search space of different ANN's. It can escape from structural local minima due to its global search capability. The experimental results have shown that EPNet can explore the ANN space effectively. It can discover novel ANN's which would be very difficult to design by human beings.

EPNet would be most useful for applications where the ANN designing and training time is not critical, since it searches a much larger space than that searched by most other constructive or pruning algorithms and thus may require longer computation time. The computation time of EPNet could be estimated by the method given in Section IV-B7. In its current implementation, EPNet may have too many user specified parameters although this is not unusual in the field. Fortunately, these parameters are not very sensitive to moderate changes. No attempts were made in EPNet's experiments to optimize parameters. Parameters were set after some limited preliminary experiments. One of the future improvements to EPNet would be to reduce the number of parameters or make them adaptive.

ACKNOWLEDGMENT

The authors are grateful to anonymous referees for their constructive comments which helped to improve the paper significantly.

REFERENCES

- [1] L. J. Fogel, A. J. Owens, and M. J. Walsh, *Artificial Intelligence Through Simulated Evolution*. New York: Wiley, 1966.
- [2] D. B. Fogel, *System Identification Through Simulated Evolution: A Machine Learning Approach to Modeling*. Needham Heights, MA: Ginn, 1991.
- [3] D. B. Fogel, *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. New York: IEEE Press, 1995.
- [4] Y. Liu and X. Yao, "Evolutionary design of artificial neural networks with different nodes," in *Proc. 1996 IEEE Int. Conf. Evolutionary Computation (ICEC'96)*, Nagoya, Japan, 1996, pp. 670–675.
- [5] S. E. Fahlman and C. Lebiere, "The cascade-correlation learning architecture," in *Advances in Neural Information Processing Systems 2*, D. S. Touretzky, Ed. San Mateo, CA: Morgan Kaufmann, 1990, pp. 524–532.
- [6] J.-P. Nadal, "Study of a growth algorithm for a feedforward network," *Int. J. Neural Syst.*, vol. 1, pp. 55–59, 1989.
- [7] N. Burgess, "A constructive algorithm that converges for real-valued input patterns," *Int. J. Neural Syst.*, vol. 5, no. 1, pp. 59–66, 1994.
- [8] R. Setiono and L. C. K. Hui, "Use of a quasineutron method in a feedforward neural-network construction algorithm," *IEEE Trans. Neural Networks*, vol. 6, pp. 273–277, 1995.
- [9] R. Reed, "Pruning algorithms—A survey," *IEEE Trans. Neural Networks*, vol. 4, pp. 740–747, 1993.
- [10] P. J. Angelino, G. M. Saunders, and J. B. Pollack, "An evolutionary algorithm that constructs recurrent neural networks," *IEEE Trans. Neural Networks*, vol. 5, pp. 54–65, 1994.
- [11] G. F. Miller, P. M. Todd, and S. U. Hegde, "Designing neural networks using genetic algorithms," in *Proc. 3rd Int. Conf. Genetic Algorithms Their Applications*, J. D. Schaffer, Ed. San Mateo, CA: Morgan Kaufmann, 1989, pp. 379–384.
- [12] D. B. Fogel, L. J. Fogel, and V. W. Porto, "Evolving neural networks," *Biol. Cybern.*, vol. 63, pp. 487–493, 1990.
- [13] J. R. McDonnell and D. Waagen, "Evolving recurrent perceptrons for time-series modeling," *IEEE Trans. Neural Networks*, vol. 5, pp. 24–38, 1994.
- [14] ———, "Neural-network structure design by evolutionary programming," in *Proc. 2nd Annu. Conf. Evolutionary Programming*, D. B. Fogel and W. Atmar, Eds. La Jolla, CA: Evolutionary Programming Soc., 1993, pp. 79–89.
- [15] D. B. Fogel, "Using evolutionary programming to create neural networks that are capable of playing tic-tac-toe," in *Proc. 1993 Int. Joint Conf. Neural Networks (IJCNN'93)*. New York, NY: IEEE Press, 1993, pp. 875–880.
- [16] X. Yao and Y. Liu, "Toward designing artificial neural networks by evolution," in *Proc. Int. Symp. Artificial Life and Robotics (AROB)*, Beppu, Oita, Japan, Feb. 18–20, 1996, pp. 265–268.
- [17] J. D. Schaffer, D. Whitley, and L. J. Eshelman, "Combinations of genetic algorithms and neural networks: A survey of the state of the art," in *Proc. Int. Wkshp. Combinations Genetic Algorithms Neural Networks (COGANN-92)*, D. Whitley and J. D. Schaffer, Eds. Los Alamitos, CA: IEEE Computer Soc. Press, 1992, pp. 1–37.
- [18] X. Yao, "Evolution of connectionist networks," in *Preprints Int. Symp. AI, Reasoning and Creativity*, T. Dartnall, Ed. Queensland, Australia: Griffith Univ., pp. 49–52, 1991.
- [19] X. Yao, "A review of evolutionary artificial neural networks," *Int. J. Intell. Syst.*, vol. 8, no. 4, pp. 539–567, 1993.
- [20] ———, "Evolutionary artificial neural networks," *Int. J. Neural Syst.*, vol. 4, no. 3, pp. 203–222, 1993.
- [21] ———, "Evolutionary artificial neural networks," in *Encyclopedia of Computer Science and Technology*, A. Kent and J. G. Williams, Eds. New York: Marcel Dekker, vol. 33, pp. 137–170, 1995.
- [22] D. Whitley, T. Starkweather, and C. Bogart, "Genetic algorithms and neural networks: Optimizing connections and connectivity," *Parallel Computing*, vol. 14, pp. 347–361, 1990.
- [23] D. Whitley and T. Starkweather, "Optimizing small neural networks using a distributed genetic algorithm," in *Proc. Int. Joint Conf. Neural Networks*. Hillsdale, NJ: Lawrence Erlbaum, vol. 1, 1990, pp. 206–209.
- [24] X. Yao and Y. Shi, "A preliminary study on designing artificial neural networks using co-evolution," in *Proc. IEEE Singapore Int. Conf. Intell. Contr. Instrumentation*, Singapore, June 1995, pp. 149–154.
- [25] D. B. Fogel, "Phenotypes, genotypes, and operators in evolutionary computation," in *Proc. 1995 IEEE Int. Conf. Evolutionary Computation (ICEC'95)*. New York: IEEE Press, 1995, pp. 193–198.
- [26] V. Maniezzo, "Genetic evolution of the topology and weight distribution of neural networks," *IEEE Trans. Neural Networks*, vol. 5, pp. 39–53, 1994.
- [27] R. K. Belew, J. McInerney, and N. N. Schraudolph, "Evolving networks: Using genetic algorithm with connectionist learning," *Computer Sci. Eng. Dept., Univ. California-San Diego, Tech. Rep. CS90-174* revised, Feb. 1991.
- [28] P. J. B. Hancock, "Genetic algorithms and permutation problems: A comparison of recombination operators for neural net structure specification," in *Proc. Int. Wkshp. Combinations of Genetic Algorithms and Neural Networks (COGANN-92)*, D. Whitley and J. D. Schaffer, Eds. Los Alamitos, CA: IEEE Computer Soc. Press, 1992, pp. 108–122.
- [29] R. Smalz and M. Conrad, "Combining evolution with credit apportionment: A new learning algorithm for neural nets," *Neural Networks*, vol. 7, no. 2, pp. 341–351, 1994.
- [30] S. V. Odri, D. P. Petrovacki, and G. A. Krstonosic, "Evolutional development of a multilevel neural network," *Neural Networks*, vol. 6, no. 4, pp. 583–595, 1993.
- [31] X. Yao and Y. Liu, "Evolving artificial neural networks for medical applications," in *Proc. 1995 Australia–Korea Joint Wkshp. Evolutionary Computa.*, Kaist, Taejon, Korea, Sept. 1995, pp. 1–16.
- [32] ———, "Evolving artificial neural networks through evolutionary programming," in *Evolutionary Programming V: Proc. 5th Annu. Conf. Evolutionary Programming*, L. Fogel, P. Angelino, and T. Bäck, Eds. Cambridge, MA: MIT Press, 1996, pp. 257–266.
- [33] ———, "Evolutionary artificial neural networks that learn and generalize well," in *1996 IEEE Int. Conf. Neural Networks*, Washington, DC, vol. on Plenary, Panel, and Special Sessions. New York: IEEE Press, June 3–6, 1996, pp. 159–164.

- [34] Y. Liu and X. Yao, "A population-based learning algorithm which learns both architectures and weights of neural networks," *Chinese J. Advanced Software Res.*, vol. 3, no. 1, pp. 54–65, 1996.
- [35] P. J. Werbos, *The Roots of Backpropagation: From Ordered Derivatives to Neural Networks and Political Forecasting*. New York: Wiley, 1994.
- [36] L. Prechelt, "Proben1—A set of neural network benchmark problems and benchmarking rules," Fakultät für Informatik, Univ. Karlsruhe, Karlsruhe, Germany, Tech. Rep. 21/94, Sept. 1994.
- [37] X. Yao, "An empirical study of genetic operators in genetic algorithms," *Microprocessing Microprogramming*, vol. 38, pp. 707–714, 1993.
- [38] D. H. Ackley and M. S. Littman, "A case for Lamarckian evolution," in *Artificial Life III, SFI Studies in the Sciences of Complexity*, C. G. Langton, Ed. Reading, MA: Addison-Wesley, vol. XVIII, 1994, pp. 487–509.
- [39] D. Whitley, S. Gordon, and K. Mathias, "Lamarckian evolution, the Baldwin effect, and function optimization," in *Parallel Problem Solving from Nature (PPSN) III*, Y. Davidor, H.-P. Schwefel, and R. Männer, Eds. Berlin: Springer-Verlag, 1994, pp. 6–15.
- [40] D. Whitley and T. Starkweather, "GENITOR II: A distributed genetic algorithm," *J. Experimental Theoretical Artificial Intell.*, vol. 2, pp. 189–214, 1990.
- [41] G. Syswerda, "A study of reproduction in generational and steady-state genetic algorithms," in *Foundations of Genetic Algorithms*, G. J. E. Rawlins, Ed. San Mateo, CA: Morgan Kaufmann, 1991, pp. 94–101.
- [42] G. B. Fogel and D. B. Fogel, "Continuous evolutionary programming: Analysis and experiments," *Cybern. Syst.*, vol. 26, pp. 79–90, 1995.
- [43] F. Vavak and T. C. Forgy, "Comparison of steady-state and generational genetic algorithms for use in nonstationary environments," in *Proc. 1996 IEEE Int. Conf. Evolutionary Computa. (ICEC'96)*. New York: IEEE Press, 1996, pp. 192–195.
- [44] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing: Explorations in the Microstructures of Cognition*, D. E. Rumelhart and J. L. McClelland, Eds. Cambridge, MA: MIT Press, vol. I, 1986, pp. 318–362.
- [45] P. P. C. Yip and Y.-H. Pao, "Growing neural networks using guided evolutionary simulated annealing," in *Proc. 3d Annu. Conf. Evolutionary Prog.*, A. V. Sebald and L. J. Fogel, Eds. Singapore: World Scientific, 1994, pp. 17–25.
- [46] Y. Shang and B. Wah, "Global optimization for neural network training," *IEEE Computer*, vol. 29, no. 3, pp. 45–54, 1996.
- [47] X. Yao and Y. Liu, "Fast evolutionary programming," in *Evolutionary Programming V: Proc. 5th Annu. Conf. Evolutionary Programming*, L. Fogel, P. Angeline, and T. Bäck, Eds. Cambridge, MA: MIT Press, 1996, pp. 451–460.
- [48] W. Finnoff, F. Hergent, and H. G. Zimmermann, "Improving model selection by nonconvergent methods," *Neural Networks*, vol. 6, pp. 771–783, 1993.
- [49] X. Yao and Y. Liu, "Ensemble structure of evolutionary artificial neural networks," in *Proc. 1996 IEEE Int. Conf. Evolutionary Computa. (ICEC'96)*. New York: IEEE Press, 1996, pp. 659–664.
- [50] X. Yao, Y. Liu, and P. Darwen, "How to make best use of evolutionary learning," in *Complex Systems: From Local Interactions to Global Phenomena*, R. Stocker, H. Jelinek, and B. Durnota, Eds. Amsterdam, The Netherlands: IOS, 1996, pp. 229–242.
- [51] P. Darwen and X. Yao, "Automatic modularization by speciation," in *Proc. 1996 IEEE Int. Conf. Evolutionary Computa. (ICEC'96)*. New York: IEEE Press, 1996, pp. 88–93.
- [52] D. Stork and J. Allen, "How to solve the N -bit parity problem with two hidden units," *Neural Networks*, vol. 5, no. 6, pp. 923–926, 1992.
- [53] L. Prechelt, "Some notes on neural learning algorithm benchmarking," *Neurocomputing*, vol. 9, no. 3, pp. 343–347, 1995.
- [54] D. Michie, D. J. Spiegelhalter, and C. C. Taylor, *Machine Learning, Neural and Statistical Classification*. London: Ellis Horwood, 1994.
- [55] A. Roy, S. Govil, and R. Miranda, "An algorithm to generate radial basis function (RBF)-like nets for classification problems," *Neural Networks*, vol. 8, pp. 179–201, 1995.
- [56] K. P. Bennett and O. L. Mangasarian, "Robust linear programming discrimination of two linearly inseparable sets," *Optimization Methods Software*, vol. 1, pp. 23–34, 1992.
- [57] W. Schiffmann, M. Joost, and R. Werner, "Synthesis and performance analysis of multilayer neural network architectures," Univ. Koblenz, Inst. für Physics, Koblenz, Germany, Tech. Rep. 16/1992, 1992.
- [58] J. D. Farmer and J. J. Sidorowich, "Predicting chaotic time series," *Phys. Rev. Lett.*, vol. 59, no. 8, pp. 845–847, 1987.
- [59] M. Mackey and L. Glass, "Oscillation and chaos in physiological control systems," *Sci.*, vol. 197, p. 287, 1977.
- [60] T. M. Martinez, S. G. Berkovich, and K. J. Schulten, "Neural-gas' network for vector quantization and its application to time-series prediction," *IEEE Trans. Neural Networks*, vol. 4, pp. 558–569, 1993.
- [61] R. S. Crowder, "Predicting the Mackey–Glass time series with cascade-correlation learning," in *Proc. 1990 Connectionist Models Summer School*, 1990, pp. 117–123.
- [62] J. Moody and C. J. Darken, "Fast learning in networks of locally tuned processing units," *Neural Computa.*, vol. 1, pp. 281–294, 1989.



Xin Yao (M'91–SM'96) received the B.Sc. degree from the University of Science and Technology of China (USTC), Hefei, Anhui, P.R.C., in 1982, the M.Sc. degree from the North China Institute of Computing Technologies (NCI), Beijing, P.R.C., in 1985, and the Ph.D. degree from USTC in 1990.

He is currently a Senior Lecturer in the School of Computer Science, University College, the University of New South Wales, Australian Defence Force Academy (ADFA), Canberra. He held postdoctoral fellowships in the Australian National University (ANU) and the Commonwealth Scientific and Industrial Research Organization (CSIRO), Division of Building, Constructions, and Engineering (DBCE), before joining ADFA in 1992. He has published a number of papers in the fields of evolutionary computation and neural networks.

Dr. Yao is the Program Committee Cochair for the IEEE ICEC'97 in Indianapolis and covice-chair for IEEE ICEC'98 in Anchorage. He is also the Program Committee Chair for the Eighth Australian Joint Conference on AI (AI'95), Cochair for the First Asia-Pacific Conference on Simulated Evolution And Learning (SEAL'96), and a Program Committee Member of many other international conferences. He is an Associate Editor of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION and a Member of the IEEE NNC Technical Committee on Evolutionary Computation.



Yong Liu was born in Wuhan, P.R. China, in 1966. He received the BSc degree from Wuhan University, P.R. China, in 1988, and the M.Sc. degree from Huazhong University of Science and Technology, P.R. China, in 1991, both in computational mathematics.

In 1994, he was a Lecturer at Wuhan University. Since the end of 1994 to 1995, he was a Visiting Fellow in the School of Computer Science, University College, the University of New South Wales, Australian Defence Force Academy, Canberra. He is currently a Ph.D. candidate at the same university.

He has published a number of papers in international journals and conferences, and is the coauthor of the book *Genetic Algorithms* (Beijing, China: Science Press, 1995). His research interests include neural networks, evolutionary algorithms, parallel computing, and optimization.