Deriving Application-Specific Neural Nets Using a Massively Parallel Genetic Algorithm

H. B. Penfold
Dept. of Electrical and Computer Engineering,
The University of Newcastle, N.S.W. 2308, Australia,

U. Kohlmorgen
H. Schmeck
Institut für Angewandte Informatik und Formale Beschreibungsverfahren,
Universität Karlsruhe, Karlsruhe 7500. Germany

Submitted to Complex Systems

Abstract

This paper describes a genetic algorithm which has been applied to the derivation of both the structure and the connective weights for simple computational networks similar to recurrent neural nets. The algorithm has been implemented on a massively parallel Single-Instruction-Multiple-Data (SIMD) architecture computer. It incorporates a breeding model, and a mutation model which applies to both the genetic descriptor and the transcription from the gene to the phenotype.

Advantages of the method include the general connectivity of the resultant network with the potential for economy of structure, the ability to consider complex performance criteria (for example fault-tolerance), and the freedom to choose any desired node transfer function – not merely those functions which are continuously differentiable. The massively parallel structure allows very large population sizes (32K) at each generation, and convergence typically requires few (< 20) generations.

Examples include a single, double and triple fault-tolerant solution to the exclusive or problem, and a 2-node solution for a 6-input incomplete even parity problem.

1 Introduction

The dominant techniques in the training of neural networks tend to be variants and extensions of the backpropagation method due to Rummelhart *et al* [1]. The feature shared by such methods is that they apply only to networks having continuously differentiable neuron transfer functions and in which the interconnecting weights are modified in the light of some calculus-based optimisation procedure. It is well known that these methods suffer from convergence problems in that they can be 'trapped' by local minima, and that convergence rates fall dramatically for large networks [2]. Also, it is not clear exactly what structure should be assumed for the network to be trained, although heuristics do exist to guide the choice. These however do not necessarily lead to minimum or even to economical solutions.

Optimisation techniques other than backpropagation have been evaluated in an attempt to add desirable stochastic attributes to the gradient-descent properties of that method. This work, of which simulated annealing is an example, has a long history [3], and employs random perturbations to the weights of a network in an attempt to span the solution space in search of the optimum solution. Such methods are reminiscent of the biological phenomenon of mutation in which small random changes are presumed in the describing gene of an organism to allow it an opportunity to better 'fit' its environment.

Genetic algorithms (GA's) have been extensively assessed for application to the training of neural networks, and a current and comprehensive review of the field has been provided by Yao [4]. They are generally employed to investigate possible architectures or configurations of connecting weights for a network – often in association with a learning algorithm. They are typically based on a 'genetic descriptor' or 'gene' encoding the configuration of the network(s) as either a binary or real number string. Variation is introduced by either 'splicing' together segments of the gene for selected solutions, or by subjecting the gene to random - and usually small - perturbations in the numerical values. The former process is thought to be analogous to breeding, and the latter analogous to mutation.

We extend this biological metaphor in this paper and observe several motivating issues which are thought to add impetus to this line of investigation.

- It is desirable to incorporate in networks node transfer functions arbitrarily defined, and not limited to the class of differentiable functions as required by calculus-based gradient descent methods.
- It is useful to be able to specify optimisation criteria which may not be easily captured by a mathematically formulated cost function.
- It is desirable to encompass in the method the derivation of solutions with a 'general' interconnection between nodes. That is, the networks may be recurrent.
- It is instructive to investigate the configuration of networks adequate for a given problem, and not *a priori* specified by the investigator.
- Because solutions are specific to the problem stated, they cannot be presumed to have a capacity to solve any other problem. Indeed, there is no evident way to modify them with this in mind; they are *Application-Specific*.

We define a genetic structure, in this case an ordered set of numbers representing the weights of connections between nodes in (or on inputs to) the net, and the bias of the nodes. The process of breeding requires that genes be selected from the string describing each 'parent' network (the chromosomes). We also generate the complementary 'sibling' network to ensure that the requirements for gene retention is observed [6]. During the breeding process, mutations are permitted to occur, their incidence and severity being governed by a measure of the genetic diversity in the population.

The algorithm typically begins with a specified number of randomly generated nets, and produces the next generation according to rules. The members of this generation are tested against the required task and culled on the basis of performance to derive the next parent set. If performance improvement stagnates, the probability of mutation is increased. Initial development and evalustion of the method were conducted on a standard workstation [5], but the algorithm has now been implemented on a massively parallel Single Instruction, Multiple Data (SIMD) machine – a MasPar MP-1 – an architecture well suited to problems of this type.

2 Methodology

We adopt a conventionally described, but general, network comprising nodes which generate an output which is a function of the weighted sum of their inputs. Thus, for an n-node network having m data inputs, the output S_j of node j is:

$$S_j = \mathcal{F}_j(\sum_{i=1}^{n} w_{ij} S_i + \sum_{k=1}^{m} w_{kj} \zeta_k + \theta_j) \tag{1}$$

where $\mathcal{F}_j$ may be any desired transfer function for that node, ζ_k are inputs to the network, θ_j are bias inputs, and $w_{\cdot j}$ are weights on connections to node j. Note that node j may be self-connected ($w_{jj} \neq 0$) and that the recurrent property means that the performance assessment of a network must include checking that the network output relaxes to a stable condition. That is, of course, unless the stipulated behaviour is in fact cyclic or periodic.

For a genetic algorithm to be definable, we need to resolve the structure of a data string which we regard as analogous to a gene, a method for combination of sections of this data string which is analogous to breeding, and the mutation mechanism for variations to the data string or to its interpretation.

2.1 Gene Definition

There is considerable latitude in the selection of a suitable string of real numbers such that it will represent an arbitrary network. We have chosen to represent the connectivity

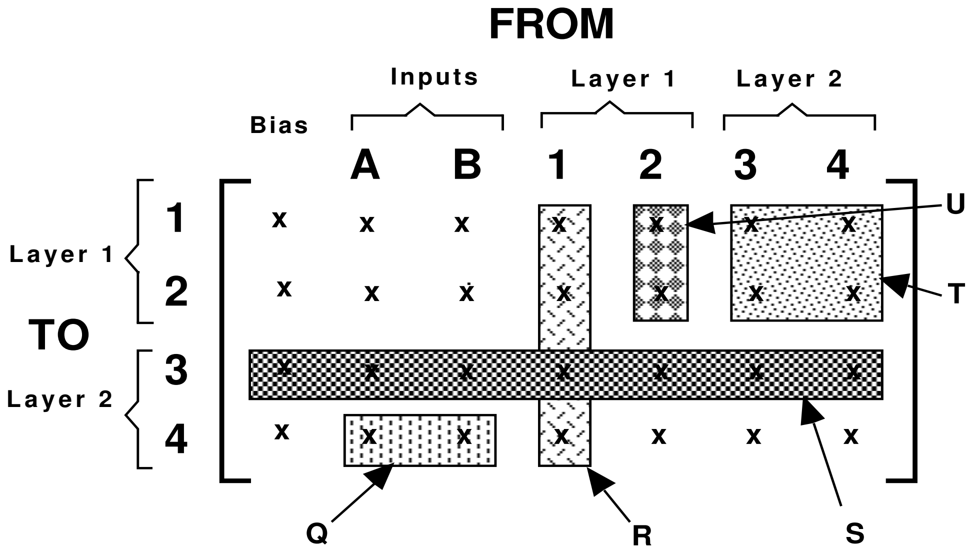

Figure 1: Network Connection Matrix

of a network as a matrix of weights where the real number elements (in the range -1 to +1) represent the weights on interconnections of the inputs, biases, and between nodes.

An example of this connection matrix is given in Figure 1 for a 2-input, 4-node network. The concept of a layer is no longer clear in a generally connected network, but we have chosen to refer to a layer as comprising a number of nodes equal to the number of data inputs. Thus the total number of nodes in a network equals the number of inputs times the (arbitrarily defined) number of layers. We will consider segments of the connection matrix as constituting genes which we assemble into a string to represent the chromosome for this network. The choice is open, and the possibilities are indicated on Figure 1. Structure Q represents all inputs to a node from a layer; structure R represents all outputs from a node. Structure S represents all inputs to a node; T represents all inputs to a layer from a layer; U represents all outputs from a node to a layer.

The choice of gene will affect the degree to which structural properties of the parents will appear in offspring; the fine-grain properties correspond to structures like Q (or even individual entries) and coarse structures to those such as S. We have experimented with different choices – space precludes their discussion here – and have found that the choice of structure U gives reliable convergence, with an appropriate balance between the stability endowed by breeding and the randomness introduced by mutation. Examples in this paper have been derived using this gene representation. The string of genes so defined is assembled into a row vector constituting the chromosome.

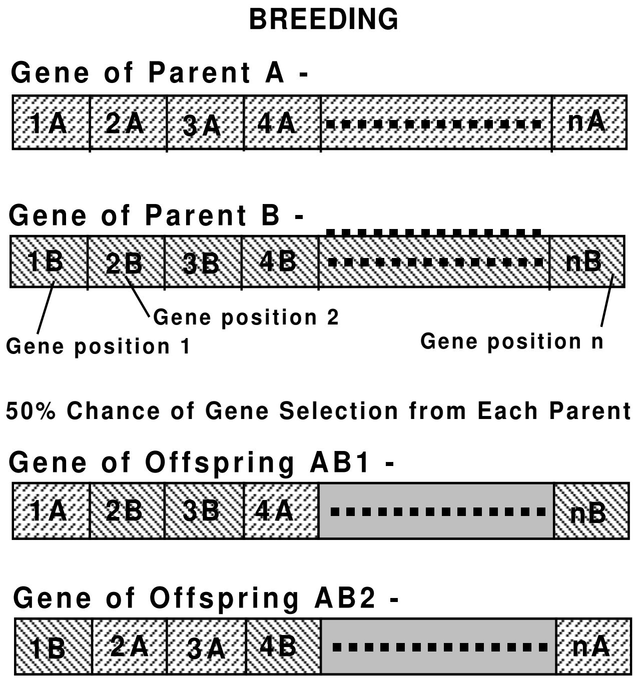

Figure 2: Breeding Rule

2.2 Breeding

Much of the work which represents a 'breeding' function between networks uses the crossover operator in which all bits up to a randomly chosen position in the gene of one parent are copied to the offspring, with the remaining bits being copied from the other parent. Whilst this is well founded in optimisation theory we have chosen to depart from this practice. Instead each gene in the offspring is selected at random with equal probability from the two parents. A second offspring is simultaneously generated from the complementary genes.

Our justification for this method is at present heuristic, but bears a clear parallel with biology. Experience with the procedure suggests that the method produces offspring which are 'topologically similar' to the parent generation. The procedure is depicted in Figure 2.

2.3 Mutation

The other genetic operator usually incorporated is mutation. Indeed, many algorithms rely solely on mutation for their search of the optimisation space [3]. We treat mutation as a method to reintroduce genetic diversity into a population when the preservation of structural similarity imposed by a breeding rule loses that diversity without having reached optimum performance. We have also introduced a mutation-like process in the rules which govern the transcription from the chromosome string back into the connectivity matrix and the implemented network.

Gene mutation is accounted for by adding to selected weights (a single gene) a zero-mean Gaussian random variable. The gene selection criterion is a uniformly distributed random variable which ranges up to the number of genes in the chromosome; the number of such mutated genes is controlled by the user, but usually is chosen to increase as the population diversity diminishes provided the fitness of solutions has not increased significantly between generations.

There is a well-defined correspondence between the position of a gene in the connectivity matrix, and its position in the chromosome vector. We have incorporated transcription mutations where the interpretation of the chromosome after breeding is in error. Three types of erroneous transcription are allowed: repetition of a gene, incorrect insertion of a single gene, and the omission of one or more genes from the sequence. The occurrence of transcription mutations is governed by appropriate random processes. This mutation type permits the algorithm to explore variation in the *structural* properties of the networks – for example the copying of one part of a network into another part, and complements the data mutations discussed above.

2.4 Parallel Implementation

It is easy to recognise that a genetic algorithm is highly appropriate for implementation on a massively parallel SIMD machine. For each candidate solution, the steps of breeding, mutation, and fitness testing invoke the same instructions, albeit on different data. The MasPar MP-1 is a fine-grained massively parallel SIMD machine with a processor element (PE) array containing 1K, 2K, 4K, 8K, or 16K processors. Each PE is a load/store arithmetic processor with dedicated register space and RAM.

The algorithm was implemented by associating with both the row and column addresses of the PE array corresponding members of the parent set for the next generation. Thus the intersection of, for example, row 10 with column 20 indicates the PE responsible for the generation of offspring from the 10th and 20th most fit members of the previous generation. Of course, the diagonal represents a parent breeding with itself and is desirable for the retention of successful genetic characteristics into the next generation. After the evaluation of fitness, the PE addresses are ordered by this variable and chromosomes from the best 128 (on a 16K machine) are re-allocated to the array for the next generation. At each generation the front-end processor evaluates the progress of the algorithm and allocates control variables as necessary.

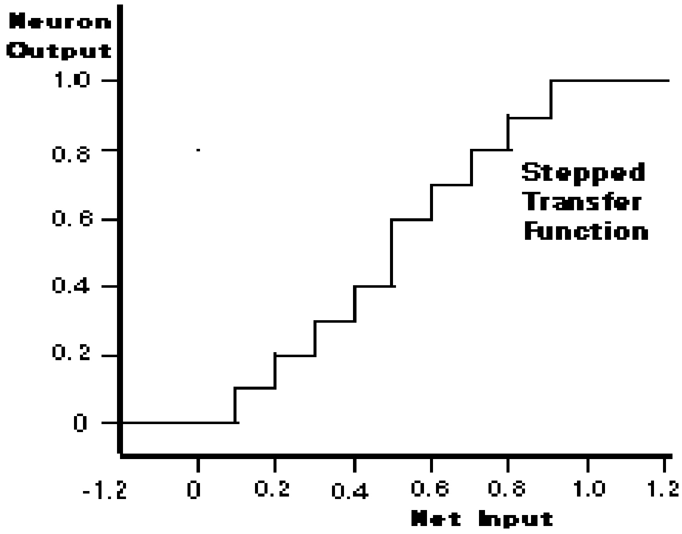

Figure 3: Stepped Transfer Function

The algorithm has been run on both 4K and 16K machines; in the former case there are 8,064 new individuals per generation, and in the latter 32,512. Typical runs may require 20 or so generations (for a difficult problem like fault-tolerance) with a very large number of candidate solutions being considered but with modest run times – typically less that a minute.

3 Test Results

Two example problems are presented here to demonstrate the flexibility of the method. Although they are both Boolean problems, the method has also been used successfully on problems involving real variables.

Each of the following problems was solved for nodes incorporating the "stepped" transfer function shown in Figure 3. We have found that this function is more reliable for Boolean problems than, for axample, sigmoids, and has the advantage of saturation at 0 and 1. It thereby avoids problems with large weights being needed to obtain saturated outputs. The stepped function was derived from a quantised approximation to the sigmoid form. In each instance, Node 1 was defined to be the network output node. The computer used for these tests had 16K processors.

3.1 Parity Problem

The training set for the incompletely specified six input even parity problem is shown in Table 1. Ten data cases were chosen, with five having an even number of bits set, and five an odd number. If back-propagation were to be used for this problem, one would probably begin with a network having at least 10 nodes, and several thousand presentations of the data would probably be necessary [7].

Table 2(a) and Table 2(b) present two sample solution to the problem. The two solutions are markedly different, yet both solve the problem and require, surprisingly, only 2 nodes. Several other solutions – requiring more than 2 nodes – were disclosed by a single run of the genetic algorithm.

x_1	x_2	x_3	x_4	x_5	x_6	z
0	0	0	0	0	0	1
0	1	0	0	0	0	0
0	0	0	0	0	1	0
1	0	1	0	0	0	1
0	0	0	1	0	1	1
0	1	0	1	0	0	1
0	0	1	0	1	0	1
1	0	1	0	1	0	0
0	0	1	0	1	1	0
1	1	1	1	0	1	0

Table 1: Six-Input Parity Problem

To	Node	Input Connections						From Node	
Node	Bias	x_1	x_2	x_3	x_4	x_5	x_6	1	2
1	0.67	-0.345	-0.618	0	1.0	-0.251	-1.0	0.886	-0.015
2	0.589	0	0	-0.525	0.301	-0.803	0.786	0	0.686

Table 2(a): Parity Problem Results - Network 1

To	Node	Input Connections						From Node	
Node	Bias	x_1	x_2	x_3	x_4	x_5	x_6	1	2
1	0.546	-0.345	-0.641	0	1.0	-0.251	-0.581	0.794	0
2	0	-0.583	0.840	0.218	0	-0.378	-0.33	0	0.738

Table 2(b): Parity Problem Results - Network 2

3.2 Fault-Tolerant XOR

This problem requires the network to learn the 2 input exclusive-or problem. However, correct performance of the task by an intact network is only part of the selection criterion. It is also required that the network continue to perform the task correctly under the condition of node failure. The only 'protected' node is the output node, Node 1. This means that the network is to be fault-tolerant for any single (or double, or triple) node failure. Networks with 6 nodes were specified as the starting population, but if the problem could be solved with fewer nodes, then these more economical solutions would be preferred.

It is problematic in breeding fault-tolerant solutions whether one should first select for a population which is capable of solving the problem with all nodes intact, and then selecting from that population fault-tolerant structures. Alternatively, selection on both criteria could proceed simultaneously. We have elected to use the following 'simultaneous convergence' strategy which first develops a population which is close to solution, and then emphasising the fault-tolerant aspect. This procedure was necessary to prevent the fault-tolerant selection from operating on a population which could in fact solve the OR problem, which differs by only one bit from the required solution. The procedure to be described details the method used for the case of the single and double fault-tolerant networks. It will be noted later, however, that a small variation to this procedure was made for the case of the triple fault tolerant network.

Denote by d_i and a_i, $i = 1, 2, 3, 4$ the desired network output and the actual intact network output respectively. The subscript i denotes the four possible input cases for 2-input XOR. Further, denote by f_{ij} the faulted network output over the four input cases, i, with the subscript $j = 1, \ldots, n$ indexing the possible fault conditions to be evalusted. For a 4-node network with a single node fault, $n = 3$; for a 6-node network with one or two faulted nodes, $n = 15$ – there are 5 single node fault possibilities, plus 10 two-node possibilities. Then define,

$$S = \sum_{i=1}^{4} |d_i - a_i| \tag{2}$$

as the aggregate performance score for the intact network, Also define,

$$F = \sum_{j=1}^{n} \sum_{i=1}^{4} |f_{ij} - a_i| \tag{3}$$

as the composite score for the faulted cases. Note that F is a measure of the degradation to the network performance, and is not dependent on the specific problem to be solved.

The simultaneity is introduced as follows, where S_F is the overall score to be assigned to a candidate solution, $\overline{S_F}$ is the average of S_F over all candidate solutions of the previous generation, and c is a constant chosen as a tuning parameter.

$$S_F = \begin{cases} S + (1 - \overline{S_F})F & \text{if } S < c \\ S & \text{otherwise} \end{cases} \tag{4}$$

For the cases reported here, c was chosen as 1, to ensure that the network concerned was close to the XOR solution in its intact condition. If this were so, the optimisation based on the score F was introduced gradually depending on the overall population performance as is evident in the above equation.

Table 3 shows the connectivity matrix for a 4 node solution. Any one of nodes 2, 3, or 4 may be disabled and the network retains its ability to perform the function. A simple redundancy-with-vote strategy would require seven nodes if a 2-node individual solution to XOR were adopted, so the presented solution achieves significant economy. Evidently the algorithm has disclosed a configuration in which the three, 3-node solutions intersect at one node.

Table 4 shows a 6-node network in which any *one or two* nodes may be disabled (again excepting the output node) and the network retains its function. The simple three-out-of-five approach would need 11 nodes to have the same fault-tolerance.

Trials were extended to find a *triple* fault-tolerant 6–node network for which any one, two, or three nodes can fail without affecting performance. Clearly this case would need a 4-out-of-seven voting strategy if implemented by a redundancy method, and would need 15 nodes. In this case a change was made to the scoring scheme previously outlined. Equation 3.2 was altered to

$$F = \sum_{j=1}^{n} \sum_{i=1}^{4} |f_{ij} - d_i| \tag{5}$$

so that F becomes a measure of the degradation to the desired network performance, and not from the intact network performance as earlier specified. The algorithm found the solution shown in Table 5 in 12 generations.

We regard these solutions as having significant implications for the study of fault-tolerant architectures for specific problems where no obvious (or simple) method exists to reduce the redundancy to less than the multiple-copy-with-vote strategy. In these experiments we have adopted the criterion that a node failure means that it remains at zero output for any input. This is equivalent to setting to zero all the elements in a column of the connectivity matrix, and obvioully generalises to other specifications of failure mode.

To Node	Node Bias	Inputs x_1	x_2	From Node 1	2	3	4
1	0.014	-0.477	0.466	1.0	0.675	1.0	0.862
2	0.018	0.008	-0.147	-0.196	0.639	0.389	0.364
3	-0.067	0.478	-0.523	1.0	0.592	-0.131	0.15
4	0.088	0.333	-0.457	-0.151	-0.238	0.809	0.821

Table 3: Four-Node Fault-Tolerant XOR Network

To Node	Node Bias	Inputs x_1	x_2	From Node 1	2	3	4	5	6
1	0.029	-0.337	0.29	0.879	-0.5	0.544	0.605	0.772	0.09
2	-0.591	-1.0	-0.055	0.332	-0.113	0.363	-0.044	-0.224	0.589
3	-0.215	0.631	-0.368	0.485	0.736	0.794	0.26	-0.937	1.0
4	-0.228	1.0	-1.0	-0.363	1.0	-0.185	0.597	0.525	-0.008
5	-0.249	0.508	-0.575	0.353	0.199	0.091	-0.074	0.771	0.679
6	-0.716	-0.15	-0.285	0.326	0.366	-0.316	-0.415	-0.077	0.725

Table 4: Six-Node Double-Fault-Tolerant XOR Network

To Node	Node Bias	Inputs x_1	x_2	From Node 1	2	3	4	5	6
1	0.061	-0.504	0.462	0.982	0.457	0.940	1.000	1.000	1.000
2	-0.011	-0.985	0.502	-0.424	-0.012	0.776	0.961	0.196	0.954
3	-0.215	0.631	-0.368	0.349	0.248	0.480	0.206	0.157	-0.324
4	-0.228	1.000	-1.000	0.058	-0.114	-0.050	0.445	-0.473	0.056
5	-0.225	0.474	-0.472	0.100	0.477	0.323	0.183	0.696	0.030
6	0.063	0.095	-0.289	-0.142	0.461	0.220	-0.079	0.175	1.000

Table 5: Six-Node Triple-Fault-Tolerant XOR Network

4 Conclusion and Future Directions

The method has been shown to be an effective approach to the derivation of a network with specific capacities, especially those in which performance criteria are not easily captured in a simple objective function. Also the opportunity to choose arbitrary transfer functions for each node (not necessarily the same) is valuable, as is the investigation of minimal and fault-tolerant architectures which would be difficult or impossible by other methods. It is acknowledged that we employ a very large computing resource to derive the relatively simple networks given in the examples. However, it is regarded as important to examine the potential for "application-specific" networks as distinct from the usual practice of using a training algorithm on a general network to confer upon it certain properties. Of course, this approach can only be expected to be useful in a task environment which is static over time, and where the computing resource used to derive the application-specific network is small relative to the importance (or multiple use) of the solution network.

With regard to future directions for investigation, it is evident that the following aspects would benefit by clarification -

- The method of parallelizing the genetic algorithm presented in this Report is by no means the only one possible. We have elected to use the entire SIMD processor array to run a single large population in parallel. Alternatively one could divide the PE's into several groups with each group running the algorithm in parallel, and each with a different random initial population.
- The similarity of networks belonging to the solution set needs to examined to determine how many distinctly different networks are represented in the 128 solutions present in the final generation.
- It would be instructive to examine the process of convergence in more detail to determine whether networks selected for the next generation parent set are associated with PE's in distinct regions of the PE array, or whether they are randomly distributed across the array.
- The foundation in optimisation theory of the multiple cross-over breeding rule we employ needs to be clarified.
- Some of the observed results are suggestive of the usefulness of the method in exploring experimentally issues in evolution theory, such as the nature of speciation phenomena, and this aspect could be further studied.

ACKNOWLEDGEMENT: H. B. Penfold wishes to thank The Institut für Angewandte Informatik und Formale Beschreibungsverfahren at the University of Karlsruhe for its hospitality.

References

[1] D. E. Rummelhart, G. E. Hinton and R. J. Williams, "Learning Internal Representations by Error Propagation", in *Parallel Distributed Processing, Vol.1*, D. E. Rummelhart and J. L. McClelland Eds., MIT Press, pp. 318-362, 1986.

[2] G. E. Hinton, Connectionist Learning Procedures, Artificial Intelligence, 40: 185-234, 1989.

[3] L. J. Fogel, A. J. Owens, M. J. Walsh, *Artificial Intelligence Through Simulated Evolution*, John Wiley and Sons, 1967.

[4] Xin Yao, (1993) A Review of Evolutionary Artificial Neural Networks, International Journal of Intelligent Systems, *to appear*.

[5] H. B. Penfold, O. J. Diessel and M. W. Bentink,"A Genetic breeding Algorithm which exhibits Self-organizing in Neural Nets", IASTED Int. Conf. on Artificial Intelligence Applications and Neural Networks, Zurich, June 25–27, 1990.

[6] J. C. Schank and W. C. Wimsatt, "Generative Entrenchment and Evolution", PSA 1986 Volume 2 , pp 33-60.

[7] P. Bartlett and T. Downs, "Training a Neural Network with a Genetic Algorithm", First Australian Conference on Neural Networks, Sydney, January, 1990.