4
5

Design of Artificial Neural Networks Using Genetic Algorithms: review and prospect

İbrahim Kuşçu and Chris Thornton

Cognitive and Computing Sciences
University of Sussex
Brighton BN1 9QN
Email: ibrahim@cogs.susx.ac.uk christ@cogs.susx.ac.uk

April 30, 1994

Abstract

The design of Artificial Neural Networks by Genetic Algorithm is useful in terms of (1) automating and optimising the design and (2) finding biologically plausible models. This paper presents a review of the state of the art and research prospects in the area.

Keywords: Genetic algorithms, artificial neural networks.

1 Introduction

Research involving some sort of combination of Genetic Algorithms (GA) and Artificial Neural Networks (here forth ANNs or Networks for short) has been growing. Some recent work in the area can be found in [47] and a review of the state of the art is presented in [40]. This paper presents a review and research prospects for designing ANNs by GA.

The design of Artificial Neural Networks (ANN) using GAs can be helpful in terms of two main issues. First, it *automates* the design of the network which would otherwise have to be done by hand using trial and error. Second, the process of the design can be analogous to a biological process in which the ANN blueprints encoded in chromosomes *develop* through an evolutionary process.

Designing networks by hand may be very complex. Even though a design is found to be sufficient for a task by trial and error, the risk of missing more promising architecture is not eliminated. Given the complex combinations of performance criteria, such us learning speed, compactness, generalisation ability, and noise-resistance, it is very difficult to optimize a network design. The problem of designing an ANN for a specific problem involves searching the space of architectures for one which will perform best in meeting the requirements of the problem. The search space for such a problem may be infinitely large, un-differentiable, complex, noisy, deceptive and multi-modal [31].

The Schema Theorem developed by Holland [23] has proved useful in many applications involving large, complex and deceptive search spaces [12]. Thus, an evolutionary process based on genetic search can help to automate and improve, if not optimize, ANN design required to produce complex behaviors.

A major issue in genetic-based design of ANNs is that the representation (i.e., encoding strategy) should be able to capture all potentially useful designs for the task in hand while excluding any flawed or meaningless ones. In terms of using genetic search, this means that any representation schema should allow new, meaningful and valid network structures (i.e., a particular learning rule can be applied) to be produced by the genetic operators used (i.e., crossover or mutation). In such cases the representation is said to be 'closed' under the genetic operators. This is referred as the structural/functional problem in [37]. The possible network parameters would include the number of layers, the number of units in a layer, the number of feedback connections allowed, the degree of connectivity from one layer to another, the learning rate and the error term utilized by the learning rule.

GAs are applied to neural networks in two different ways: they either employ a fixed network structure (i.e., the number of nodes and the connections among them are fixed) with connection weights under evolutionary control (see for example [35] [46] [48] [8] [4] [6]) or they are used in designing the structure of the network itself.

This paper is concerned with the researches in the latter category. The researches in this category concentrate on two distinctive approaches:*direct encoding* and *generative encoding*. In the case of direct encoding strategies [48] [18] [37] [36] [10] [31] [26] the architecture of the network is *directly* encoded onto the chromosome representation. In the case of the generative encoding strategies, however, some sort of grammar which *generates* network architectures is used [27] [13] [14] [15] [25] [16] [17] [5].

In the following sections we will start by describing *direct encoding* methods, and next, concentrate on some recent studies which use some sort of grammar in *generating* ANNs. Then, we will present some other research which uses evolutionary design of ANNs as a tool in pursuing their primary research interest. Finally, we will conclude with a discussion of future research directions in the area.

2 Direct Encoding Methods

Early strategies for genetic-based ANN design can be classified according to degree of *developmental specification*: degree of specificity employed in mapping from genotype to phenotype. Some of the design strategies (called 'weak') use a loose representation based on abstract genetic 'blueprints' which can be translated through 'developmental machinery' into a network phenotype [18]. These strategies are good at producing large networks efficiently. However, they impose a severe constraint on the network search space. They represent the layer of the network in a single gene facilitating the application of genetic operators for regular networks. However, they face a difficulty in encoding detailed connections. Other design strategies (called 'strong')are good at capturing patterns of connections in smaller networks more efficiently since they represent connections more directly in the chromosome.

2.1 The Genesis System: weak representation

In Genesis [18] a 'blueprint' representation of network structure is encoded on a bit string. It is composed of one or more segments. Each segment, in turn, contains an *area*, and its *projections* (i.e., connections). The first and the last areas in the representation show input and output areas respectively. Within each segment, a fixed length of bits is used to specify area parameters (APS) and one or more projection specification fields (PSFs) which describe connections between areas. Since there are an unknown number of areas or projections, the beginning and the end of these are marked so that strings can be parsed into network architectures and crossover operations can easily produce meaningful strings.

The APS contains some fields to describe the address and identification of the area, and the size (i.e., number of units) of it. In addition, 'dimension share' parameters determine the spatial organisation of the units. Since the representation used in Genesis does not assume a simple fully connected network structure, PSFs may be used to determine where a specific unit can make a connection. In PSFs the identity of the target area is coded either as an absolute (i.e., target ID itself) or relative address (i.e., position of the target area relative to current area) mode. Again, the dimension parameters allow connections only in that localised area. Finally, the degree of connectivity (between 30 to 100 percent) and learning rate parameter of back-propagation are also coded in PSFs.

A two point crossover is modified to allow identification of the points by referring to the markers in the blueprint. The variable length representation and the modified crossover seems to allow a much broader space of network architectures to be searched. This can result in more complex architectures.

This design strategy is used to solve two different problems: digit recognition and XOR problem. In both cases Genesis has produced reasonable networks and showed improvements over its initial random structures. But the over all results suggests that the representation used by Genesis is inadequate. More attention to representation of connectivity is needed. For example, a typical chromosome contains concatenation parameters describing the number of layers, size of the layers and how these layers are interconnected. Due to this abstraction, larger nets can be encoded with small chromosomes. However this is only true for some particular group of networks. This method would fail to encode some modular architectures with well defined and repeated groups of neurons.

2.2 The Innarvator System: strong representation

In the Innarvator System [31] a layered feed-forward network of N units is represented by a *connectivity constraint matrix* with dimensions $Nx(N+1)$. Each of the values of the matrix specified by (Column, Row) indices specifies the nature of the constraint of connection from one unit to another. The constraints can be either none (indicated by a zero), learnable (L), learnable but limited to positive values (L+) or learnable but limited to negative values (L-). The rows of the matrix are successively concatenated to form a bit string representation of a network. The following figure shows an example of a constraint matrix representing a 5 unit network (2 input and single output unit). The last column (i.e. the $N+1$th) in the matrix specifies the threshold biases of the units.

The representation used clearly defines the layers of the network and, thus, the translation from genotype to phenotype can be more easily interpreted. The crossover operator applied in this design strategy involves selecting a random row of the constraint matrix and swapping all the entries in that row between the parents. This is the simplest and the safest way of applying the crossover operator since any row contains a basic building block of a single unit and crossover always produces a valid network structure. The mutation operator involves simply moving along all the values in the matrix and choosing a new (0 or L) constraint with a specified probability.

The fitness evaluation is based on the success of learning an input/output mapping specified for the task. A possible fitness function could include some other criteria such as ability to generalise or the size of the network for a particular task.

In order to evaluate the fitness of a particular network, the constraint matrix is initialised with learnable and no connection values. Next, the learnable connections are given small random weights. The network is trained for a specific target mapping using the back-propagation algorithm. After the training the total sum squared error (TSSE) is used to derive the fitness.

This design strategy is applied for different tasks such as XOR and the four-quadrant

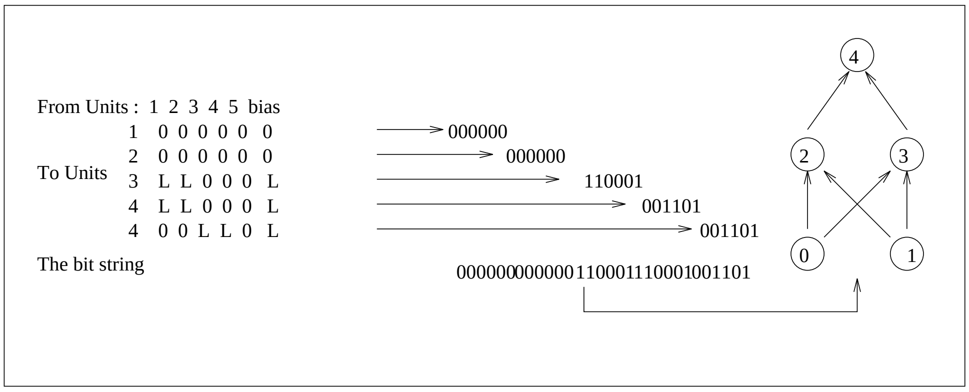

Figure 1: Encoding for XOR problem using strong representation (taken from [31]).

problem. The results have shown that such genetic based design can discover successful architectural solutions with faster learning of the tasks at hand. However, it is limited to encode a fixed number of neurons. It gives a chromosome of length n^2 for a network of n units. If the number of the units gets large the search space becomes too big. Moreover if the weights are also encoded then the crossover operation may result in non-functional off-springs: the structural/functional problem pointed out by [37].

3 Generative Strategies

3.1 Kitano's Grammar Encoding method

A system developed by Kitano [27] employs a different approach encoding ANN architectures. It uses a graph-generation grammar which can encode regular connectivity patterns with shorter chromosomes. Basically, it involves encoding a set of rules which can generate the ANN. Kitano argues that previous design strategies encode ANN configurations directly onto the chromosome and therefore require longer chromosome length and larger search space. As the size of the networks grows the time it takes to converge to a near-optimal configuration will increase. So, they are not suitable for designing large networks. Besides, these methods assume a rigid, one-to-one correspondence between the connectivity patterns and the generic information. This creates a substantial difficulty in encoding a network with repeated patterns and complex internal structure. Therefore, they are also biologically less plausible with respect to morphogenesis of the neural system.

In his design strategy Kitano's grammar generates a family of matrices of the size 2^k. The elements contained in these matrices are some characters of a finite alphabet. A larger matrix is developed using rewrite rules corresponding to these characters. This is translated to a connectivity matrix which describes the structure of an ANN.

Kitano's *grammar encoding method* is different from the previous methods where the structure of the network is not directly encoded in the chromosome. Rather, this method uses a set of re-write rules encoded in the chromosome to generate networks. It is based on Graph L-system which is an extension of Lindenmayer's L-system [28] [29]. Figure 2 shows the generation of a typical XOR network using Kitano's graph generation system.

The followings are the rules used in developing the connectivity matrix for the XOR network. Starting from the initial state "S" the graph is developed by rule-matching in

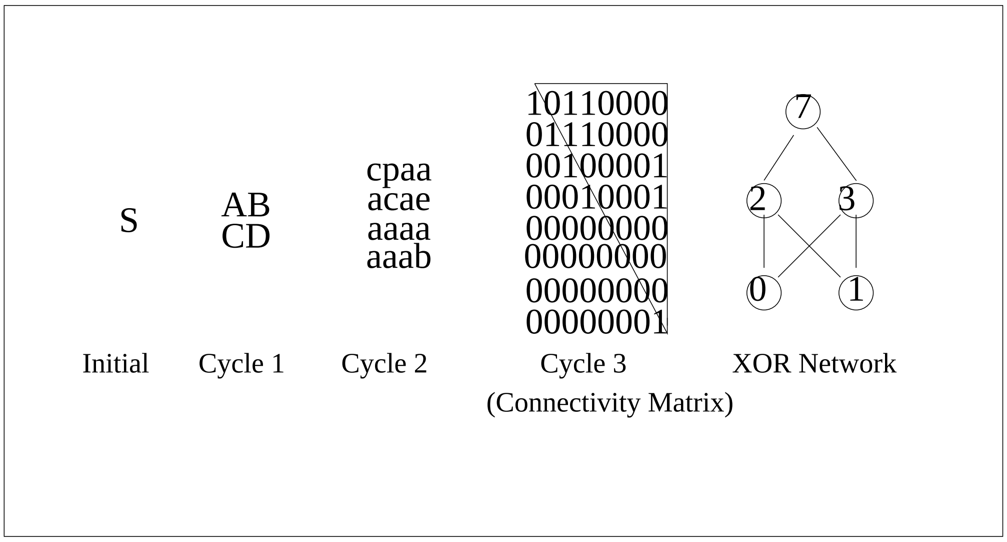

Figure 2: Generation of XOR graph (taken from [27].

each cycle.

```
       A B
S -->  C D

       c p          a a          a a           a a
A -->  a c   B --> a e    C --> a a     D --> a d

       0 0          0 0          1 0           0 1          1 1
a -->  0 0   b --> 0 1    c --> 0 0     e --> 0 1   p -->  1 1
```

For example, in the first cycle start symbol 'S' is re-written using the relevant rule. There after, for every symbol at the right hand side of first rule, the relevant rule is processed. At the end the connectivity matrix is developed which shows the existence of a connection between two units by a "1" and the non-existence of a connection with a "0".

A typical chromosome representing a network has two parts: a variable and a constant part. The constant part does not change and contains a set of static rules that are used to re-write symbols. The genetic algorithm is only applied to the variable part to acquire rules through a selection process. Since the constant part is not involved in the recombination and mutation processes, the length of the variable part constitutes the chromosome length. Each of these parts are divided into some fragments. Each fragment is made up of five bits representing a rule, in which the first bit represents the left-hand-side of a rule and the rest represent the right-hand-side of a rule. For example, the first rule

```
        AB
S -->   CD
```

(among the rules above) would be represented as follows:

S	A	B	C	D

In order to ensure that a cell division will always take place, the beginning of the chromosomes will always contain "S"; the initial state. The variable part contains symbol-generating rules in the range between "A" to "p" and the constant part contains pre-encoded re-write rules of symbols from "a" to "p".

The grammar encoding system is tested on XOR, 4-X-4 and 8-X-8 encoding problems using the back-propagation learning rule on feed forward networks.

The results of several experiments showed that the grammar encoding method converges much faster than direct encoding methods. Also it creates more regular network connections than direct encoding methods would normally do. This means that the grammar-encoding system shows a better scaling property and ability to generate more complex networks. Finally, it is biologically more plausible since the connectivity information is encoded in the chromosome in a more flexible manner.

Although with the grammar encoding method the same abstraction of the rules can be used repeatedly to produce the same patterns of connections in different parts of the network, Kitano's experiment did not involve developing a recurrent network.

The scalability in Kitano's work is characterized as the ability of the GA to find a larger network for a parameterised problem of larger size. This is tested in an encoder/decoder problem. In [13] Gruau argues that this is a typical case where the number of hidden units is larger than the number of input or output units. He suggests that it would be more challenging to solve a problem where the number of the hidden units is the logarithm of the number of the input neurons and suggests a theoretical rather than an experimental property of the scalability. Finally, it is not clear whether a grammar encoding can express every architecture.

3.2 Gruau's Cellular Encoding Method

Gruau [13] [14] [15] developed a *Cellular Encoding* (CE) system similar to grammar encoding, which instead of re-writing symbols, it re-writes the cells themselves. That is, rather than using a matrix, the development rules are applied to cells of the network. Each cell contains a copy of the chromosome. A chromosome consists of some alphanumeric symbols (i.e., program symbols) which provides instructions as to what kind of actions will take place in the process of development. The instructions contained in the chromosome may be interpreted differently by different cells. Depending on the information that a cell receives, it can divide, change some internal parameters and finally become a neuron.

In this system, the initial graph is a single non-terminal unit which contains default program symbols in its chromosome. When the instructions on the chromosome are carried out, it gives birth to other cells which through their own program symbols do the same. Eventually, each cell becomes a neuron creating the ANN structure.

There are different kinds of program symbols and corresponding instructions. For example a division program symbol is used to create two cells from one cell. This can be done in parallel (denoted by "P") or sequentially (denoted by "S"). In a typical sequential division the first child inherits the input links and connects to the second child with a weight of 1. The second child inherits the output link. In a parallel division both children inherit the input and output links from their parent cell. Another program symbol (i.e., value program symbol) is used to modify the structure of the connections. The plus (+) and minus (-) signs are used to set the value of the weights either to 1 or -1 respectively. Other symbols are used to pause the re-writing process (denoted by "W") or to stop the process (denoted by "E"), causing the cells generated to be neurons.

This strategy is biologically more plausible and efficient than matrices. The language used to describe network structures is more elegant and compact and suitable for the genetic algorithms. It also allows for coding of the weights. Various properties of this strategy have been formalised by Gruau in [14]. Some of these can be summarised as follows:

1. Completeness: any network can be encoded using the CE strategy.
2. Compactness: the ANN representations created by CE and manipulated by the GA are of minimal size. This ensures a reduction in the genetic search space.
3. Closure: the process of CE is closed under GA. It always produces meaningful structures, for either acyclic or recurrent neural networks, via the reproduction process.
4. Modularity: for larger decomposable networks, the code of the network is the concatenation of the codes of subnetworks. This result in formation of the building blocks which can be used in several different places in a typical ANN structure. It also implies more regular ANN structures.
5. Scalability: the complexity of the problem is not reflected in the representation schema. A family of ANNs can be encoded with a fixed size code.
6. Power of expression: the CE strategy can be used to encode both the architecture and the weights.

3.3 More Generative Methods

In [5] a combination of L-systems, production rules and the GA is used to design modular ANNs. The system uses L-systems as a basis for re-writing production rules which constitute string representation of the network topologies. Thus, a chromosome encodes an ANN structure in the form of production rules. The GA is used to evolve a population of these representations. The fitness of each population member is determined by the residual error after a certain amount of back-propagation training.

Another system, which is also inspired by the Kitano's work is presented by Voigt *et al.* in [16] [17]. In this approach a *stochastic* L-system is used. Although the basic algorithm involves a grammar-encoding schema similar to that of Kitano's, the production rules used are probability-dependent. This aspect has been shown to be useful in preventing the generation of large numbers of redundant production rules.

After the network structure is generated in the same way as it is in Kitano's system, the sub-networks are iteratively and randomly modified. Sub-networks are chosen in a probabilistic manner. This corresponds to an individual development process. The GA is applied to a population of individuals who have passed through this process. This strategy also uses feed-forward network structures with the back-propagation learning rule. The fitness criterion applied is interesting: it is determined by mixing learning-error, classification-error, number of training iterations, number of connections, and minimal and maximal path-length in the network structure.

Finally, another generative method presented in [45, 43, 44] uses emergent modelling to construct ANNs within an incremental, comprehensive and biologically plausible life cycle of development, plasticity, natural selection and genetic changes. The ANNs are represented by a set of production rules describing the local behaviors. They are organised hierarchically. At the lowest level are cells with their connections; next comes the individual with its behavior and, at the top level ,the environment encloses all. Similar to Gruau's approach, starting from a single cell, the system controls the division of the cells and the growth of the connections. The system works in a similar way to the knowledge based systems. This strategy can encode feed-forward networks as well as recurrent networks. However, the use of GAs for rule-generation and recombination is limited.

4 Other Designs

There are quite a number of works in which ANNs are created and evolved as part of specific research interest. These either adapt the design strategies mentioned in this paper or create their own. For example, in [7] and [50] an ANN was evolved to find better solutions to some control problems. Also, in [9] a flexible application was carried out in which a variable number of neurons (units) with arbitrary links among them could be evolved by using species adaptation genetic algorithm (SAGA)[19] [20] [21] [24]; This is an extended form of GA which uses variable lengths of genotypes. This approach is well suited to the generation of highly recurrent neural networks. Another example of designing recurrent nets can be found in [41].

Some reinforcement learning methods have also employed evolutionary methods with ANNs. For example in [49] a larger multi-layer network design is evolved and in [1] interaction between learning - as the adaptation of individual, and evolution - as the adaptation of population is observed. In fact, there are quite a number of researchers concentrating on the relationship between evolution and learning [4] [30] [22] [38] [2] [32] [3] [11] [39] [42]. In these works individual network structures (representing the learning) are evolved to optimise the adaptive behavior (hence, the structure).

In [33] and [34] the generation of regular networks is presented by using a recursive algorithm. However, the strategy uses simulated annealing instead of GA as a search technique.

Finally, in [37] Muhlenbein proposes a general framework for applying genetic Algorithms to neural networks. In his system, the GA is used indirectly in designing neural networks. It is applied to some sort of structures which encode neural networks.

4.1 Further Research Issues

When designing network architectures, almost all of the studies mentioned used back-propagation learning rule to train the network. It would be interesting to develop a representation which could involve alternative learning models rather than back-propagation. This is likely to increase the difficulties of determining structures and parameters. Major research issues in designing ANNs include the following:

1. How to develop an encoding schema that can specify both the structure and the learning rule. Such a design process would involve a combination of learning dynamics that could adjust the architecture and the weights. This might even lead to discovery of new connectionist algorithms and structure.

2. Adaptation of genetic operators to construct meaningful networks. The attributes of networks are many and varied and our relative interest in them may change from one application to another. This might require adaptation of the types and the nature of the GA's parameter values (i.e, the rate of crossover, mutation etc.) in designing network structures. However, in most of the systems, we would expect that careful studies of these parameters would be computationally impractical.

3. How to vary the fitness evaluation functions. The possible functions would involve the learning speed, accuracy and cost parameters such as size and complexity of the network rather than how successful the network is in learning the task in hand.

It seems that together with the above considerations the main direction of research in network design will be focussing on designing larger networks efficiently.

Finally, the researches which involve genetic-based design of ANNs as part of their particular research aim can be very helpful. They can serve as an empirical test for what is proposed by genetic-based ANN design researches. At the same time, they can provide new and empirical ideas for future development in the genetic based design researche.

References

[1] D.H. Ackley and M.S. Litman. Learning from natural selection in artificial environments. In *Second Artificial Life Conference*, 1990.

[2] D.H. Ackley and M.S. Litman. Interactions between learning and evolution. In Langton et al, editor, *Artificial Life II.* Addison-Wesley, 1992.

[3] R.K. Belew. Evolution, learning and culture: computational methaphors for adaptive algorithms. *Complex systems*, 4:11–49, 1990.

[4] R.K. Belew, J. McInerney, and N.N. Schraudolph. Evolving networks: using the genetic algorithms with connectionist learning. In Langton et al, editor, *Artificial Life II.* Santa Fe Institute, 1992.

[5] E.J.W. Boers, H. Kuiper, B. Happel, and I. G. Sprinkhuizen. Designing modular artificial neural networks. Technical Report 93-24, Dept. of Computer Science, Leiden University, The Netherlands, 1993.

[6] T. P. Caudel and C.P. Dolan. Parametric connectivity: training of constrained networks using genetic algorithms. In J.D. Schaffer, editor, *Proceedings of Third International Conference on Genetic Algorithms*, 1989.

[7] F. Cecconi and D. Parisi. Evolving organisms that can reach for objects. In Meyer et al., editor, *From animals to animats: simulation of adaptive behavior*, 1991.

[8] D.J. Chalmers. Evolution of learning: an experiment in genetic connectionism. In Touretzky et al, editor, *Connectionist Models.* Morgan Kaufmann, 1990.

[9] D. Cliff, I. Harvey, and P. Husbands. Incremental evolution of neural network architectures for adaptive behavior. Technical Report CSRP-256, University of Sussex, COGS, 1992.

[10] N. Dodd. Optimisation of network structure using genetic algorithms. In *International Neural Network Conference, INNC-90-Paris.* Kluwer, Dordrecht, 1990.

[11] J.F. Fontanari and R. Meir. The effect of learning on evolution of sexual populations. *Complex Systems*, 4:401–414, 1990.

[12] D. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, Massacheusettes, 1989.

[13] Frederic Gruau. The cellular encoding of genetic neural network. Technical Report Tr 92.21, Labaratoire de l'informatique du parallelisme, Ecole Normale Superieure de Lion, 1992.

[14] Frederic Gruau. Genetic synthesis of boolean networks with a cell re-writing developmental process. In D. Whitley and J.D. Schaffer, editors, *COGAN-92, Combination of Genetic Algorithms and Neural Network.* IEEE Computer Society Press, 1992.

[15] Frederic Gruau and D. Whitley. The cellular developmental of neural networks: the interaction of learning and evolution. Technical Report 93-04, Labaratoire de l'informatique du parallelisme, Ecole Normale Superieure de Lion, 1993.

[16] I. Santibanez-Koref H.-M. Voigt, J. Born. Evolutionary structuring of artificial neural network. Technical Report TR-02-93, Technical University Berlin, Bionics and Evolution Techniques Laboratory, 1993.

[17] I. Santibanez-Koref H.-M. Voigt, J. Born. Structuring of artficial neural networks with generative grammars. In *Seminar Neurainformatik 1992.* Berlin, springer-Verlag, 1993.

[18] S. A. Harp, T. Samad, and Aloke Guha. Toward a genetic synthesis of neural networks. In J.D. Schaffer, editor, *Proceedings of Third International Conference on Genetic Algorithms*, 1989.

[19] I. Harvey. Adding species adaptation genetic algorithms:a basis for a conitinuing saga. Technical Report CSRP-221, University of Sussex, COGS, 1992a.

[20] I. Harvey. Evolutionary robotics and saga: The case for hill crowling and tournement selection. Technical Report CSRP-222, University of Sussex, COGS, 1992b.

[21] I. Harvey. The saga cross:the mechanics of recombination for species with variable-length genotypes. Technical Report CSRP-223, University of Sussex, COGS, 1992c.

[22] G.E. Hinton and S. J. Nowlan. How learning can guide evolution. *Complex systems*, 1:495–502, 1987.

[23] J. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, USA, 1975.

[24] D. Cliff I. Harvey, P. Husbands. Issues in evolutionary robotics. Technical Report CSRP-219, University of Sussex, COGS, 1992.

[25] Koza J.R. and Rice J.P. Genetic generation of both the weights and the architecture for a neural network. pages 397–404, 1992.

[26] M. Kerzberg and A. Bergman. The evolution of data processing abilities in competing automata. In M. J. Rodney, editor, *Computer Simulation in Brain Science.* Cambridge University Press, 1988.

[27] Hiroaki Kitano. Designing neural networks using genetic algorithms with graph generation system. *Complex systems*, 4:461–76, 1990.

[28] A. Lindenmayer. Mathematical models in cellular interactions in development. *J. of Theoretical Biology*, 18:280–899, 1968.

[29] A. Lindenmayer. Developmental systems without cellular interactions, their language and grammars. *J. of Theoretical Biology*, 30:455–484, 1971.

[30] J. Maynard Smith. When learning guides evolution. *Nature*, 329:761–762, 1987.

[31] G.F. Miller, P. M. Todd, and S. U. Hegde. Designing neural networks using genetic algorithms. In J.D. Schaffer, editor, *Proceedings of Third International Conference on Genetic Algorithms*, 1989.

[32] G.F. Miller and P.M. Todd. Exploring adaptive agency i:theory and methods for simulating the evolution of learning. In Touretzky et al, editor, *Connectionist Models.* Morgan Kaufmann, 1990.

[33] E. Mjolness and D. H. Sharp. Preliminary analysis of recursively generated neural networks. In John Denker, editor, *Neural Networks for Computing.* American Institute for Physics, 1986.

[34] E. Mjolness, D. H. Sharp, and K. Alpert, Bradley. Recursively generated neural networks. *IEEE*, 3:165–171, 1987.

[35] D.J. Montana and L. Davis. Training feedforward neural network using genetic algorithm. Technical report, BBN Systems and Technologies Inc., 1989.

[36] H. Muhlenbein. Limitations of multi-layer perceptrons networks: step toward genetic neural networks. *Parallel Computing*, 14:249–260, 1990.

[37] H. Muhlenbein and J. Kinderman. The dynamics of evolution and learning-towards genetic neural networks. In R. Pfeifer et al, editor, *Connectionism in Perspective.* Elsevier Science Publishers, North-Holland, 1989.

[38] S. Nolfi, J.L. Elman, and D. Parisi. Learning and evolution in neural networks. Technical Report CRL 9019, Uni. of California, 1990.

[39] D. Parisi, S. Nolfi, and F. Cecconi. Learning behavior and evolution. In Varela and Bourgie, editors, *First European conference on Artificial Life*, 1992.

[40] J.D. Schaffer, D. Whitley, and L. Eshelman. Introduction. In D. Whitley and J.D. Schaffer, editors, *COGAN-92, Combination of Genetic Algorithms and Neural Network*. IEEE Computer Society Press, 1992.

[41] P. Spiessens and J. Toreek. Massively parallel evolution of recurrent networks: an approach to temporal processing. In Varela and Bourgie, editors, *First European conference on Artificial Life*, 1992.

[42] P. M. Todd and G. F. Miller. Exploring adaptive agency ii: simulating the evolution of asociative learning. In Meyer et al., editor, *From animals to animats: simulation of adaptive behavior*, 1991.

[43] Jari Vaario. *An Emergent Modeling Method for Artificial Neural Networks*. PhD thesis, The University of Tokyo, 1993.

[44] Jari Vaario and Setsuo Ohsuga. Adaptive neural architectures through growth control. In Cihan H. Dagli, Soudar R.T. Kumara, and Shin, editors, *Intelligent Engineering Systems through Artificial Neural Networks*, pages 11–16. ASM Press, New York, 1991.

[45] Jari Vaario and Setsuo Ohsuga. An emergent construction of adaptive neural architectures. *Heuristics - The Journal of Knowledge Engineering*, 5(2), 1992.

[46] D. Whitley and T. Hanson. Optimizing neural networks using faster, more accurate genetic search. In J.D. Schaffer, editor, *Proceedings of Third iInternational Conference on Genetic Algorithms*, 1989.

[47] D. Whitley and J.D. Schaffer. *COGAN-92, Combination of Genetic Algorithms and Neural Network*. IEEE Computer Society Press, 1992.

[48] D. Whitley, T. Starkweather, and C. Bogart. Genetic algorithms and neural networks: optimising connections and connectivity. *Parallel Computing*, 14:347–361, 1990.

[49] L.D. Whitley, S. Dominic, and R.Das. Genetic reinforcement learning with multilayer neural networks. In Belew and Booker, editors, *Proceedings of Fourth International Conference on Genetic Algorithms*, 1991.

[50] A.P. Wieland. Evolving controls for unstable systems. In Touretzky et al, editor, *Connectionist Models*. Morgan Kaufmann, 1990.