Evolving Neural Networks Using a Dual Representation with a Combined Crossover Operator

João Carlos Figueira Pujol, Riccardo Poli

Abstract—**In this paper a new approach to the evolution of neural networks is presented. A linear chromosome combined with a grid-based representation of the network, and a new crossover operator, allow the evolution of the architecture and the weights simultaneously. In our approach there is no need for a separate weight optimization procedure and networks with more than one type of activation function can be evolved. A pruning strategy is also introduced, which leads to the generation of solutions with varying degrees of complexity. Results of the application of the method to several binary classification problems are reported.**

I. Introduction

The reliable, general purpose, automatic design of neural networks (NNs) is still a largely unsolved problem. Constructive and destructive algorithms attempt to offer a solution to the problem, by beginning with a small network and adding new features as needed, or by starting from a large network and removing unnecessary elements, respectively [1], [2], [3]. However, both approaches constrain the architectures achieved, either from the beginning, or through the structural modifications they introduce.

Recently, new promising approaches based on evolutionary algorithms, such as evolutionary programming (EP) [4] and genetic algorithms (GAs) [5], have been applied to the development of artificial neural networks. However, the results obtained so far are limited by the lack of specialized operators to handle neural networks.

Approaches based on EP rely exclusively on mutation and operate on the neural network directly [6], [7], [8]. Although it has been advocated that EP is a more suitable approach to the development of neural networks, we think that crossover is an inportant mechanism provided by GAs for the exploitation of information on different regions of the search space.

Methods based on genetic algorithms usually represent the structure and the weights of NNs as a string of bits or as a combination of bits and real numbers [9], [10], [11], and perform the crossover operation as if the network were a linear structure. However, neural networks cannot naturally be represented as binary vectors. They are oriented graphs, whose nodes are neurons and whose arcs are synaptic connections. Therefore, it is arguable that any efficient approach to evolve NNs should use operators based on this structure.

The authors are with the School of Computer Science, University of Birmingham, Birmingham B15 2TT, UK. E-mail: {J.Pujol,R.Poli}@cs.bham.ac.uk

Some recent work based on genetic programming (GP) [12], originally developed to evolve computer programs, is a first step in this direction. For example, in [12], [13] neural networks have been represented as parse trees which are recombined using a crossover operator which swaps subtrees representing subnetworks. However, the graph-like structure of neural networks is not ideally represented directly with parse trees either. Indeed, an alternative approach based on GP, known as cellular encoding [14], has recognized this and used an indirect network representation in which parse trees represent rules which grow a complete network from a tiny neural embryo. Although very compact, this representation enforces a considerable bias on the network architectures that can be achieved.

Recently, a new form of GP, Parallel Distributed Genetic Programming (PDGP) [15], in which programs are represented as graphs instead of parse trees, has been applied to the evolution of neural networks [16]. The method allows the use of more than one activation function in the neural network but it does not include any operators specialized in handling the connection weights. Nonetheless the results were encouraging and led us to believe that a proper representation, with specialized operators and meaningful building blocks, could be used to efficiently evolve neural networks. Indeed, in [17] we improved and specialized PDGP by introducing a dual representation, where a linear description of the network was dynamically associated to a two-dimensional grid. Several specialized genetic operators, some using the linear description, others using the grid, allowed the evolution of the topology and the weights of moderately complex neural networks very efficiently. A pruning strategy was also introduced, which led to the generation of solutions with varying degrees of complexity.

In this work we further improve our previous method and propose a new combined crossover operator which allows the determination of the architecture, the activation function, and the weights of a neural network concurrently and very efficiently. In the following sections, our representation and operators are described, and the results of the application of this paradigm to binary classification problems are presented.

II. Representation

In PDGP, instead of the usual parse tree representation used in GP, a graph representation is used, where the functions and terminals are allocated in a two-dimensional grid of fixed size and shape. The grid is particularly useful to

solve problems whose solutions are graphs with a natural layered structure, like neural networks.

However, this representation can make inefficient use of the available memory if the grid has the same shape for all individuals in the population. In fact, in this case there is no need to represent the grid explicitly in each individual. Also, in some cases, it is important to be able to abstract from the physical arrangement of neurons into layers and to only consider the topological properties of the net. To do this, it is more natural to use a linear genotype.

These limitations led us to propose [17], [18] a dual representation, in which a linear chromosome is converted, when needed, into the grid-based PDGP representation. The dual representation includes every detail necessary to build and use the network: connectivity, weights and the activation function of each neuron. A chromosome is an ordered list of nodes (see Figure 1a). An index indicates the position of the nodes in the chromosome. All chromosomes have the same number of nodes.

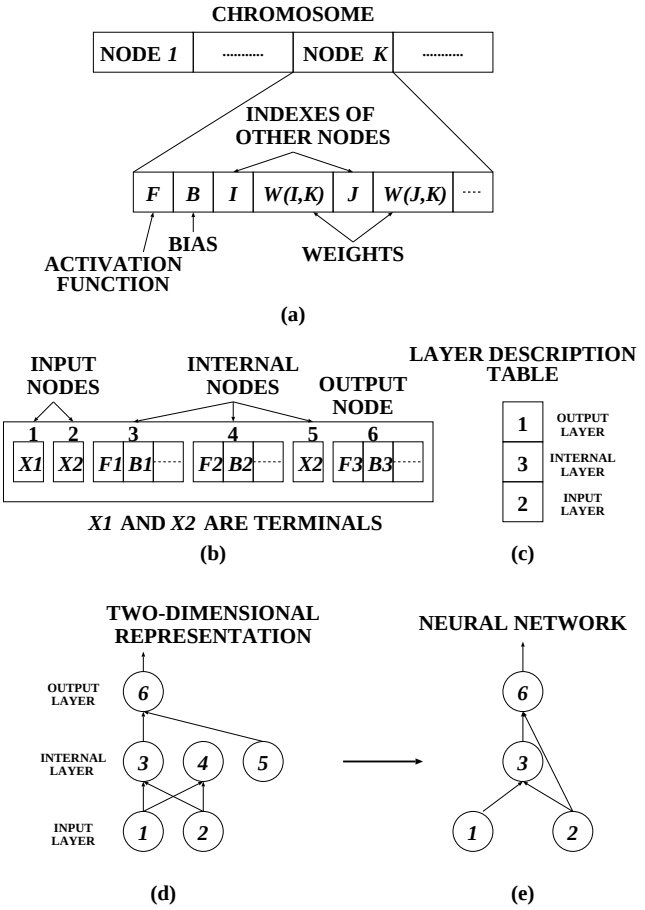

Fig. 1. (a) Chromosome and node description. (b) Example of a chromosome. (c) The grid description table describing the number of nodes per layer of the network. (d) The two-dimensional representation resulting from the mapping of the nodes of the chromosome in (b), according to the description table in (c) (note the intron in the internal layer). (e) Resulting neural network after elimination of introns and transfer of connection from terminals in the internal layer to corresponding terminals in the first layer (node 5 is the same terminal as node 2).

Like in standard genetic programming, the nodes are of two kinds: functions and terminals. The functions represent the neurons of the neural network, and the terminals are the variables containing the input to the network (see Figure 1b).

When the node is a neuron, it includes the activation function and the bias, as well as the indexes of other nodes sending signals to the neuron and the weights necessary for the computation of its output. Multiple connections from the same node are allowed. Nodes are evaluated according to their order in the chromosome.

When necessary (see section III), the linear representation just described is transformed into the layered representation used in PDGP. A description table with the same number of nodes as the chromosomes defines the number of layers and the number of nodes per layer of a two-dimensional representation (see Figure 1c and d). The nodes of the linear chromosome are mapped onto this representation according to the description table. The connections of the network are indicated by links between nodes in the two-dimensional representation. The description table is a characteristic of the population and it is not included in the genotype. The two-dimensional representation may have any number of layers, each of different size. This feature may be used to constrain the geometry of the network (e.g. to obtain encoder/decoder nets) and can also be used to guide specialized crossover operators [17].

The nodes in the first layer are terminals, whereas the nodes in the last layer represent the output neurons of the network. The number of input and output nodes depends on the problem to be solved. The remaining nodes, called *internal nodes*, constitute the internal layer(s), and they may be either a hidden neuron or a terminal.

Although the size of the chromosome is fixed for the entire population, the neural networks represented may have different sizes. This happens because terminals may be present as internal nodes from the beginning, or may be introduced by crossover and mutation (this is discussed in section III). They are removed from the network during the decoding phase, which is performed before each individual is evaluated. Connections from the removed terminals are replaced with connections from corresponding terminals in the input layer (see Figures 1d and e).

Our model allows the use of more than one activation function, so that a suitable combination of activation functions can be evolved to solve a particular problem (this is discussed in the next section).

III. Crossover

By experimenting with different combinations of crossover and mutation operators in our previous work [17], we have drawn the conclusion that, for the evolution of neural networks, it is important to use operators which induce a fitness landscape as smooth as possible, and that it is also important to treat connections from terminals differently from connections from functions (neurons).

The crossover operator proposed in this paper works by randomly selecting a node a in the first parent and a node b

in the second parent (see Figure 2a), and by replacing node a in a copy of the first parent (the offspring). Depending on the types of node a and node b, the replacement of node a is carried out as follows:

Both nodes are terminals This is the simplest case, node b replaces node a, and there is no change either in the topology or in the weights of the network.

Node b is a terminal and node a is a function In this case, node b also replaces node a, but the complexity of the network is reduced, because a neuron is removed from the network.

Node b is a function and node a is a terminal In this situation, the crossover operation increases the complexity of the network. A temporary node, c, is created as a copy of node b. Before node c replaces node a in the offspring, each of its connections is analyzed and possibly modified, depending on whether they are connections from terminals or functions.

- If the connection is from a function, the index of the connected node is not modified.
- If the connection is from a terminal, the index is modified to point to another node, as if the connection had been rigidly translated from node b to node a. For example, the connection between node 10 and node b in Figures 2a and b is transformed into a connection between node 8 and node c in Figures 2c and d. In some cases, translating the connection rigidly might lead to point to an unexistent node outside the limits of the layer. In this case, the index of the connected node is modified as if the connection had been wrapped around the layer. For instance, the connection between node 7 and node b in Figures 2a and b is transformed into a connection between node 1 and node c in Figures 2c and d. If the rigid translation of the connection requires the presence of a node below the input layer, the connection is deleted.

Sometimes, a connection in node c, modified or not, generates feedback loops in the offspring. In this case, as we are interested in feedforward networks, the connection is deleted. For example, the connection between node 13 and node b in Figure 2b was deleted when node c was created.

This procedure for connection inheritance aims at preserving as much as possible the information present in the connections and weights.

Both nodes are functions This is the most important case. By combining the description of two functions, the topology and the weights of the network can be changed. After creating node c as described above, its description and the description of node a are combined by selecting two random crossover points, one in each node, and by replacing the connections to the right of the crossover point in node a with those to the right of the crossover point in node c. Thus creating a new node to replace node a in the offspring. See Figure 3.

This process can easily create multiple connections between the same two nodes. These are very important because their net effect is a fine tuning of the connection strength between two nodes. However, as this may reduce the efficiency of the search, we only allow a prefixed maximum number of multiple connections. If more than the allowed maximum number of multiple connections are created, some of them are deleted before the replacement of node a in the offspring.

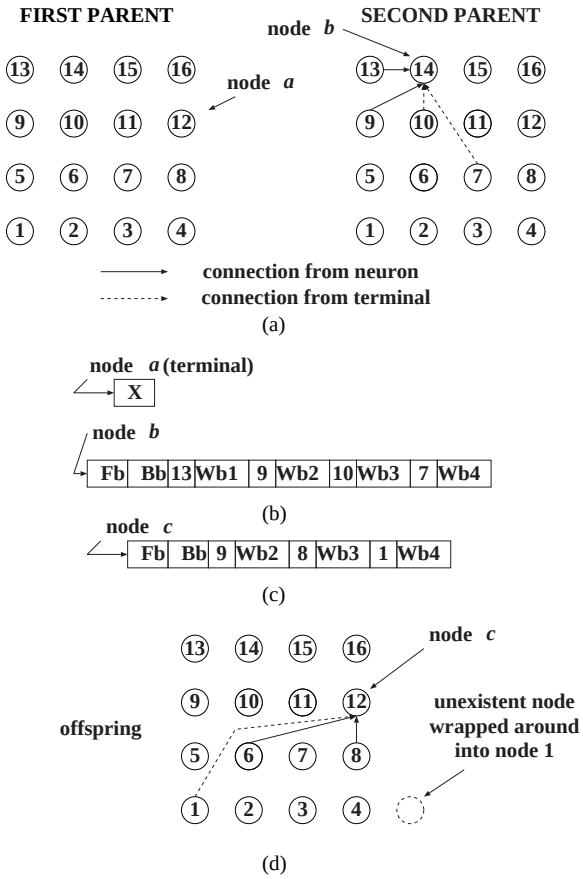

Fig. 2. (a) Two-dimensional representation of the parents. For clarity, only connections relevant to the operation are shown. (b) Nodes a and b. (c) Node c is a copy of node b with modified connections. The connections of node b whose indexes indicated connections from terminals received new indexes. The connection from node 13 was deleted. (d) Offspring generated by replacing node a with node c.

Modification of the activation function and bias of a node is not performed with our crossover operator. However, this can be indirectly accomplished by replacing a function with a terminal, which can then be replaced with a function with different features.

This crossover operator can not only evolve the topology and strength of the connections in a network, but also the number of neurons and the neurons themselves, by replacing their activation functions and biases. Similarly to GP, even if both parents are equal, the offspring may be different. This helps to keep diversity.

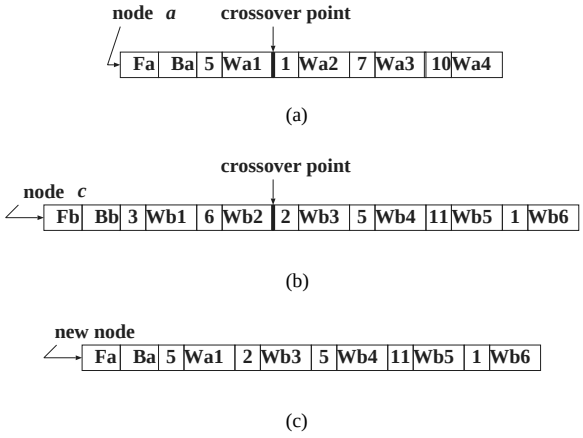

Fig. 3. Combination of two functions. (a) Node *a*. (b) Node *c* . (c) New node created to replace node *a* in the offspring. Note the multiple connections created.

IV. Mutation

Our representation allows the implementation of the whole set of mutation operators associated with evolutionary methods applied to neural networks: addition and deletion of connections, crossover of an individual with a randomly generated one, crossover of a randomly selected node with a randomly generated one, addition of Gaussian noise to the weights and biases, etc..

The deletion or addition of nodes is not allowed in the current form of the representation, as the size of the chromosome is constant. However, a function may be replaced with a terminal or vice-versa, and this may be used to reduce or increase the complexity of the network, within predefined limits.

In this work we have used a special form of mutation (pruning). The strategy is the following: after a 100% correct solution is found, a function in the internal layers of each individual in the population is replaced with a terminal, and the evolution process is resumed. This pruning procedure is repeated until a specified number of generations is achieved. This strategy has the advantage of generating solutions of varying degrees of complexity, allowing the user to decide which solution is preferable: an early but complex one, possibly presenting fault tolerance, or a late but parsimonious one, with probably more generalization power. No penalty term is included in the fitness function to enforce more parsimonious solutions.

V. Results

In all experiments a population of 200 individuals was evolved for a maximum of 500 generations. For each problem, 100 runs were performed with different random seeds. Unless otherwise stated, all individuals were initialized with 10 internal nodes in a single internal layer. The weights and biases were randomly initialized within the range [-1.0, +1.0]. We only used two mutation operators: crossover with a randomly created individual and pruning. A maximum of 5 multiple connections was allowed between each pair of nodes. We used a generational genetic algorithm with tournament selection (tournament size = 4). The crossover and mutation probabilities were 70% and 5%, respectively. The mean square error of the output of the network for all input patterns was used as fitness function.

In a first set of experiments, only a threshold activation function was used. They are discussed in section V.A. In a second set of experiments, we used both a threshold and a sigmoid activation function (see section V.B).

To show the performance of the method proposed, it was applied to a test suite of standard benchmarks present in the literature: the odd-2 (XOR), 3 and 4 parity problems, the 2-bit adder and the TC problem. In the 2-bit adder problem, the network is required to return the sum of two 2-bit input numbers. In the TC task, the network is required to identify the characters T and C represented by a 3×3 bit template, placed in any position and orientation within a 4×4 matrix.

A. One activation function

Table I shows the results of the experiments. Column 2 represents the average number of generations to obtain the last solution through the pruning strategy described. Column 3 and 4 show the minimum, average and maximum number of hidden neurons and of connections after applying the pruning strategy described, respectively. Column 5 shows the computational effort, i.e. the number of fitness evaluations, necessary to obtain a solution with 99% probability [12]. (Note that the values in the Table refer to the pruned solutions, larger solutions are obtained much earlier).

TABLE I
Summary of the results obtained in 100 independent runs with a single activation function.

Task	Gen	Neurons min/avg/max	Connections min/avg/max	Effort
XOR	27	1/1.1/4	5/5.3/11	18,400
3 parity	88	1/1.8/8	7/9.6/32	72,000
4 parity	292	2/4.8/9	13/27.0/53	282,600
Adder	382	4/6.8/10	25/44.6/69	1,000,000
TC	305	1/5.3/12	13/53.5/112	286,200

The results for the parity problems compare very favorably with those reported in the literature. For example, to solve the XOR problem the following numbers of generations to attain solutions have been reported: 90 [19], 513 [11], less than 100 with a minimal solution at generation 200 [20]. For the odd-3 parity problem, Yao and Shi [19] reported an average of 739 generations to get a solution.

The odd-4 parity solution reported by Zhang and Mühlenbein [21] was achieved in fewer generations (9), but a population of 1000 individuals was used, and they used training by a hillclimbing procedure at each generation. The resulting minimal network had 6 neurons in the hidden layer and 23 connections, to be contrasted to the minimal solution with 2 hidden neurons and 13 connections

obtained with our approach (a typical solution is shown in Figure 4). For the 2-bit adder task, a solution with 4 hidden neurons and 25 connections was obtained. Whitley *et al.* [22], evolving only the weights of the connections, report a solution with 4 hidden neurons and 29 connections for this problem.

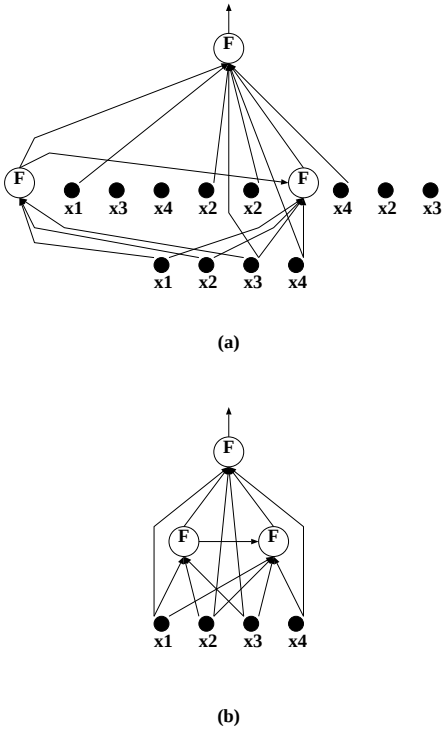

Fig. 4. Typical solution for the odd-4 parity problem. (a) Two-dimensional representation. (b) The corresponding network, after grouping of duplicate connections, removal of unused terminals and replacement of connections from terminals in the internal layer with connections from corresponding terminals in the input layer.

In the TC task, in order to compare our method with previous methods, the individuals were initialized with 16 nodes in the internal layer. Our method produced a minimal solution with a 11-1-1 topology and 13 connections (see Figure 5). This means that more than 96% of the 408 possible connections within the feedforward architecture of 16-16-1 were eliminated, and 5 unnecessary input neurons were removed. This testifies the efficiency of the pruning strategy. In comparison, Braun and Zagorski [23] report a minimal 11-1-1 architecture with 22 connections, and McDonnell and Waagen [24] *et al.* report a minimal 13-6-1 topology with 34 connections.

The generalization power of the minimal solution found for the TC problem was tested. This was performed inverting one bit of the 16 bits of each character, and presenting them to the minimal neural network. The network was still able to identify the noisy templates in 79% of the cases.

B. Two activation functions

In order to investigate the potential of our PDGP based approach, all tasks were rerun with a sigmoid and a threshold activation function. The results are reported in Table II. It was not possible to find similar results in the literature to compare with ours.

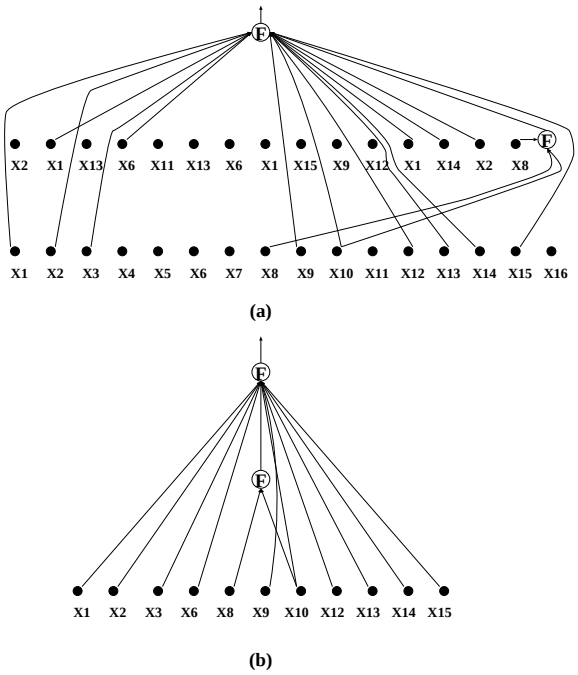

Fig. 5. Minimal solution for the TC task. (a) Two-dimensional representation. (b) The corresponding network, after grouping of duplicate connections, removal of unused terminals and replacement of connections from terminals in the internal layer with connections from corresponding terminals in the input layer.

TABLE II
SUMMARY OF RESULTS FOR 100 INDEPENDENT RUNS USING TWO ACTIVATION FUNCTIONS IN THE FUNCTION SET.

Task	Gen	Neurons min/avg/max	Connections min/avg/max	Effort
XOR	34	1/1.2/3	5/5.5/11	27,600
3 parity	99	1/2.5/10	7/12.0/35	94,800
4 parity	279	2/5.3/9	14/28.6/52	601,200
Adder	364	4/7.2/10	27/43.8/62	3,311,600
TC	309	1/5.6/10	15/55.5/101	469,000

In the beginning, all chromosomes were randomly initialized with both activation functions. The results are slightly worse than those obtained with only the threshold activation function. This is due to the fact that sigmoid functions are not particularly suitable to solve Boolean classification problems.

Starting with a distribution of 50% for both activation functions in the population, in 500 generations our method increased the presence of the threshold activation function in the population to 79% in the XOR problem, and to 70% in the odd-3 parity problem. This testify the ability of our method to select a suitable activation function for a task.

VI. Conclusion

In this paper, a new approach to the automatic design of neural networks has been presented, which makes natural use of their graph structure. The approach is based on a dual representation, a new crossover operator and a pruning procedure. The method was applied to the design of feedforward networks in a variety of problems showing promising results. In future research we intend to extend the power of the representation, and to apply it to a wider range of practical problems.

Firstly, with the exception of the crossover with a randomly created chromosome, no other mutation operator has been used in the present study. To keep diversity within the population we want to explore the use of other mutation operators. Secondly, we want to extend the method to recurrent neural networks, allowing it to be applied to a wide range of tasks. Thirdly, we want to allow the evolution of the size of the two-dimensional representation.

Promising preliminary results have already been attained in some of these areas.

Acknowledgements

The authors wish to thank the members of the EEBIC (Evolutionary and Emergent Behavior Intelligence and Computation) group for useful discussions and comments. This research is partially supported by a grant under the British Council-MURST/CRUI agreement and by CNPq (Brazil).

References

[1] R. Reed, "Pruning algorithms: a survey," *IEEE Transactions on Neural Networks*, vol. 4, no. 5, pp. 740–747, 1993.

[2] S. E. Fahlman and C. Lebiere, "The cascade-correlation learning architecture," in *Advances in Neural Information Processing Systems* (D. Touretzky, ed.), vol. 2, pp. 524–532, Morgan Kaufmann, 1990.

[3] C. Campbell and C. Vicente, "The target switch algorithm: a constructive learning procedure for feed-forward neural networks," *Neural Computation*, vol. 7, pp. 1245–1264, 1995.

[4] T. Bäck, G. Rudolph, and H. Schwefel, "Evolutionary programming and evolution strategies: Similarities and differences," in *Proceedings of the Second Annual Conference on Evolutionary Programming*, pp. 11–22, Evolutionary Programming Society, 1993.

[5] D. Goldberg, *Genetic algorithm in search, optimization and machine learning*. Reading, Massachusets: Addison-Wesley, 1989.

[6] P. J. Angeline, G. M. Saunders, and J. B. Pollack, "An evolutionary algorithm that constructs recurrent neural networks," *IEEE Transactions on Neural Networks*, vol. 5, no. 1, 1994.

[7] J. McDonnell and D. Waagen, "Neural network structure design by evolutionary programming," in *Proceedings of the Sec. Annual Conference on Evolutionary Programming* (D. Fogel and W. Atmar, eds.), (La Jolla, CA, USA), pp. 79–89, Evolutionary Programming Society, Feb. 1993.

[8] D. Fogel, "Using evolutionary programming to create neural networks that are capable of playing Tic Tac Toe," in *IEEE International Conference on Neural Networks (ICNN)*, pp. 875–880, IEEE Press, 1993.

[9] V. Maniezzo, "Genetic evolution of the topology and weight distribution of neural networks," *IEEE Transactions on Neural Networks*, vol. 5, no. 1, pp. 39–53, 1994.

[10] D. Whitley, S. Dominic, R. Das, and C. Anderson, "Genetic reinforcement learning for neurocontrol problems," *Machine Learning*, vol. 13, pp. 259–284, 1993.

[11] K. Tang, C. Chan, K. Man, and S. Kwong, "Genetic structure for NN topology and weights optimization," in *Proceedings of the International Conference on Genetic Algorithms in Engineering Systems: innovations and applications (GALESIA)*, pp. 250–255, Sept. 1995.

[12] J. R. Koza, *Genetic Programming, on the Programming of Computers by Means of Natural Selection*. Cambridge, Massachusets: The MIT Press, 1992.

[13] B. Zhang and H. Muehlenbein, "Genetic programming of minimal neural nets using Occam's razor," in *Proceedings of the 5th international conference on genetic algorithms (ICGA'93)* (S. Forrest, ed.), pp. 342–349, Morgan Kaufmann, 1993.

[14] F. Gruau, *Neural network synthesis using cellular encoding and the genetic algorithm*. PhD thesis, Laboratoire de L'informatique du Parallélisme, Ecole Normale Supériere de Lyon, Lyon, France, 1994.

[15] R. Poli, "Some steps towards a form of parallel distributed genetic programming," in *Proceedings of the First On-line Workshop on Soft Computing*, pp. 290–295, Aug. 1996.

[16] R. Poli, "Discovery of symbolic, neuron-symbolic and neural networks with parallel distributed genetic programming," in *3rd International Conference on Artificial Neural Networks and Genetic Algorithms (ICANNGA)*, 1997.

[17] J. C. F. Pujol and R. Poli, "Evolution of the topology and the weights of neural networks using genetic programming with a dual representation," Technical report CSRP-97-07, The University of Birmingham, School of Computer Science, 1997.

[18] J. C. F. Pujol and R. Poli, "A new combined crossover operator to evolve the topology and the weights of neural networks using a dual representation," Technical report CSRP-97-12, The University of Birmingham, School of Computer Science, 1997.

[19] X. Yao and Y. Shi, "A preliminary study on designing artificial neural networks using co-evolution," in *Proceedings of the IEEE Singapore International Conference on Intelligent Control and Instrumentation*, pp. 149–154, Jun. 1995.

[20] D. Dasgupta and D. McGregor, "Designing application-specific neural networks using the structured genetic algorithm," in *Proceedings of International Workshop on Combinations of Genetic Algorithms and Neural Networks (COGANN)* (L. Whitley and J. Schaffer, eds.), pp. 87–96, IEEE Computer Society Press, Jun. 1992.

[21] B. T. Zhang and H. Mühlenbein, "Evolving optimal neural networks using genetic algorithms with Occam's razor," *Complex Systems*, vol. 7, no. 3, pp. 199–220, 1993.

[22] D. Whitley, T. Starkweather, and C. Bogart, "Genetic algorithms and neural networks: Optimizing connections and connectivity," *Parallel Computing*, vol. 14-3, pp. 347–361, 1990.

[23] H. Braun and P. Zagorski, "ENZO-M - a hybrid approach for optmizing neural networks by evolution and learning," in *Parallel Problem Solving from Nature (PPSN3)* (Y. Davidor, H. Schwefel, and H. Manner, eds.), vol. 866, Springer-Verlag, 1994. Lecture Notes in Computer Science.

[24] J. McDonnell and D. Waagen, "Evolving neural network connectivity," in *Proceedings of IEEE International Conference on Neural Networks (ICNN)*, (San francisco, CA, USA), pp. 863–868, 1993.