??, ??, 1–17 (??)
© ?? Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

Evolving the Topology and the Weights of Neural Networks Using a Dual Representation

JOÃO CARLOS FIGUEIRA PUJOL AND RICCARDO POLI
School of Computer Science, The University of Birmingham, Birmingham B15 2TT, UK
{J.Pujol,R.Poli}@cs.bham.ac.uk

Received ??; Revised ??

Editors: ??

Abstract. Evolutionary computation is a class of global search techniques based on the learning process of a population of potential solutions to a given problem, that has been successfully applied to a variety of problems. In this paper a new approach to the construction of neural networks based on evolutionary computation is presented. A linear chromosome combined to a graph representation of the network are used by genetic operators, which allow the evolution of the architecture and the weights simultaneously without the need of local weight optimization. This paper describes the approach, the operators and reports results of the application of this technique to several binary classification problems.

Keywords: Evolutionary computation, genetic algorithms, neural networks, genetic programming

1. Introduction

The design of neural networks is still largely performed using a lengthy process of trial and error definition of the topology, followed by the application of a learning algorithm such as backpropagation [1]. The literature describes some approaches which try to automatically determine the topology and the weights of a network simultaneously. A strategy to do this is, for example, to delete or add structural elements (neurons and connections) to an existent unsatisfactory architecture. Destructive approaches (see for example [2]) begin with a trained architecture richer than necessary and prune unimportant elements of the network, whereas constructive approaches [3, 4, 5, 6] begin with a small topology and add new elements as need arises during the training process. The disadvantage of these strategies is that they constrain the architecture of the networks evolved, either from the beginning, or through the structural modifications they introduce. For example, Fahlman [4] adds hidden neurons fully connected to existing ones in a feedforward architecture, Frean [3] uses threshold activation functions and adds new hidden neurons fully connected to the input layer, and Campbell *et al.* [6] use binary weights.

Recently, new promising approaches based on evolutionary computation have been proposed for the development of artificial neural networks.

Evolutionary computation is a class of global search techniques based on the learning process of a population of individuals [7, 8, 9, 10, 11, 12]. Each individual represents a potential solution to a given problem. Typical evolutionary algorithms update this population seeking for better regions of the search space using operations of selection, recombination and mutation, inspired by biological evolution.

Many methods perform separately the building of the architecture and the learning of the weights [13, 14, 15, 16, 17]. Although biologically plausible, this approach is very inefficient. Other methods include partial training by gradient descent or simulated annealing in the evolution process [18, 19], and transfer the learned weights to the offspring in a Lamarckian fashion, in order to speed up evolution. The problem with these methods is that the performance of a partially trained network with a good architecture and bad weights may be worse than the performance of a network with a bad architecture but a good initial set of weights. This can mislead the evolutionary process.

Other methods try to develop the weights and the architecture concurrently, but impose restrictions on the value of the weights [20, 21, 22, 23]. This problem has been reduced using evolutionary programming techniques [24, 19, 18, 25, 26], which rely exclusively on mutation to adjust the weights and the architecture at each generation. However, this approach misses the benefits of the exchange of genetic material [9].

An alternative approach to the development of neural networks is to use genetic algorithms [27, 9] and genetic programming(GP) [28]. They are a class of evolutionary algorithms which make large use of crossover as an evolution operator. Genetic programming, originally developed to evolve computer programs, uses parse trees to represent neural networks. [28, 29, 30].

Methods based on GP. although promising, have been largely limited by the lack of good approaches to evolve the weights and the fact that parse trees are not a natural representation for neural networks. Indirect methods such as cellular encoding [21, 31] avoid the problem of representing the network directly as a parse tree, by representing the rules to construct the network instead. However, there are still constraints on the weights evolved, and to construct the network using the rules at each generation can be a considerable overhead.

Recently, a new form of GP, Parallel Distributed Genetic Programming (PDGP), in which programs are represented as graphs, has been applied to the development of neural networks with promising results [32, 33]. In PDGP, a two-dimensional grid is associated with the network to represent its layers. The genetic operators are specialized to act on this grid. However, no operators specialized in handling the connection weights were defined in PDGP, which also presented some representational biases, limiting its applicability to neural networks.

In this paper, we build upon the preliminary work done with PDGP, by introducing a dual representation and new operators. The graph representation of the network based on a two-dimensional grid is not static, but created dynamically on demand by some of the genetic operators, whereas other operators use a linear representation of the network. As this is more compact, we use it as the main representation. All operators are specialized to handle neural networks.

The new approach introduces interesting possibilities to be explored, and has been successfully applied to the determination of the architecture and the weights of feedforward networks. In the following sections, the representation and the operators are described, and the results of the application of this paradigm to binary classification problems are presented.

2. Representation

In PDGP, instead of the usual parse tree representation, a graph representation was used, where nodes are allocated in a two-dimensional grid of fixed size and shape which forms the chromosome. The grid is particularly useful to evolve solutions, like neural networks, which are graphs with a natural layered structure, as it allows the definition of genetic operators which preserve desired properties of the network. However, there is no need to keep the grid in the chromosome, as it is the same for all individuals of the population. It is more natural to use a linear genotype and a separate grid description.

The representation includes every detail necessary to build and use the network: the connectivity, the weights and the activation function of each node. All chromosomes in the population have the same number of nodes. Like in standard genetic programming, there is a set of functions, the admissible activation functions, and a set of

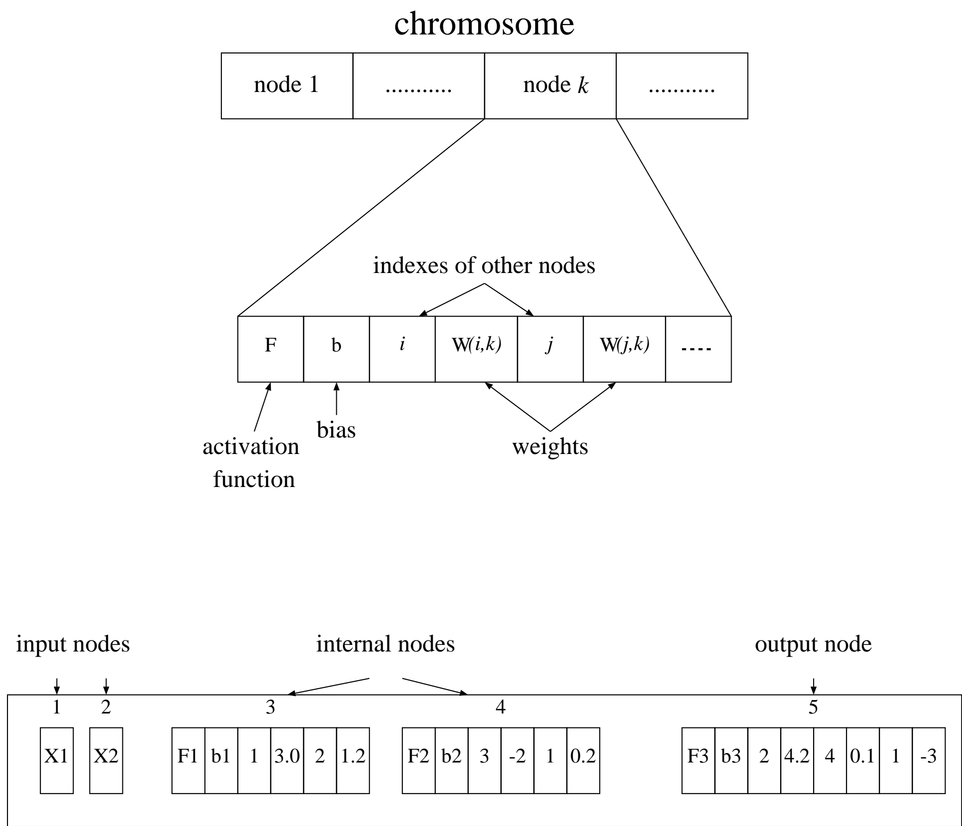


Fig. 1. Function node description (top) and an example of chromosome with 5 nodes (bottom).

terminals, the variables containing the input to the network.

The genotype is represented by a list of nodes. Each node may contain either a terminal or a neuron. In the first case, the output of the node is the value of the terminal. In the second case, the node represents a processing element of the encoded network, it not only contains the activation function, but also the connections and the weights (see Figure 1), to allow the computation of its output.

An index indicates the position of the nodes in the chromosome. As a rule, the first nodes must contain the terminals responsible for the input to the network, whereas the last ones must contain the neurons which return the output of the network, and their number depends on the problem to be solved. The remaining nodes are internal. Internal nodes may contain either a neuron or a terminal. In the first case, the node represents a hidden neuron of the encoded network. In the second case, the node represents an input to the network, introduced either initially or by the genetic operators, as described in the next section. For example, Figure 2 shows a network with two inputs, one hidden neuron and one output neuron, a common topology for the solution of the XOR problem. In the representation, there are 2 internal nodes, one containing a neuron and another containing a terminal. During decoding of the genotype, the internal node containing the terminal was removed, and the connection from this node was replaced with a connection from the node containing the corresponding terminal in the input layer, resulting in a four-neuron network.

In standard genetic programming the size of the chromosomes (parse trees) may grow exces-

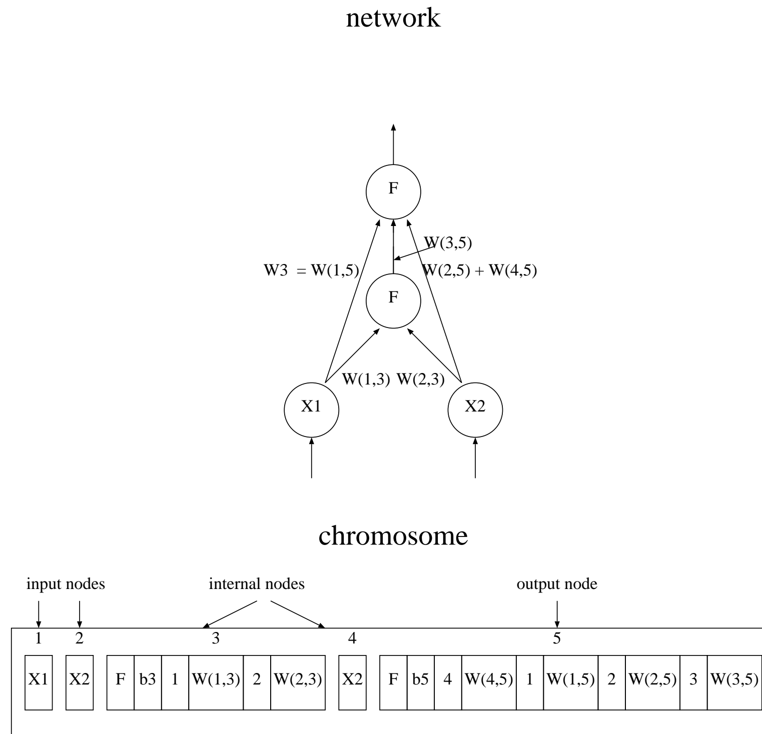


Fig. 2. Network to solve the XOR problem (top) and the corresponding chromosome (bottom).

sively. The linear representation avoids this problem constraining all chromosomes to having the same number of nodes. This limits the maximum size of the neural networks represented. However, as terminals may be present as internal nodes, the networks are not necessarily of the same size.

The order of the connection declarations within a node description is irrelevant, and multiple connections from the same node are allowed (this feature is used to adjust the weights, as described later).

The linear chromosome may be interpreted as a two-dimensional grid, in which each node occupies a cell. The grid has the same number of nodes as the chromosomes. The structure of the grid is defined by a description table, which determines the number of nodes in each layer. For example, the grid description table in Figure 3a is used to interpret the chromosome in Figure 3b as the two-dimensional structure in Figure 3c. Note that the grid is not necessarily rectangular, and it may have any number of layers, each of different size. This may be used to constrain the geometric form of the network (e.g. to evolve encoder/decoders).

The layered structure of the chromosome is used when the crossover operators are applied, allowing the application of constraints to the operators, to preserve desirable properties of the network. For example, a population of networks may be initialized with no connections within the same layer or between non-adjacent layers, and the grid may be used to guide the operators to comply with this pattern of connectivity if desired. However, it must be pointed out that the structure of the grid does not constrain the connectivity of the chromosomes in the initial population, connections between non-adjacent layers and within the same layer are allowed, unless explicitly stated.

The grid-based representation is very similar to the phenotype (compare Figures 3c and d), but more efficient to handle during crossover, because of its regularity.

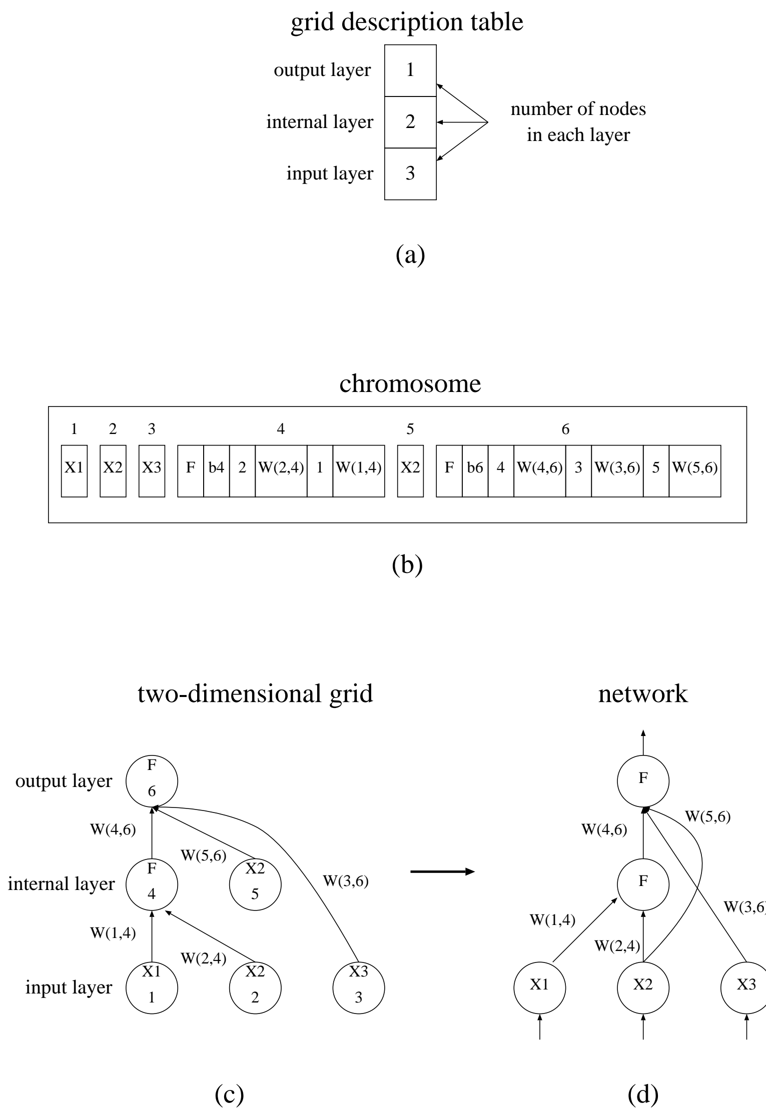


Fig. 3. (a) A grid description table. (b) A linear chromosome. (c) The two-dimensional grid structure of the chromosome as defined by the grid description table. (d) The decoded neural network (for simplicity, the biases were not indicated in the grid and in the network).

The current implementation is restricted to feedforward networks, and the genetic operators were designed to comply with this constraint. However, the extension to recurrent neural networks is straightforward, and work is already in progress to explore the potential of the representation in evolving recurrent networks.

The dual (linear and two-dimensional) representation just described gives us a large flexibility in the definition of the genetic operators within the framework of the new paradigm. A few of them are discussed in the next section.

3. Operators

3.1. Crossover

Crossover in genetic programming is performed by swapping subtrees between parents. In our representation, this operation may be performed swapping sub-graphs or node description between parents. Let us suppose that *node1* and *node2* are randomly selected nodes in the first parent and in the second parent, respectively. The operators are as follows:

- *Layered sub-graph crossover.* A randomly selected sub-graph from the second parent is transferred into the first parent using the two-dimensional grid. First, a sub-graph is defined by all the nodes connected to *node2* backward in the chromosome, and *node2* is transferred into the first parent at the position of *node1*. Note that the complete node is transferred, not only the weights. This means a node containing a terminal may replace a node containing a neuron, and vice-versa. In the grid, *node1* and *node2* define a vertical and a horizontal displacement. The transfer of the other nodes of the sub-graph to their new positions in the first parent is performed according to such displacements (see Figure 4a). If the vertical displacement of a node results in a position below the first layer of the grid (an infeasible node), the node and its connections are removed from the subgraph and are not transferred. If the horizontal displacement results in a position outside the limits of the corresponding layer, the position of the node is wrapped around. This operation preserves the structure of the network as defined initially, in the sense that if there are no connections between non-adjacent layers or within a layer before the operation, there will be none after the operation either (see Figure 4a). The only exception is when a terminal replaces a neuron, creating a connection to the input layer in the phenotype. This crossover operator is similar to the *SAAN* crossover in PDGP [32].
- *Linear sub-graph crossover.* First a linear displacement is defined by the difference between the positions of *node1* and *node2* within the linear chromosome. Secondly, *node2* is transferred into the first parent at the position of *node1*. Subsequently, the units connected to *node2* backward in the chromosome are transferred to their new positions in the first parent according to the linear displacement. Connections corresponding to node positions before the first unit are eliminated (see Figure 4b). This operation may create connections within the same layer.
- *Node transfer crossover.* Instead of transferring the sub-graph defined by the nodes connected to *node2*, only *node2* is transferred. As in the previous operations, infeasible nodes and connections are deleted.
- *One-point crossover.* Another possibility is to take advantage of the linear structure of the chromosome and perform one-point crossover as in a standard genetic algorithm, by randomly cutting the parents in a common point and swapping the left-hand sides to produce the offspring, with no regard to the connectivity.

- *Connection preserving crossover.* This can be used with any of the aforementioned crossover operators. When a node is transferred, infeasible nodes and connections are kept in their original place. This operation does not preserve the initial structure of the network, but may be useful to preserve the information contained in the weights of infeasible connections, which would be lost otherwise.
- *Node description crossover.* In this case, *node1* and *node2* are cut at random points and an offspring is generated connecting the first part of *node1* to the second part of *node2* or vice-versa. This is a way not only of introducing new connections into a node, but also of adjusting the values of the weights, as multiple connections between two nodes may be created. After the application of the operator, multiple weights are added and multiple links grouped back into a unique connection (see Figure 5).

The idea behind linear and sub-graph crossover is that connected sub-graphs are functional units (building blocks) whose output is used by other

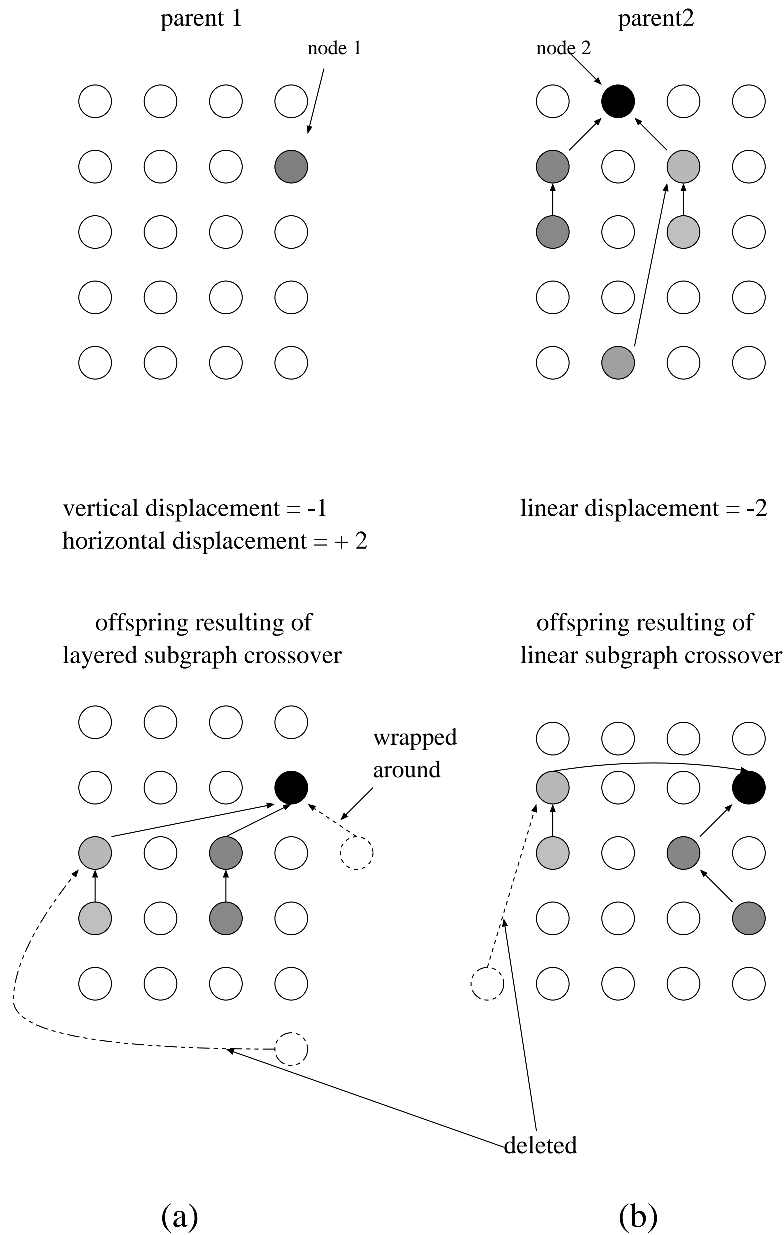


Fig. 4. Crossover by sub-graph transfer. The relevant connections and nodes are indicated with different patterns to make easier to track them to their new positions. Dotted nodes and connections exceed the limits of the grid. In the case of layered crossover (a), one node and the related connection were wrapped around, while another node (and connection) was eliminated. In the case of linear subgraph crossover (b), only one node/connection was deleted.

functional units. Therefore, by replacing a subgraph by another, different ways of combining building blocks are explored. The other crossover operators should be seen as methods to change the features of the building blocks, and to fine tune them.

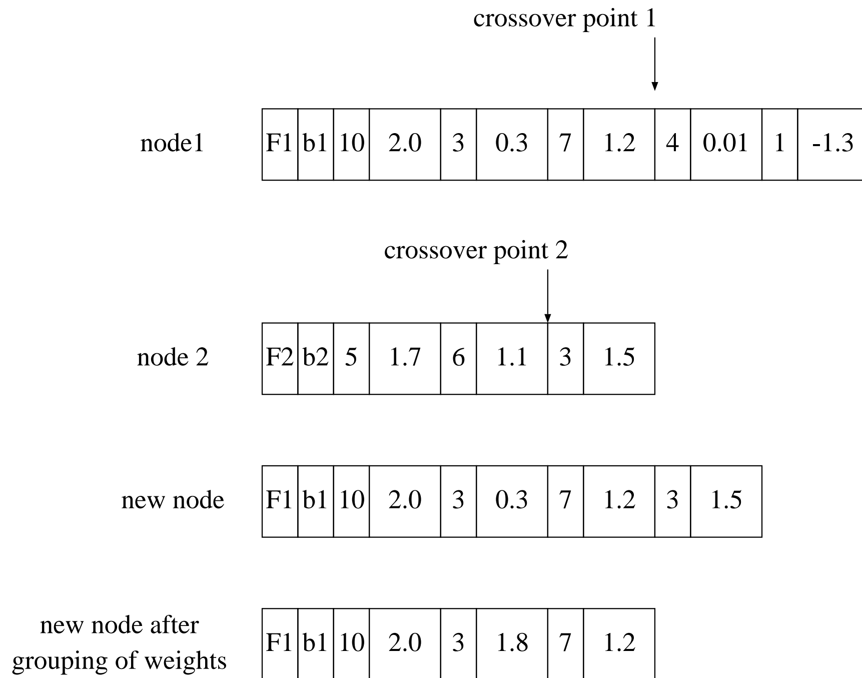


Fig. 5. Node description crossover creates new connections and adjusts weights.

3.2. *Mutation*

Our dual representation also 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 a chromosome 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 our representation, as the size of the chromosomes is constant. However, a hidden neuron 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.

4. **Pruning the network**

Crossover and mutation alone can, in principle, generate solutions with varying degrees of complexity. However, unless one includes a term in the fitness function to penalize the complex networks, there is no guarantee that parsimonious solutions will be achieved. Penalty terms usually depend on arbitrary coefficients which must be given beforehand. We preferred to adopt a different procedure, which does not involve any arbitrary parameter or additional computation. In order to find solutions with minimum complexity, the following strategy is applied in our runs: after a 100% correct solution is found, we replace one hidden node of each chromosome in the population with a terminal, and the evolution process is resumed. This pruning procedure is repeated until a specified number of generations is achieved. This strategy generates solutions of varying degrees of complexity, allowing the user to decide which solution is preferable: a complex one, possibly presenting fault tolerance, or a parsimonious one, with probably more generalization power. Actually, this procedure may be viewed as a special form of mutation to be applied to the population as a whole at each generation with a time varying probability of 1, if a new solution is found, of 0 otherwise.

5. Results

Unless otherwise stated, in all experiments a population of 200 chromosomes was evolved for a maximum of 200 generations. For each problem, we performed 100 runs with different random seeds. In the experiments, we used a generational genetic algorithm with: tournament selection (tournament size = 4) and crossover and mutation probabilities of 70% and 5%, respectively.

A three-layer grid with 10 nodes in the internal layer was used, and the chromosomes were initialized with hidden neurons only (no terminals were present in the internal layer initially) and random connections. The weights and biases were randomly initialized within the range [-1.0,+1.0]. It must be pointed out that during evolution, the weights are not restricted to this range anymore, as they are adjusted by *node description crossover*.

The only mutation operator used was the addition of zero mean Gaussian noise (with standard deviation 0.05) to the weights. The fitness function was the standard mean square error of the output of the network for all input patterns. The function set included only a threshold activation function.

The method was applied to the following standard benchmark problems present in the literature:

- N-bit odd parity problems (for N=2 (XOR), 3, 4 and 5), where the network output must be 1 if there is an odd number of 1's in the input pattern, and 0 otherwise.
- TC recognition problem, where 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 bit matrix.
- 2-bit adder problem, where the network has to compute the sum of 2-bit numbers.

In a first set of experiments *node description crossover* was used in conjunction with each of the other crossover operators with a relative probability of 1 to 3. The *node description crossover* was used to fine tune the weights, whereas the other operators were responsible for determining the architecture of the network.

Tables 1a–d summarize the results obtained. Column 2 represents the average number of generations at which the last solutions were produced by the pruning process described. Column 3 and 4 show the number of hidden neurons and the number of connections of such solutions, respectively. Column 5 shows the computational effort, i. e. the number of fitness evaluations, necessary to obtain a solution with 99% probability [28], and column 6 shows the percentage of runs in which solutions were found.

Apparently, *node transfer crossover* (Table 1a) gives the best results in terms of effort and percentage of solutions. However, the results obtained with *layered subgraph crossover* (Table 1b) are really very close to those in Table 1a, with the exception of the computational effort of the 5-parity problem. We believe that this result is a statistical artifact, and that the two crossover operators are essentially equivalent.

The grid representation of a typical solution for the XOR problem and the corresponding network is shown in Figure 6a. The 11th node without output connections and the disconnected nodes in the internal layer are introns, which are eliminated by the interpreter when decoding the chromosome. The interpreter also replaces connections departing from terminals in the internal layer with connections departing from terminals in the first layer. Typical results for the 3 and 4-parity problems are shown in Figures 6b and c, respectively.

Results for the XOR, 3 and 4-parity problems reported in the literature are shown in Tables 2a–c, respectively. To compare with, results of comparable complexity obtained with the dual representation are also shown (these results are from the same runs as those shown in Table 1a, but selected before the end of the pruning procedure). Although some parameters are not available, our results compare very favorably with those found in the literature, e.g. in terms of number of generations to achieve solutions of equivalent complexity.[1]

The scalability of the method can be assessed by the effort shown in Table 1a. Efforts for the odd 3, 4 and 5-parity problems are comparable to those reported using other methods. For example, Koza [28] reported an effort of 80,000 for even 3-parity, 912,000 for odd 4-parity, and 7,840,000 for even

Table 1. Summary of odd parity and TC results using: (a) *node transfer crossover* and *node description crossover* with relative probability 3 and 1, respectively; (b) *layered sub-graph crossover* and *node description crossover* with relative probability 3 and 1, respectively; (c) *linear sub-graph crossover* and *node description crossover* with relative probability 3 and 1, respectively; (d) *one-point crossover* and *node description crossover* with relative probability 3 and 1, respectively.

(a)

task	generations	hidden neurons min/avg/max	connections min/avg/max	effort	solutions found
XOR	66.0	1/1.2/3	5/5.5/11	40,200	100%
3-parity	95.0	1/2.2/9	7/10.7/35	72,000	98%
4-parity	136.3	2/5.2/9	14/26.5/42	720,000	23%
5-parity	149.7	6/6.7/7	37/37.3/38	5,350,400	3%
TC	129.7	1/3.7/6	14/36.1/62	554,400	29%

(b)

task	generations	hidden neurons min/avg/max	connections min/avg/max	effort	solutions found
XOR	67.0	1/1.2/4	5/5.5/11	39,800	100%
3-parity	103.1	1/2.0/10	7/10.3/31	78,800	93%
4-parity	121.2	2/5.3/8	14/26.1/37	884,400	19%
5-parity	179.0	5/5.0/5	33/33.0/33	16,524,000	1%
TC	136.9	2/4.7/7	23/39.3/55	603,000	27%

(c)

task	generations	hidden neurons min/avg/max	connections min/avg/max	effort	solutions found
XOR	69.5	1/1.5/6	5/6.2/18	40,000	100%
3-parity	105.6	1/2.0/6	7/10.0/24	152,000	75%
4-parity	112.5	5/5.0/5	22/25.0/28	5,472,000	2%
5-parity	N/A	N/A	N/A	N/A	0%
TC	142.1	1/3.8/8	15/33.8/58	697,200	22%

(d)

task	generations	hidden neurons min/avg/max	connections min/avg/max	effort	solutions found
XOR	48.1	1/1.8/8	5/7.0/24	23,800	100%
3-parity	69.3	1/3.3/10	7/13.6/36	141,000	52%
4-parity	130.0	6/6.0/6	29/29.0/29	12,025,800	1%
5-parity	N/A	N/A	N/A	N/A	0%
TC	329.4	2/6.0/10	27/47.6/83	995,000	17%

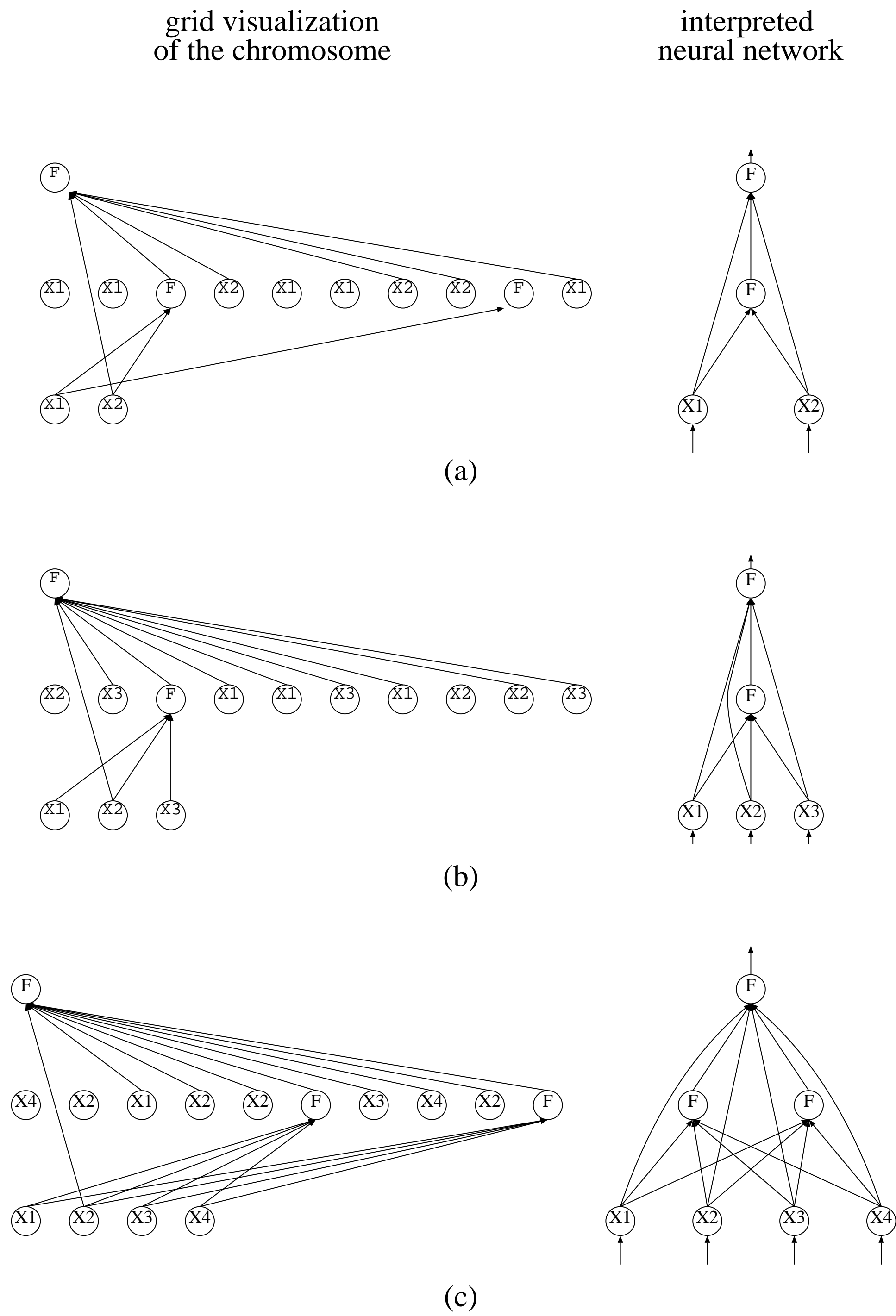


Fig. 6. Grid representation and decoded solutions obtained for the XOR (a), 3-odd parity (b) and 4-odd parity (c) problems.

Table 2. Comparison with other methods (some values represent averages over a varying number of runs): (a) results for the XOR problem found in the literature (entries marked with an asterisk were not explicitly stated by the authors, and were inferred); (b) results for the 3-parity problem found in the literature; (c) results for the odd 4-parity problem reported in the literature; (d) architectures to solve the TC problem described in the literature.

(a)

	generations	population	hidden neurons	connections	training
McDonnell [26]	*60	20	2	*6	no
Yao [34]	90	N/A	*2	*6	yes
Tang [35]	513	N/A	N/A	N/A	no
Zhang [23]	2.9	100	2.2	6.5	yes
Dasgupta [36]	100	*100	N/A	N/A	no
Dual	15	200	2	6	no

(b)

	generations	population	hidden neurons	connections	training
McDonnell [26]	1000	20	4	13	no
Yao [34]	739	N/A	2	10	yes
Dual	50	200	4	13	no

(c)

	generations	population	hidden neurons	connections	training
Zhang [30]	8	200	4	15	yes
Zhang [23]	9	1000	5	23	yes
Dual	130	200	4	23	no

(d)

	topology	connections	generations
McDonnell [26]	13-6-1	34	aprox. 800
Braun [18]	11-1-1	22	N/A
Dual representation	10-1-1	12	423

5-parity, using a specialized function set including Boolean functions.

A second set of experiments was performed with the TC task in order to compare our method with previous methods. The chromosomes were initialized with 16 nodes in the internal layer and the population was allowed to evolve for 500 generations. In this case the number of successful solutions increased to 73%, demanding 390400 chromosome evaluations to find a solution with 99% probability, Our method produced a minimal solution with a 10-1-1 architecture and 12 connections at generation 423. This means that 97% of the 408 possible connections within the 16-16-1 feedforward architecture were eliminated, and 6 unnecessary input neurons were removed. This

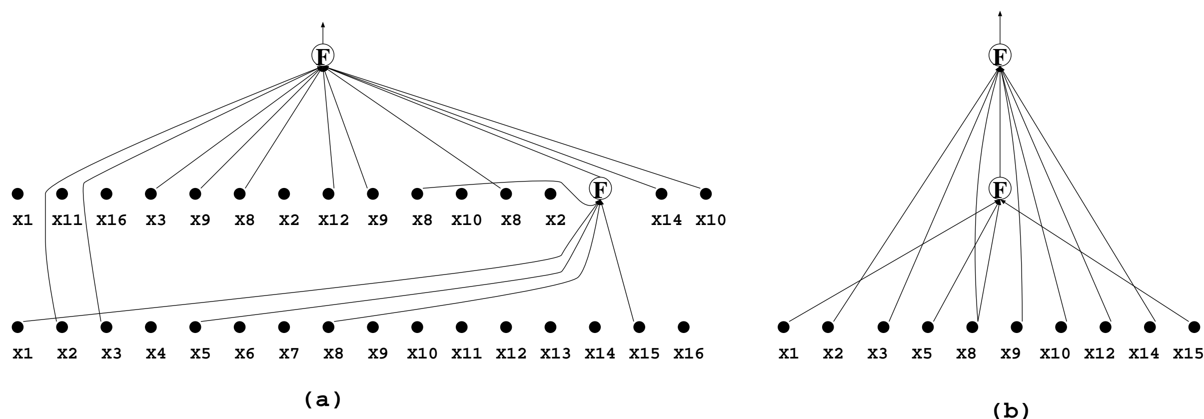


Fig. 7. Minimal solution for the TC task. (a) Structure of the chromosome. (b) The corresponding network, after grouping of duplicate connections, removal of unused terminals and replacement of connections from terminals in the internal layer with connections from corresponding terminals in the input layer.

Table 3. Results for the XOR problem using different grids.

grid	generations	hidden neurons min/avg/max	connections min/avg/max	effort	solutions found
2x2x1	17.3	1/1.3/2	5/5.6/8	14,000	94%
2x5x1	49.4	1/1.2/4	5/5.6/11	36,000	100%
2x2x3x1	49.7	1/1.4/5	5/5.9/13	32,000	100%
2x4x4x1	56.2	1/1.4/5	5/6.0/16	40,000	100%

shows the efficiency of the pruning strategy. Table 2d reports a comparison with results found in the literature. In Figures 7a and b, the layered structure of the chromosome corresponding to the minimal solution and the resulting neural network after decoding are presented. A comparison between this result and that obtained in the first set of experiments, where the initial architectures contained only 10 hidden nodes, suggests that by combining large grids with the pruning strategy, our method can effectively evolve neural networks to solve hard problems.

In order to assess the quality of the minimal solution achieved for the TC problem, its generalization power was tested. This was carried out inverting one bit out of the 16 bits representing each character, and presenting them to the network. The network was still able to identify the noisy templates in 80% of the cases.

The 2-bit adder problem was especially hard to solve: only a 2% of the runs were successful. A solution with 5 hidden neurons and 25 connections was obtained. Whitley *et al.* [37] report a solution with 4 hidden neurons and 29 connections. To the best of our knowledge, the minimal solution, reported in [38], includes 2 hidden neurons and 13 connections.

To evaluate the effect of the size and form of the grid, a third set of experiments was performed with the XOR problem using *node transfer crossover*. The results are shown in Table 3. As can be deduced from the average size of the final solutions in comparison to the initial number of hidden neurons, the pruning strategy consistently reduces the number of hidden neurons.

The slightly smaller percentage of solutions found with a small grid indicates that larger grids are better at obtaining solutions, though the average number of generations to achieve the pruned solutions increases. Rows 2 and 3 show the same number of hidden neurons distributed in one and two layers, respectively. Apparently, two layers give a better performance.

Finally. some experiments were carried out in which both sigmoid and threshold functions were present in the function set. This interesting combination is shown in Figure 8 for the 4-parity and the 2 bit adder problems. This ability of combining different activation functions is a feature offered by very few methods, mostly GP-based.

6. Further work

The results obtained so far are promising, but they were achieved without any optimization of the relative probabilities of the operators. Although some operators seem to perform better than others, a thorough and systematic investigation of their interaction will need to be performed, in a broader context involving other tasks.

In the current implementation, biases have not benefited from the fine tuning accomplished by *node description crossover*. This may be done considering the biases as connections to a constant node.

With the exception of the addition of Gaussian noise, no other mutation operator has been used in the present study. To keep diversity within the population other mutation operators should be explored.

The extension to recurrent neural networks is a natural one, allowing the method to be applied to a wide range of tasks. Most of the operators implemented need not to be changed (actually most of them would be simplified).

Finally, the potential of automatic defined functions shall be explored as an alternative way of computing the weights of the network. A function to compute the weights can be evolved in parallel to the architecture, using the usual arsenal of functions of genetic programming.

7. Conclusion

In this paper, a new approach to the automatic design of neural networks based on evolutionary computation has been presented. New operators were introduced , which exploited a dual representation, in which a linear encoding is used in conjunction with a graph representation.

The representation of the neural network in a linearized form allowed the development of efficient forms of crossover operations and the introduction of a strategy to reduce the complexity of the solutions, whereas the grid description table allowed to control the connectivity properties of the network.

The method was applied to evolve feedforward networks for a variety of binary classification problems showing promising results. Further extensions have been proposed, which will increase the power of the representation and of the operators, allowing it to be applied to an even wider range of practical problems.

Acknowledgements

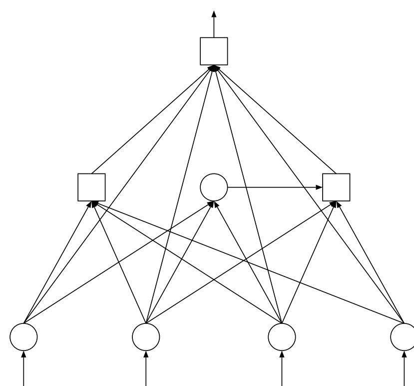
The authors wish to thank the members of the EEBIC (Evolutionary and Emergent Behaviour Intelligence and Computation) group for useful discussions and comments. This research is partially supported by a grant under the British Council-MURST/CRUI agreement and by CNPq (Brazil).

Notes

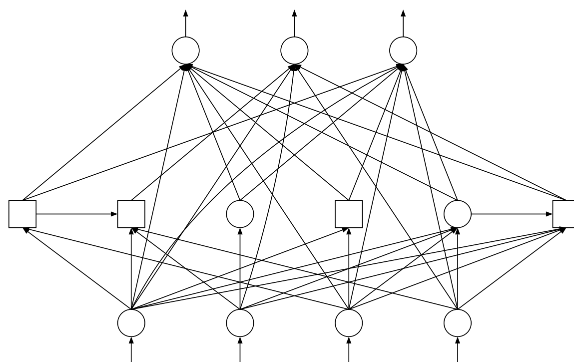
1. Solutions reported by Zhang *et al.* [23, 30] were achieved in fewer generations, but training by a hill-climbing procedure was performed at each generation. In addition, due to the training method applied, only binary weights can be used.

References

1. S. Haykin. *Neural networks, a comprehensive foundation.* Macmillan College Publishing Company, Inc., 866 Third Avenue, New York, New York 10022, 1994.
2. R. Reed. Pruning algorithms: a survey. *IEEE Transactions on Neural Networks*, 4(5):740–747, 1993.
3. M. Frean. The upstart algorithm: a method for constructing and training feed-forward neural networks. *Neural Computation*, 2:198–209, 1990.
4. S. E. Fahlman and C. Lebiere. The cascade-



(a)



(b)

Fig. 8. (a) A solution for the 4-odd parity problem using a mix of sigmoid and threshold activation functions. (b) A solution for the 2 bit adder problem. The squares represent threshold units, whereas the circles in the hidden and output layers represent sigmoid units.

correlation learning architecture. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2, pages 524–532. Morgan Kaufmann, 1990.

5. D. Chen, C. Giles, G. Sun, H. Chen, Y. Less, and M. Goudreau. Constructive learning of recurrent neural networks. In *IEEE International Conference on Neural Networks (ICNN)*, pages 1196–1201, 1993.
6. C. Campbell and C. Vicente. The target switch algorithm: a constructive learning procedure for feedforward neural network. *Neural Computation*, 7:1245–1264, 1995.
7. X. Yao. An overview of evolutionary computation. *Chinese Journal of Advanced Software Research*, 3(1), 1996. To be published.
8. D. Fogel. *Evolutionary computation: toward a new philosophy of machine Intelligence.* IEEE Press, Piscataway, NJ, USA, 1995.
9. D. Goldberg. *Genetic algorithm in search, optimization and machine learning.* Addison-Wesley, Reading, Massachusetts, 1989.
10. M. Mitchell. *An introduction to genetic algorithms.* MIT Press, Cambridge, Massachusetts, USA, 1996.
11. L. Davis. *Handbook of Genetic Algorithms.* Van Nostrand Rheinhold, New York, NY, 1991.
12. Michalewicz Zbigniew. *Genetic Algorithms [plus] Data Structures = Evolution Programs.* Springer-Verlag, Berlin, 1994.
13. S. Harp, T. Samad, and A. Guha. Toward the genetic synthesis of neural networks. In J. Schaffer, editor, *Proceedings of the 3rd International Conference on Genetic Algorithms (ICGA)*, pages 360–369, San Mateo, CA, USA, 1989.
14. M. Mandischer. Evolving recurrent neural networks with non-binary encoding. In *Proceedings of the*

2nd IEEE Conference on Evolutionary Computation (ICEC), volume 2, pages 584–589, Perth, Australia, Nov. 1995.
15. M. Mandischer. Representation and evolution of neural networks. In *Proceedings of the International Conference on Artificial Neural Nets and Genetic Algorithms (ICANNGA)*, pages 643–649, 1993.
16. H. Kitano. Neurogenetic learning: an integrated method of designing and training neural networks using genetic algorithms. *Physica D*, 75:225–238, 1994.
17. S. Fujita and H. Nishimura. An evolutionary approach to associative memory in recurrent neural networks. *Neural Processing*, 1(2), 1994.
18. H. Braun and P. Zagorski. ENZO-M - a hybrid approach for optimizing neural networks by evolution and learning. In Y. Davidor, H. Schwefel, and H. Manner, editors, *Parallel Problem Solving from Nature (PPSN3)*, volume 866. Springer-Verlag, 1994. Lecture Notes in Computer Science.
19. X. Yao and J. Liu. Evolutionary artificial neural networks that learn and generalize well. In *Proceedings of the 1996 IEEE International Conference on Neural Networks*, Washington, DC, Jun. 1996. Submitted.
20. V. Maniezzo. Genetic evolution of the topology and weight distribution of neural networks. *IEEE Transactions on Neural Networks*, 5(1):39–53, Jan. 1994.
21. 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.
22. S. Nolfi and D. Parisi. Growing neural networks. Technical Report PCIA-95-15, Rome, Italy, Jun. 1991.
23. B. T. Zhang and H. Mühlenbein. Evolving optimal neural networks using genetic algorithms with Occam's razor. *Complex Systems*, 7(3):199–220, 1993.
24. P. J. Angeline, G. M. Saunders, and J. B. Pollack. An evolutionary algorithm that constructs recurrent neural networks. *IEEE Transactions on Neural Networks*, 5(1), 1994.
25. 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)*. IEEE Press, 1993.
26. J. McDonnell and D. Waagen. Evolving neural network connectivity. In *Proceedings of IEEE International Conference on Neural Networks (ICNN)*, pages 863–868, San Francisco, CA, USA, 1993.
27. J. Holland. *Adaptation in natural and artificial systems.* University of Michigan Press, Ann Arbor, Michigan, 1975.
28. J. R. Koza. *Genetic Programming, on the Programming of Computers by Means of Natural Selection.* The MIT Press, Cambridge, Massachusetts, 1992.
29. B. Zhang and H. Muehlenbein. Genetic programming of minimal neural nets using Occam's razor. In S. Forrest, editor, *Proceedings of the 5th international conference on genetic algorithms (ICGA'93)*, pages 342–349. Morgan Kaufmann, 1993.
30. B. Zhang and Muehlenbein. Synthesis of sigma-pi neural networks by the breeder genetic programming. In *Proceedings of IEEE International Conference on Evolutionary Computation (ICEC), World Congress on Computational Intelligence*, pages 318–323, Orlando, Florida, USA, Jun. 1994. IEEE Computer Society Press.
31. F. Gruau, D. Whitley, and L. Pyeatt. A comparison between cellular encoding and direct encoding for genetic neural networks. In J. Koza, D. Goldberg, D. Fogel, and R. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 81–89, Stanford University, CA, USA, Jul. 1996. MIT Press.
32. R. Poli. Some steps towards a form of parallel distributed genetic programming. In *Proceedings of the First On-line Workshop on Soft Computing*, Aug. 1996.
33. 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.
34. 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*, pages 149–154, Jun. 1995.
35. 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)*, pages 250–255, Sept. 1995.
36. D. Dasgupta and D. McGregor. Designing application-specific neural networks using the structured genetic algorithm. In L. Whitley and J. Schaffer, editors, *Proceedings of International Workshop on Combinations of Genetic Algorithms and Neural Networks (COGANN)*, pages 87–96. IEEE Computer Society Press, Jun. 1992.
37. D. Whitley, T. Starkweather, and C. Bogart. Genetic algorithms and neural networks: Optimizing connections and connectivity. *Parallel Computing*, 14-3:347–361, 1990.
38. D. Rumelhart and J. McClelland. *Parallel Distributed Processing.* IEEE Press, Piscataway, NJ, USA, 1986.

João Carlos Figueira Pujol Picture and biography to be sent later.

Riccardo Poli Picture and biography to be sent later.