Computing and Informatics, Vol. 24, 2005, 1001–1023, V 2005-Oct-18

NON-DIRECT ENCODING METHOD BASED ON CELLULAR AUTOMATA TO DESIGN NEURAL NETWORK ARCHITECTURES

German Gutiérrez, Araceli Sanchis, Pedro Isasi,
José M. Molina, Inés M. Galván

Scalab, Department of Informatics
Avda. de la Universidad 30
28911-Leganés, Madrid, Spain
e-mail: igalvan@inf.uc3m.es

Manuscript received 2 November 2004; revised 14 June 2005
Communicated by Vladimír Kvasnička

Abstract. Architecture design is a fundamental step in the successful application of Feed forward Neural Networks. In most cases a large number of neural networks architectures suitable to solve a problem exist and the architecture design is, unfortunately, still a human expert's job. It depends heavily on the expert and on a tedious trial-and-error process. In the last years, many works have been focused on automatic resolution of the design of neural network architectures. Most of the methods are based on evolutionary computation paradigms. Some of the designed methods are based on direct representations of the parameters of the network. These representations do not allow scalability; thus, for representing large architectures very large structures are required. More interesting alternatives are represented by indirect schemes. They codify a compact representation of the neural network. In this work, an indirect constructive encoding scheme is proposed. This scheme is based on cellular automata representations and is inspired by the idea that only a few seeds for the initial configuration of a cellular automaton can produce a wide variety of feed forward neural networks architectures. The cellular approach is experimentally validated in different domains and compared with a direct codification scheme.

Keywords: Hyper-heuristic approach for neural network architecture design, evolutionary neural network, indirect encoding, cellular automata

1 INTRODUCTION

The design of Feed forward Neural Networks (FNN) is a crucial point in the successful application because the architecture may strongly drive the neural network's information processing abilities. In most cases, a large number of FNN architectures are suitable to solve an approximation problem. Although some heuristics based on constructive algorithms has been developed to design FNN architectures [1], the architecture design is unfortunately still a human expert's job. It depends heavily on the expert experience and on a tedious trial-and-error process. There is no systematic way to design a near optimal architecture automatically for a given task.

In the last years, many works have been centred on the automatic resolution of the design of neural network architectures. Most of the works are based on genetic approaches because the design of optimal architectures can be formulated as a search problem in the architecture space, where each point represents a FNN architecture. The search space of all possible architectures is very large, and the task of finding the best architecture may be a hard and mostly random task. Hence, genetic algorithms are appropriate techniques to drive the search in the space of FNN.

The research has mainly been concentrated on design of FNN architectures using genetic approaches; there are many works in the literature [2, 3, 4, 5, 6, 7, 8]. This kind of approximation has recovered importance in the last years, the hyper-heuristic searches [9, 10]. Other authors have paid attention to other factors, such as weights of the network [11, 12, 13, 14, 15]; activation functions [16]; learning rules (or related parameters) [17, 18]. Reviews of using evolutionary techniques to evolve different aspects of neural networks can be found in [19, 20].

This paper is focused on the design of FNN architectures using genetic algorithms. In this context, a key point is represented by the codification of FNN architectures as elements of the genetic algorithm population. Two main representation approaches exist to find the FNN architecture to solve a problem. One, based on complete representation of all possible connections, and the other based on an indirect representation of the architecture. The first one is called *direct encoding method*, and is based on the codification of the complete network (connections matrix) into the chromosome of the genetic algorithm, starts with Ash's work [21] and continues with other works, including [2, 12, 22, 23, 24]. Direct representations are relatively simple and straightforward to implement. However, they do not scale very well since large architectures require very large chromosomes to be represented. It is specially suitable for small architectures. In these cases, some unpredictable designs could be reached [24]. However, the capabilities of direct encoding for larger architectures are limited, remaining to be proven whether it can scale more complex tasks, because large architectures require much larger chromosomes.

In order to reduce the genotype length and to make the problem more scalable, indirect encoding scheme has been proposed in the last years. Indirect encoding scheme consists of codifying, not the complete network, but a compact representation of it, avoiding the scalability problem. One of the indirect encoding schemes was proposed by Kitano [25], introducing a constructive scheme based on grammars.

The solution proposed by Kitano was to encode neural networks as grammars, and let the genetic algorithm to evolve grammars instead of network architectures. An extension and improvement of the Kitano's method can be found in [26, 27]. In those works, a mechanism of representing FNN by means of graph grammars, without restrictions in word's size or recursion, is developed. Other works have considered some variations of Kitano's rule descriptions using recursive equations to model the growth of connections matrix [28]. In this case, the coefficients of some fixed equations were codified as chromosomes and evolved by a genetic algorithm.

The grammatical approach is not the unique proposed representation method. Merril and Port [29] introduce a fractal representation for encoding the architectures, arguing that it is more related with biological ideas than constructive algorithms. They used fast simulated annealing for the evolution of architectures. Valls et al. [30] proposed a multiagent system to find optimal architectures for radial basis neural networks.

In this work, an indirect constructive encoding scheme, based on cellular automata, is proposed to find automatically an appropriate FNN architecture for a given problem. Cellular automata [31] consist of an n-dimensional grid of m machines, called cells, which could be in different states. Usually, the dimension of the lattice is one or two, and the states of the cells are binary. The cells can change their state depending on the state of the cells in their neighbourhood. This modification is specified through the automata rules. All the cells change their state following the same rules. The state of each cell in the lattice at one instant determines what is called configuration. The evolution of a cellular automaton is determined by the application of the rules over a configuration to obtain a new configuration, and so on. The features of a cellular automaton depend on the rules that govern its behaviour.

The paper is organized as follows: Section 2 describes the general architecture of this indirect encoding method of obtaining FNN. The cellular module, the proposal of this paper, is described in Section 3. The dynamics of the complete system is presented in Section 4. Section 5 is related with the experiments and the obtained results. Some conclusions are presented in Section 6.

2 CELLULAR SYSTEM APPROACH

As mentioned, an indirect constructive encoding scheme, based on cellular automata, is presented in this work, and is called cellular approach. Some preliminary results are presented in [32]. The method finds automatically appropriate FNN architectures to solve a given problem. The global system is composed of three different modules: the genetic algorithm module, the cellular module and the neural network module. The proposed system follows the general mechanism of other well-known systems like Kitano's [25] and the GANET system [26, 27].

All the modules are related to make a general procedure of generating and optimising FNN architectures. The cellular module is composed of two two-dimensional cellular systems, and it has been designed to generate FNN architecture with one

hidden layer. This assumption has been done because it is proven that FNN with as few as one hidden layer are able of approximating any non-linear continuous function [33]. Several seeds give the initial configurations of the two-dimensional cellular systems and the rules of these systems are applied to generate final configurations. The first cellular system, called growing cellular automata, is related to the generation of FNN with a large number of connections. The second cellular system, named pruning cellular automata, is related to the reduction of this number of connections. The complete cellular system allows obtaining the required FNN to solve a considered problem. The detailed description of the cellular system is accomplished in Section 3. The complete dynamics of the system, considering the cellular, the neural and the genetic modules, will be described in Section 4. In the next section, both neural and genetic modules and their hierarchical dependencies are explained.

As previously mentioned (and will be described in Section 3), the two cellular systems are expanded and a binary matrix M of $Dim_x \times Dim_y$ dimension is obtained. The size of this matrix depends on the number of input and output neurons of the FNN, which are given by the problem and on the maximum number of hidden neurons to be considered. Thus, Dim_x (rows) is equal to the number of input neurons n, plus the number of output neurons m, and Dim_y (columns) corresponds with the maximum number of hidden neurons to be considered (see Figure 1).

To relate that matrix with an architecture of a FNN with one hidden layer, the meaning for the grid position (i, j) is defined as follows. Let n be the number of inputs neurons; if $i \leq n$ then (i, j) represents a connection between the i^{th} input neuron and the j^{th} hidden neuron; if $i > n$, (i, j) represents a connection between the j^{rmth} hidden neuron and the $(i - n)^{\text{th}}$ output neuron. That relation is shown in Figure 1.

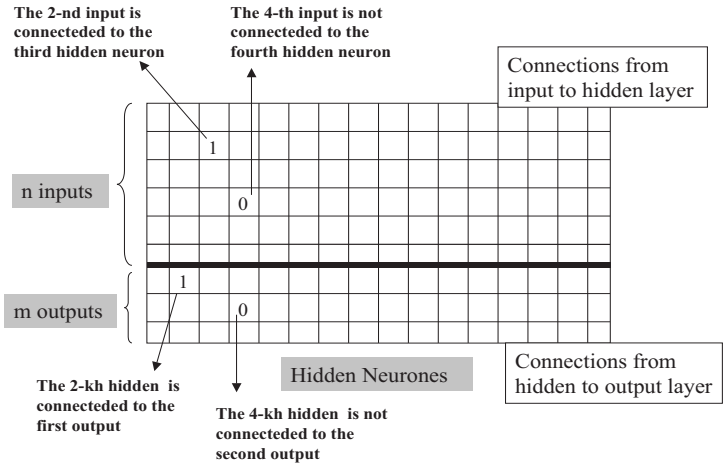


Fig. 1. Relation between binary matrix (M) and FFN architectures

Hence, the meaning of each 1 in M is interpreted as a connection and a 0 as the absence of connection. Before obtaining the definite FNN architecture, the matrix M must be transformed to a shorter matrix as follows:

- The columns with values 0 in the matrix are removed. If the elements of the k^{th} column are zero, there are no connections from the inputs to the k^{th} hidden neurons and there are no connections from the k^{th} hidden neuron to the outputs. Hence, that hidden neuron must be removed.
- When there is a neuron in hidden layer without any connection to output layer, this node is also eliminated from the network, because it will not have any influence in the outputs of the network.

The obtained FNN is trained to solve the particular problem considered. Weights of the neural network are randomly initialised, and learned using the backpropagation learning method. A value measuring the efficiency of the architecture is computed after the learning phase of the network and is used as the fitness function of the chromosome.

In order to evaluate the efficiency of one FNN, different criterions must be taken into account. First, the FNN must provide the best approximation and generalization. For this purpose, the training and validation errors would be measured. Secondly, in order to find the optimal architecture it is necessary to establish an arrangement between the training and validation errors produced by the network and the number of connections (number of neurons) in the network. In the context of neural networks it is interesting to obtain the simplest network being able to achieve adequate validation errors. Hence, the number of connections might be introduced to evaluate the FNN. However, the complexity of the network depends also on the number of learning cycles. Neural network architecture with few connections may require a large number of learning cycles to reach appropriate approximations depending on the placement of the connections. The computational effort involved in the training phase of the network might also be measured to evaluate the efficiency of FNNs. The fitness function used in this paper will be described in the experimental section.

The genetic algorithm module takes charge of generating the positions of the seeds in the two-dimensional grid of cellular systems, which determines initial configurations of cellular systems. Those positions are codified in the chromosome of a genetic population, which evolves to maximise the fitness function provided by the neural network module which evaluates the efficiency of the network to solve the considered problem.

Chromosomes have been codified in base b, where b is the number of rows in the grid (i.e. Dim_x) and, as previously mentioned, is given through the number of input neurons plus the number of output neurons in the FNN to solve a given problem. Each seed is defined through two coordinates (i, j). The first coordinate i could be represented by only one gene (see Figure 2), indicating the row in which the seed is located. The second coordinate j will require more than one gene if, as usual, the

maximal number of hidden neurons is bigger than b (input plus output neurons). In this case, two genes have been used to codify the coordinate j (see j_1 and j_2 of Figure 2), what allows a maximum of $b \times b$ hidden neurons. For instance, if there are 3 inputs and 2 outputs, the maximum size of the hidden layer is: $5 \times 5 = 25$. This could be a good estimation of the maximum number of neurons in the hidden layer, but any other consideration could be taken into account without modifying the proposed method.

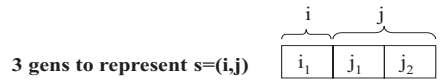


Fig. 2. Codification of seeds position in the chromosome structure

The chromosome will have 3 genes for each seed to be placed in the grid. Firstly, the seeds for the first automata (growing seeds) are represented, and finally the seeds for the pruning automata (decreasing seeds). If, for instance, there are five growing seeds and five decreasing seeds, the size of the chromosome is 30, divided in ten trios of genes; the first five trios represent the growing seeds, and the final five trios represent the decreasing seeds.

3 CELLULAR MODULE

The cellular module is composed of two two-dimensional cellular systems (see Figure 3), called "growing cellular system" and "pruning cellular system". The first one is used to build up FNN architectures with a large number of connections, as well as fully connected FNN. The second cellular system is incorporated to remove connections in FNN architectures obtained by the first system. Both systems allow obtaining a wide variety of FNN architectures.

Both two-dimensional cellular systems consist of a regular grid of sites or cells. Each site or cell takes different values and the system is updated in discrete time steps according to some rule that depends on the value of sites in some neighbourhood around it. In this work, the neighbourhood structure for cellular systems is defined by the square region around a cell, which is usually referred to as "nine – neighbourhood square". Hence, the value $a_{i,j}$ of a cell at the position (i, j) in the two-dimensional grid evolves according to Equation (1):

$$a_{i,j}^{(t+1)} = \Phi(a_{i-1,j-1}^{(t)}, a_{i-1,j}^{(t)}, a_{i-1,j+1}^{(t)}, a_{i,j-1}^{(t)}, a_{i,j}^{(t)}, a_{i,j+1}^{(t)}, a_{i+1,j-1}^{(t)}, a_{i+1,j+1}^{(t)}, a_{i+1,j+1}^{(t)}) \quad (1)$$

where Φ is given by the rule of the cellular system. Depending on the rule and on the initial configuration – i.e. the initial values of cells in the grid – both cellular systems evolve towards different final configurations.

In this work, the size of the grid for the cellular systems is previously defined and fixed to $Dim_x \times Dim_y$. As defined in the previous section, Dim_x is equal to

the number of input neurons n, plus the number of output neurons m, and Dim_y corresponds with the maximum number of hidden neurons to be considered.

In the next, the initial configurations, possible values of each cell and the evolution rules of the growing and pruning cellular systems are presented. After that, the complete mechanism to evolve and combine those cellular systems is described.

3.1 Growing Cellular System

The growing cellular system is designed in order to obtain FNN architectures with a large number of connections between the input and hidden layer and between hidden and output layer. The initial configuration of the growing cellular system is given by n seeds, $(s_1, s_2, \ldots, s_n)$, called "growing seeds". Two coordinates define each seed (Figure 2), which indicates the positions of the seed in the grid. The genetic algorithm module provides those positions. In order to apply the automaton rules for the first time it is necessary to replicate each seed sequentially, over its quadratic neighbourhood. The replication is made in such a way that if a new seed has to be placed in a position previously occupied by another seed, the first one is replaced.

Thus, the value $a_{i,j}$ of the cell at the position (i, j) in the growing cellular system can take two possible values:

- $a_{i,j} = 0$: the cell (i, j) is inactive,
- $a_{i,j} = s_k$: the cell (i, j) contains the seed s_k.

The rule of the growing cellular system has been designed to allow the reproduction of growing seeds. The idea is to copy a particular growing seed sk when a cell is inactive (i.e. $a_{i,j} = 0$) and there are at least three identical growing seeds in its neighbourhood. The rule of the growing cellular automaton is defined in Equation (2):

$$
\begin{aligned}
a_{i,j}^{(t+1)} = s_k \text{ IF } a_{i,j}^{(t)} = 0 \text{ AND} \qquad (2)\\
a_{i-1,j-1}^{(t)} = a_{i-1,j}^{(t)} = a_{i-1,j+1}^{(t)} = s_k \text{ OR}\\
a_{i+1,j-1}^{(t)} = a_{i+1,j}^{(t)} = a_{i+1,j+1}^{(t)} = s_k \text{ OR}\\
a_{i-1,j-1}^{(t)} = a_{i,j-1}^{(t)} = a_{i+1,j-1}^{(t)} = s_k \text{ OR}\\
a_{i-1,j+1}^{(t)} = a_{i,j+1}^{(t)} = a_{i+1,j+1}^{(t)} = s_k \text{ OR}\\
a_{i-1,j-1}^{(t)} = a_{i-1,j}^{(t)} = a_{i,j-1}^{(t)} = s_k \text{ OR}\\
a_{i-1,j}^{(t)} = a_{i-1,j+1}^{(t)} = a_{i,j+1}^{(t)} = s_k \text{ OR}\\
a_{i,j-1}^{(t)} = a_{i+1,j-1}^{(t)} = a_{i+1,j}^{(t)} = s_k \text{ OR}\\
a_{i+1,j}^{(t)} = a_{i+1,j+1}^{(t)} = a_{i,j+1}^{(t)} = s_k\\
a_{i,j}^{(t+1)} = a_{i,j}^{(t)} \text{ otherwise.}
\end{aligned}
$$

According to that rule, a seed s_k is reproduced when there are at least three identical growing seeds in its neighborhood, which must be located in the same row, or in the same column or in the corner of the neighborhood. In Table 1 a graphical representation of the rule is shown. Here, s_k is a growing seed, 0 is an inactive state for the cell and $*$ means that the cell could be in any state or contains any type of seed.

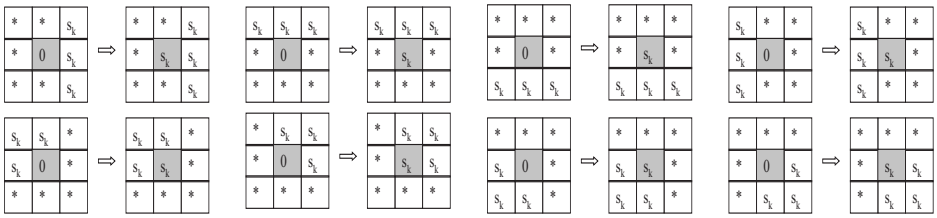


Table 1. Configurations of the rule for the growing cellular system

3.2 Pruning Cellular System

Once the growing cellular system is expanded, most of the cells in the grid are occupied by growing seeds. If the presence of a growing seed is considered as the presence of a connection in the network, it could be convenient to remove seeds in the grid in order to obtain a large variety of architectures. Hence, the pruning cellular system is incorporated to represent and to obtain FNN architectures in which neurons in different layers are not fully connected.

Thus, the initial configuration of the pruning cellular system is given by the final configuration of the growing cellular system and by m seeds $(d1, \ldots, d_m)$, called "decreasing seeds" in this work. As in the previous cellular system, two coordinates that indicate the position of the seed in the grid define each seed and they are provided by the genetic algorithm module.

The value $a_{i,j}$ of the cell at the position (i, j) in the decreasing cellular system can take the following possible values:

- $a_{i,j} = 0$: the cell (i, j) is inactive,
- $a_{i,j} = s_k$: the cell (i, j) contains the growing seed s_k,
- $a_{i,j} = d_r$: the cell (i, j) contains the decreasing seed d_r.

The rule of the pruning cellular system is designed to remove growing seeds in the grid. A growing seed s_k is removed when two contiguous neighbouring cells contain identical growing seeds and another neighbouring cell contains a decreasing seed. If two decreasing seeds are present in the neighbourhood, the rule is not activated. The rule of the decreasing cellular system is defined in Equation (3).

$$a_{i,j}^{(t+1)} = d_r \text{ IF } (a_{i,j}^{(t)} = a_{i-1,j-1}^{(t)} = a_{i,j-1}^{(t)} = s_k \text{ AND } a_{i-1,j}^{(t)} = d_r) \text{ OR} \qquad (3)$$

$$(a_{i,j}^{(t)} = a_{i-1,j-1}^{(t)} = a_{i-1,j}^{(t)} = s_k \text{ AND } a_{i,j-1}^{(t)} = d_r)$$
$$(a_{i,j}^{(t)} = a_{i-1,j+1}^{(t)} = a_{i,j+1}^{(t)} = s_k \text{ AND } a_{i-1,j}^{(t)} = d_r)$$
$$(a_{i,j}^{(t)} = a_{i-1,j}^{(t)} = a_{i-1,j+1}^{(t)} = s_k \text{ AND } a_{i,j+1}^{(t)} = d_r)$$
$$(a_{i,j}^{(t)} = a_{i,j-1}^{(t)} = a_{i+1,j-1}^{(t)} = s_k \text{ AND } a_{i+1,j}^{(t)} = d_r)$$
$$(a_{i,j}^{(t)} = a_{i,j+1}^{(t)} = a_{i+1,j+1}^{(t)} = s_k \text{ AND } a_{i+1,j}^{(t)} = d_r)$$

$a_{i,j}^{(t+1)} = a_{i,j}^{(t)}$ otherwise.

The previous rule (3) removes a growing seed in a cell when there are one decreasing seed in the neighbourhood and the rest of the cells in the neighbourhood satisfy a particular condition. Similar rules could be used, but the design must enforce that not all growing seeds in the grid are removed. Table 2 shows a graphical representation of the rule. Here, s_k is a growing seed, dr is a decreasing seed and $*$ means that the cell could be in any state or contains any type of seed.

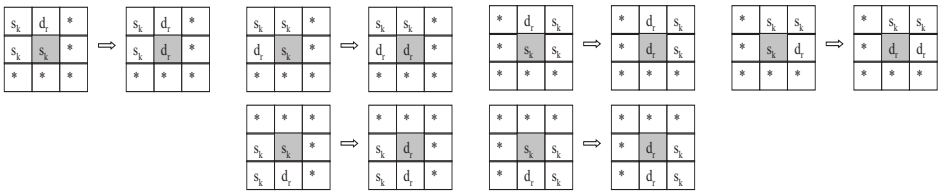


Table 2. Configurations of the rule of the pruning cellular system

3.3 Evolving Growing and Pruning Cellular Systems

In order to evolve the cellular systems and to combine both growing and pruning cellular systems, a special procedure has been proposed. This procedure allows the convergence of the complete system toward a final configuration depending on the initial configuration. The mechanism of expanding the systems is as follows:

1. All cells in the grid are set in the inactive state and the growing seeds provided by the genetic module are located in the grid. In order to apply the rule of the growing cellular system for the first time, the growing seeds are replicated over their quadratic neighbourhood in such a way that if a new seed has to be placed in a position previously occupied by another seed, the first one is replaced. That configuration is the initial configuration of the growing cellular system.
2. The rule of the growing cellular system is applied until no more rule conditions could be fired. The configuration obtained is the final configuration of the growing cellular system.

3. The decreasing seeds are placed in the grid to obtain the initial configuration of the decreasing cellular system. If there are some other seeds in those places, they are replaced.
4. The rule of the pruning cellular system is applied until the final configuration is reached.
5. A binary matrix M is finally obtained, replacing the growing seeds by a 1 and the decreasing seeds or inactive cells by a 0. That matrix will be used by the neural network module to obtain a FNN architecture, as described in Section 2.

4 COMPLETE DYNAMICS OF THE SYSTEM

In this section all the parts of the complete cellular approach and the relationship among the different modules are presented. The global system is shown in Figure 3. The complete dynamic of the cellular approach can then be described as follows:

1. Individuals of the population of the genetic module are randomly generated, i.e., random positions of seeds to the cellular systems are generated.
2. Each chromosome is decodified and converted in the grid locations according to the codification explained in Section 2. Every three genes represent a seed. The first gene represents the x-coordinate of the seed and the other two genes, the y-coordinate.
3. The growing and pruning cellular systems are evolved following the procedure described in Section 3.
4. The final configuration, binary matrix M, of the pruning cellular system is translated into a FNN architecture, as described on Section 2.
5. Weights of the FNN are randomly initialised, and the network is trained using backpropagation learning algorithm.
6. A value measuring the efficiency of the architecture is computed. This value is composed not only by the error of the network, but some other considerations could be taken into account to evaluate the efficiency of the network as the generalisation capability, the size of the network and the convergence velocity. This value is used as the fitness function of the chromosome.
7. Steps 2 to 6 are repeated for all individuals in the population. New populations are generated by genetic algorithm using the usual mutation and reproduction operators.
8. The procedure, steps 2 to 7, is carried out through different generations, until the fitness function is optimized.

5 EXPERIMENTAL VALIDATIONS

The proposed indirect cellular encoding scheme has been tested in four different domains. In all cases, the goal is to obtain optimal FNN architectures being able

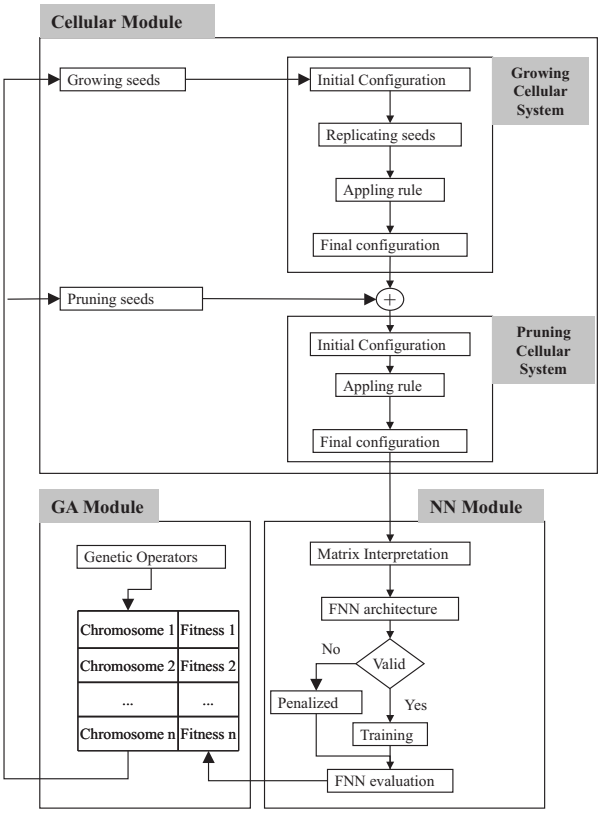


Fig. 3. Complete dynamics of the cellular approach

to solve the given problem. In this section, first the conditions of the experiments carried out and the characteristics of different domains are described. Secondly, the results obtained with the cellular method presented in this work and the results provided by a direct codification are also presented and compared.

5.1 Experimental Definition

The cellular approach has been used to design automatically appropriate FNN architectures in different domains: the minimum interesting coding problem, the parity problem, the prediction of logistic time series and a medical classification problem. In the next, the characteristics of all of them are presented.

Minimum Interesting Coding Problem. The minimum interesting coding problem (MICP) is a simple domain and our interest is due to the fact that there are some input variables that are just noise, bearing no relation to the output. In

this problem there are four inputs, all of them binary, and two outputs. The two first inputs are noise and the relationship between the third and fourth inputs and the two outputs is given in Equation (4).

$$X_1 = X_3; X_2 = X_3 \text{ xor } X_4 \tag{4}$$

Training patterns are composed by four inputs and an optimal architecture might not consider the two first inputs, which are just noise. Since the problem is given by four inputs and two outputs, the dimension of the grid to evolve the cellular systems is 6×36, which implies that the maximum number of hidden neurons allowed for the FNN is 36. According to the description of the approach, seeds have to be codified in base 6.

Parity problem. The parity problem has a long history in the study of neural networks. This is a mapping problem where the domain set consists of all different N-bit binary vectors and the result of the mapping indicates whether the addition of N components of the binary vector is odd or even. Many authors have studied the number of hidden neurons which must be considered to solve the problem. Minor [34] showed that when the sigmoidal function is used as the transfer function and direct connections from the input layer to the output layer are allowed, the required number of hidden neurons is the nearest integer to $N/2$. However, the most commonly used FNN architecture is the one where there is one hidden layer with connections only between the input and hidden layer and between the hidden and output layer. With this kind of architectures, it has been thought first that N hidden neurons are required to solve the N bit parity problem [32]. However, most current studies [35] show that a sufficient number of hidden units for the network is $(N/2) + 1$ if N is even and $(N + 1)/2$ if N is odd. Since there are studies about the number of hidden neurons required to solve the N-parity problem, it is an interesting domain to validate the proposed method to find optimal architectures being able to solve the problem.

In this work, parity seven has been considered as a study case. Hence, the network will have 7 input neurons and 1 output. For simplicity, all the hidden neurons are considered connected with the only output neuron; this reduces the size of the chromosome in one unit without loosing generality. This assumption can be done only if there is a unique output neuron. Hence, the size of the grid for the automaton is 7×49 and the seeds are codified in base 7.

The logistic time series prediction. The indirect encoding scheme has been also applied to determine the simplest FNN being able to approximate the logistic time series. The logistic map is given by Equation (5).

$$x(k + 1) = \lambda x(k)(1 - x(k)) \tag{5}$$

When $\lambda = 3.97$ and $x(0) = 0.5$, the map describes a strongly chaotic time series. The use of the logistic map has the advantage that the number of input neurons

to predict the map is known. As the value of the time series at instant t depends only on the value at instant $t-1$, the network might consider only the input carrying the $t-1$ signal. Hence, one output neuron, and one input neuron must be enough to obtain suitable approximations. However, in order to increase the complexity of the problem and to test the ability of the method to generate good architectures, five inputs have been taken into account: $x(k-4), \ldots, x(k)$. Thus, the system must find, in addition to the minimum number of hidden neurons to solve the problem, the most relevant input variables in the dynamic behaviour of the logistic time series, that is, considering only the $x(k)$ input.

As in the parity problem, it is assumed that all the hidden neurons are connected with the only output neuron. Thus the size of the grid for the automaton is 5×25 and seeds are codified in base 5.

Medical classification problem: Thyroid Gland Data. Thyroid data are measurements of the thyroid gland. Each pattern has 5 continuous attributes which are used to try to predict whether a patient's thyroid belongs to the class euthyroidism, hypothyroidism or hyperthyroidism. The diagnosis (the class label) was based on a complete medical record, including anamnesis, scan, etc. [36]. This classification problem is hard to solve for neural networks. That domain is a medical classification problem and it has been chosen to validate the proposed indirect method in a real domain. In this case, the size of the grid is 8×64 and seeds are codified in base 8.

Fitness Function Given a measured value of the error to reach, the purpose is to obtain FNN architectures with the least computational effort. In this context, the meaning of computational effort is the number of weights changed along the training process. Then, neural networks with a minimal number of hidden nodes and reaching the previously fixed error as soon as possible along the training process, are looked for.

The FNN architectures obtained from the final configuration of cellular systems are trained until a desired error is reached. Since the goal is to find FNN architectures requiring the least computational cost to reach an appropriate level of error, the fitness function provided used in this work is the inverse of computational effort:

$$Fitness = \frac{1}{(c \cdot tc)} \tag{6}$$

where c is the number of connections in the FNN architecture and tc the number of training cycles carried out.

When the level of error is not reached by the network, the training is carried out during a maximum of learning cycles. In this case, the fitness value associated is given by equation (7):

$$Fitness = \frac{1}{(c \cdot tc)} \cdot \left(\frac{e_{fixed}}{e_{reached}} \right) \tag{7}$$

where $e_{reached}$ is the reached error and e_{fixed} the desired error previously fixed. Thus, the fitness value associated to FNN architectures that do not reach the desired error depends on how large is the difference between the network reached error during the maximum number of learning cycles and the desired error previously fixed.

According to the previous definition of the fitness function, the desired error, e_{fixed}, and the maximum number of learning cycles must be determined for each domain. With this purpose, full-connected FNN architectures have been previously trained. Table 3 shows those values for each domain, as well as the trained architecture. The maximum number of learning cycles is fixed according to the learning cycles required to reach the fixed error.

	Architecture Inp-hid-out	e_{fixed}	Learning cycles to reach the error	Maximum learning cycles
MICP	4-10-2	0.001	1 000	10 000
Parity	7-7-1	0.08	500	5 000
Logistic series	1-8-1	0.002	5 000	15 000
Thyroid gland data	5-10-3	0.07	7 500	15 000

Table 3. Fitness function parameter

Experimental parameters The cellular approach requires a certain number of seeds that must be placed in the grid in order to evolve both growing and pruning cellular systems. A minimum number of seeds is required and depends on the size of the grid. For each domain, five sets of experiments have been realised, changing the number of growing and decreasing seeds: 3-3, 4-4, 5-5, 6-6, 7-7, which implies the variation of chromosome size. Hence, the length of the chromosome in the cellular indirect encoding ranges from 8 to 42 genes.

The performance of the cellular approach has been also compared with a direct encoding scheme. That direct codification consists of a binary matrix of dimension $Dim_x \times Dim_y$, where Dim_x is equal to the number of input neurons plus the number of output neurons, and Dim_y corresponds with the maximum number of hidden neurons to be considered. This binary matrix constitutes the chromosome of the individual that represents the architecture. In the direct codification, the length of the chromosome is given by $Dim_x \times Dim_y$ and depends on the domain, as illustrated in Table 4.

	Grid size for cellular automata	Chromosome length for direct codification
MICP	6×36	216
Parity	7×49	343
Logistic series	5×25	120
Thyroid gland data	8×64	512

Table 4. Length of the chromosome for direct codification

In all experiments, the GA is composed of a population of 100 individuals, which are initially randomly generated. Populations are iterated for n generations, where n varies with the domain and the codifications, as shown in Table 5. The rest of parameters take standard values, and they are shown in Table 6. Several runs changing the seed of the genetic algorithm randomly are also carried out.

	Generations for the indirect codification	Generations for the direct codification
MICP	400	400
Parity	700	300
Logistic series	170	200
Thyroid gland data	150	200

Table 5. Number of generations

	Direct Encoding	Indirect Encoding
Population size	100	100
Elitism	10 %	10 %
Selection	Roulette wheel	Roulette wheel
Crossover	100 %	100 %
Mutation	1/LC	0.01

Table 6. GA parameters. LC: length chromosome

5.2 Experimental Results

Figures 4–7 show the architectures obtained using the indirect encoding based on cellular automata for the different domains after the number of generations specified in Table 5. In those figures the architectures obtained for the different number of growing and decreased seeds used in the cellular approach are included.

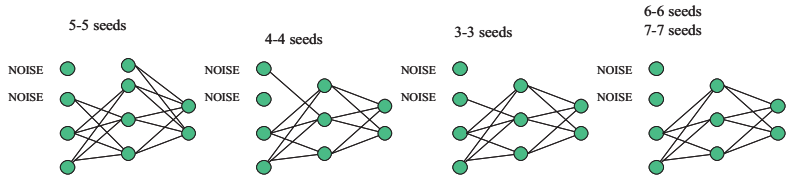


Fig. 4. FNN architectures obtained by the cellular system for the MICP

In most of the cases, the indirect method provides appropriate architectures to solve the problem. In the MICP, the best architecture obtained has 3 hidden nodes, and there are no connections from inputs X_1 and X_2 to hidden layer (see Figure 4).

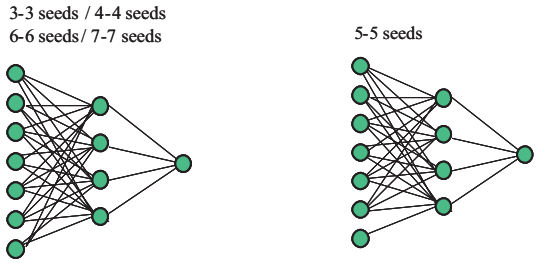


Fig. 5. FNN architectures obtained by the cellular system for the parity domain

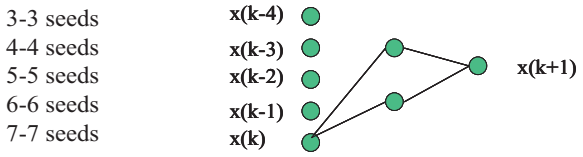


Fig. 6. FNN architecture obtained by the cellular systems for the logistic time series

This result is obtained using 6 and 7 growing and decreasing seeds. For the remaining number of seeds, the FNN architectures obtained are not much different, although some connections from some noise input appear. In the parity domain (see Figure 5), the architectures have four hidden neurons, as desired, and most of them are fully connected. Only in one case (5-5), the architecture is not fully connected, the first input neuron is only connected to one hidden neuron, without connections to the remaining hidden neurons. Also, the optimal architecture is founded in the logistic time series domain (see Figure 6), independently of the number of seeds previously fixed. They take into account the only relevant input to the problem, $x(k-1)$. In the medical problem, architectures with 3 and 4 hidden nodes and fully connected are obtained as shown in Figure 7.

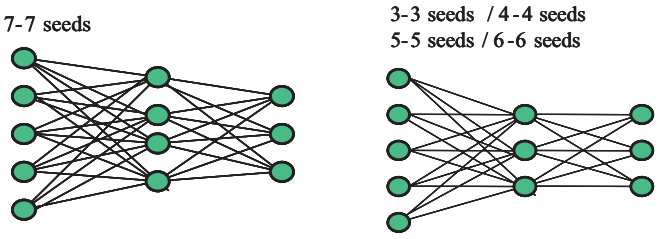


Fig. 7. FNN architecture obtained by the cellular systems for the thyroid gland problem

The best FNN architectures for the MICP and the logistic time series domain obtained with the direct codification after the generations indicated in Table 5 are shown in Figures 8 and 9, respectively. In the parity domain, the architecture obtained has 48 hidden neurons and 48 % of connectivity, and in the medical domain the network has 21 hidden neurons and 55 % of connectivity. For both cases, figures are not included in the text due to the size of the networks.

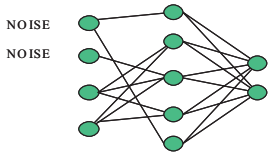


Fig. 8. FNN architecture obtained by direct codification for the MICP

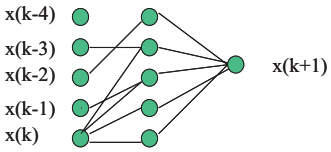


Fig. 9. FNN architecture obtained by direct codification for the logistic time series

As can be observed, in all domains the architectures provided by using a direct codification is much more complex than the FNN architectures obtained with the cellular method, and irrelevant inputs are taken into account. It is also noticed that the number of generations to obtain those more complex architectures is larger than that of the generations required by cellular approach (see Table 5).

For both indirect and direct approaches, the evolutions of average fitness of 10 best architectures along the generations for each domain are shown in Figures 10–13. For the indirect approach, the fitness evolution corresponds to the best pair of constructive and destructive seeds. The indirect cellular encoding is able to provide more optimal architectures than direct encoding in a lower number of generations. In the MICP and in the logistic time series prediction, the direct codification method is able to find architectures nearby to the architectures founded by the indirect method, although more generations are used. However, in the parity and medical classification domain, the direct method is so far to found an appropriate architecture because the length of the chromosome increases considerably (see Table 4).

6 CONCLUSIONS

The choice of appropriate FNN architectures is an important step in many problems where there is little knowledge about the problem itself. Evolutionary computation

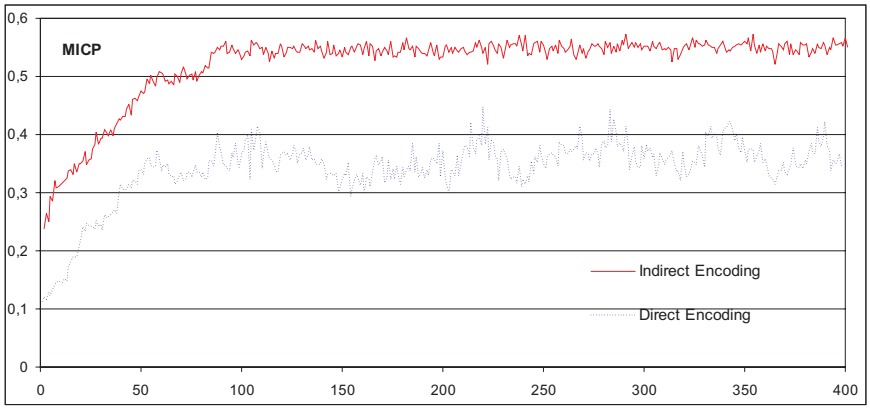


Fig. 10. Average fitness of 10 best for MICP

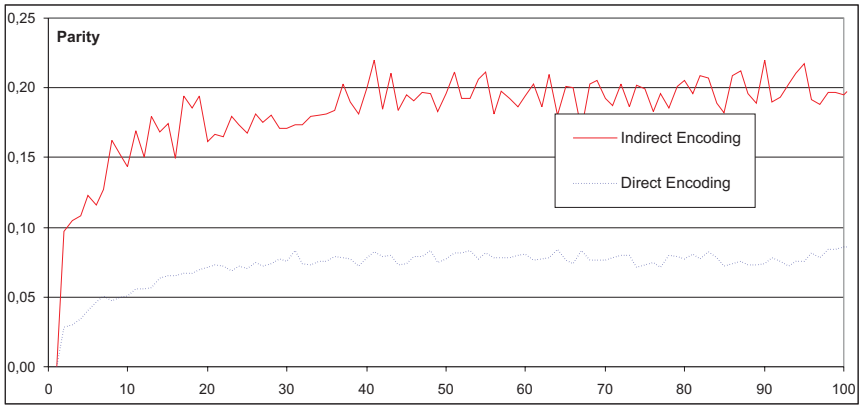


Fig. 11. Average fitness of 10 best for parity problem

techniques are good approaches for automatically generating those good architectures. However, the codification of the network is a crucial point in the success of the method. Direct codifications become inefficient from a practical point of view, making the search space bigger and redundant, and the evolution process very slow for large domains. To solve this problem, indirect encoding approach is a good candidate.

Cellular automata are candidates for non-direct codifications. The constructive representation presented in this work solves some of the problems for non-direct codifications because only few seeds for initial configuration of the cellular automata can produce a wide variety of FNN architectures. The chromosome has a reduced size and could be controlled by the number of seeds used.

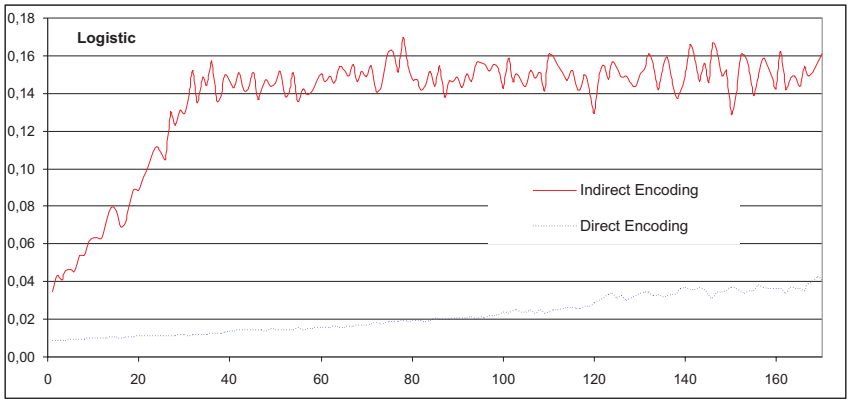


Fig. 12. Average fitness of 10 best for logistic time series problem

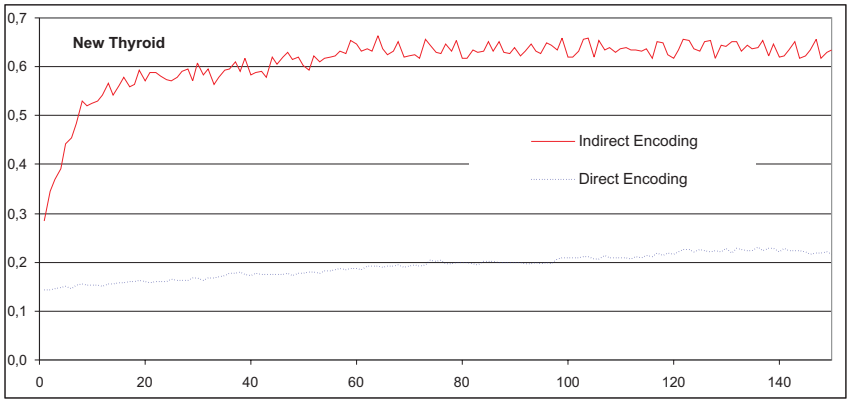


Fig. 13. Average fitness of 10 best for thyroid gland classification problem

Since the final configuration of the two-dimensional grid will represent a FNN matrix connection, the rule of the automata used in that approach has been such a design that a wide variety of architectures may be obtained. The rule of growing cellular automata has been designed to obtain FNN architectures with a large number of connections. On the other hand, the rule of the decreasing cellular system is built up to remove connections. In this work, the rules specified in Section 3 have been used to expand the cellular systems. However, different rules could be used, although the design must be enforced to generate both full-connected FNN architectures and architectures with few connections only.

The cellular approach has been validated in different domains with different characteristics and compared with a direct codification. The results show that the indirect encoding approach presented in this paper is able to find appropriate FNN

architectures and, in most of the cases, optimal architectures. In addition, the number of generations carried out over the population is less than when direct codifications are used. The main factor that contributes to decrease the number of generations is the length of the chromosome, which is considerably reduced when the indirect cellular approach is used.

REFERENCES

[1] Sietsma, J.—Dow, R. J. F.: Creating Artificial Neural Networks That Generalize. Neural Networks, Vol. 4, 1991, pp. 67–79.

[2] Miller, G. F.—Todd, P. M.—Hegde, S. U.: Designing Neural Networks using Genetic Algorithms. In J. David Schaffer, editor, Proceedings of the Third International Conference on Genetic Algorithms, pages 379–384, San Mateo, California, 1989. Philips Laboratories, Morgan Kaufman Publishers, Inc.

[3] Harp, S. A.—Samad, T.—Guha, A.: Designing Application-Specific Neural Networks Using the Genetic Algorithm. In D. S. Touretzky, editor, Advances in Neural Information Processing Systems 2, pages 447–454. Morgan Kaufman, 1990.

[4] Gruau, F.: Genetic Synthesis of Boolean Neural Netorks with a Cell Rewriting Developmental Process. Proceedings of COGANN-92 International Workshop on Combinations of Genetic Algorithms and Neural Networks, pp. 55–74, IEEE Computer Society Press, 1992.

[5] Gruau, F.: Automatic Definition of Modular Neural Networks. Adaptive Behavior, Vol. 2, 1995, No. 3, pp. 151–183.

[6] Billings, S. A.—Zheng, G. L.: Radial Basis Function Network Configuration Using Genetic Algorithms. Neural Networks, Vol. 8, 1995, No. 6, pp. 877–890.

[7] Yao, X.—Lin, Y.: A New Evolutionary System for Evolving Artificial Neural Networks. Transactions on Neural Networks, Vol. 8, 1997, No. 3, pp. 694–713.

[8] Hussain, T. S.—Browse, R. A.: Attribute Grammars for Genetic Representations of Neural Networks and Syntactic Constraints of Genetic Programming. In AIVIGI '98: Workshop on Evolutionary Computation, 1998.

[9] Ros, P.—Schulenburg, S.—Marín-Blazquez, J. G.—Hart, E.: Hyper-Heuristics: Learning to Combine Simple Heuristicsin Bin-Packaging Problems. In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2002), 2002.

[10] Handbook of Metaheuristics Series: International Series in Operations Research and Management Science, Vol. 57, Glover, Fred W.; Kochenberger, Gary A. (Eds.) 2003, 570 p., Hardcover, ISBN: 1-4020-7263-5, Kluwer Academic Publishers.

[11] Montana, D. J.—Davis, L. D.: Training Feedforward Networks Using Genetic Algorithms. In Proceedings of the International Joint Conference on Artificial Intelligence, San Mateo, CA, 1989. Morgan Kaufmann.

[12] Alba, E.—Aldana, J. F.—Troya, J. M.: Fully Automatic RNA Design: A Genetic Approach. X. Yao: Evolutionary Artificial Neural Networks 45. In Proc. of

Int'l Workshop on Artificial Neural Networks (IWRNA '93), pages 399–404. Springer-Verlag, 1993. Lecture Notes in Computer Science, Vol. 686.

[13] Lee, S.-W.: Off-Line Recognition of Totally Unconstrained Handwritten Numerals Using Multilayer Cluster Neural Network. IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 18, 1996, pp. 648–652.

[14] Belew, R. K.—McInerney, J.—Schraudolf, N.: Evolving Networks, Using the Genetic Algorithm with Connectionist Learning. Technical Report CSE-CS-174, UCSD, 1990.

[15] Mandischer, M.: A Comparison of Evolution Strategies and Backpropagation for Neural Network Training. Neurocomputing, Vol. 42, 2002, pp. 87–117.

[16] Hwang, M. W.—Choi, J. Y.—Park, J.: Evolutionary Projection Neural Networks. In Proceedings of the 1997 IEEE International Conference on Evolutionary Computation, ICEC '97, (Piscataway, NJ, USA), pp. 667–671, IEEE Press, 1997.

[17] Kim, H. B.—Jung, S. H.—Kim, T. G.—Park, K. H.: Fast Learning Method for Backpropagation Neural Network by Evolutionary Adaptation of Learning Rates. Neurocomputing, Vol. 11, 1996, No. 1, pp. 101–106.

[18] Krogh, A.—Hertz, J.—Palmer, R. G.: Introduction to the Theory of Neural Computation. Addison-Wesley Publishing Company, Redwood City, CA, 1991.

[19] Yao, X.: Evolving Artificial Neural Networks. Proceedings of the IEEE, Vol. 87, 1999, No. 9, p. 1423–1447.

[20] Garis, H.—Fogel, D.—Conrad, M.—Yao, X.: Special Issue on Evolutionary Neural Systems. Neurocomputing, Vol. 42, 2002, pp. 1–8.

[21] Ash, T.: Dynamic Node Creation in Backpropagation Networks. ICS Report 8901, The Institute for Cognitive Science, University of California, San Diego (Saiensu-sh, 1988), 1988.

[22] Caudell, T. P.—Dolan, C. P.: Parametric Connectivity: Training of Constrained Networks using Genetic Algorithms. Proc. of the third International Conference on Genetic Algorithms and their Applications, pp. 370–374. Morgan Kaufman, 1989.

[23] Fogel, D. B.—Fogel, L. J.—Porto, V. W.: Evolving Neural Network, Biological Cybernetics, Vol. 63, 1990, pp. 487–493.

[24] Schaffer, J. D.—Caruana, R. A.—Eshelman, L. J.: Using Genetic Search to Exploit the Emergent Behaviour of Neural Networks. Physica D, Vol. 42, 1990, pp. 244–248.

[25] Kitano, H.: Designing Neural Networks using Genetic Algorithms with Graph Generation System. Complex Systems. Vol. 4, 1990, pp. 461–476.

[26] Molina, J. M.—Torresano, A.—Galván, I. M.—Isasi, P.—Sanchis, A.: Evolution of Context-Free Grammars for Designing Optimal Neural Networks Architectures. GECCO 2000, Workshop on Evolutionary Computation in the Development of ANN, USA, July 2000.

[27] Guinea, M. A.—Gutierrez, G.—Galván, I.—Sanchis, A.—Molina, J. M.: Generative Capacities of Grammars Codification for Evolution of NN Architectures. Congress on Evolutionary Computation, CEC 2002, IEEE World Congress on Computational Intelligence (IJCNN, FUZZ-IEEE, ICEC), Hawai, July 2002.

[28] Mjolsness, E.—Sharp, D. H.—Alpert, B. K.: Scaling Machine Learning and Genetic Neural Nets. Advances in applied mathematics, Vol. 10, 1989, pp. 137–163.

[29] Merril, J. W. L.—Port, R. F.: Fractally Configured Neural Networks. Neural Networks, Vol. 4, 1991, pp. 53–60.

[30] Valls, J. M.—Galván, I. M.—Molina, J. M.: Multiagent System for Designing Optimal Radial Basis Neural Networks. Information Processing and Management of Uncertainty in Knowledge Based Systems, Spain, 2000.

[31] Wolfram, S.: Theory and Applications of Cellular Automata. World Scientific, Singapore, 1988.

[32] Gutiérrez, G.—Isasi, P.—Molina, J. M.—Sanchís, A.—Galván, I. M.: Evolutionary Cellular Configurations for Designing Feed-Forward Neural Networks Architectures. Lecture Notes in Computer Science, Springer-Verlag, Computer Science Vol. 2084/2001, Connectionist Models of Neurons, Learning Processes and Artificial Intelligence: 6th International Work-Conference on Artificial and Natural Neural Networks, IWANN 2001, Granada, Spain, June 13–15, 2001, Proceedings, Part I, Editors: J. Mira, A. Prieto.

[33] Cybenko, G.: Approximation by Superposition of a Sigmoidal Function. Mathematics of Control, Signals and Systems, Vol. 2, 1989, pp. 303–314.

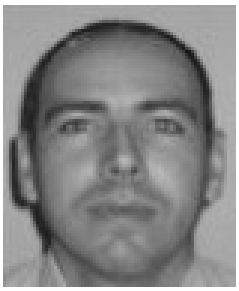
[34] Minor, J. M.: Parity with Two Layer Feedforward Nets. Neural Networks, Vol. 6, 1993, pp. 705–707.

[35] Sontag, E. D.: Feedforward Nets for Interpolation and Classification. Journal of Computer and System Sciences, Vol. 45, 1992.

[36] Blake, C. L.—Merz, C. J.: UCI Repository of Machine Learning Databases. 1998, available on `http://www.ics.uci.edu/ mlearn/MLRepository.html`, Irvine, CA: University of California, Department of Information and Computer Science.

German Gutiérrez received his master degree in physics from Universidad Complutense de Madrid in 1999. Currently he is a Ph. D. student in computer science and teaching assistant of the Computer Science Department of the Carlos III of Madrid University. He is also a member of the Complex Adaptive Systems group. His current research includes artificial neural networks and evolutionary computation.

Araceli Sanchis received her Ph. D. in computer science from Universidad Politecnica de Madrid in 1998. She is associate professor of the Computer Science Department of the Carlos III of Madrid University. She is also a member of the Complex Adaptive Systems group. Her current research includes the aplication of soft computing techniques (NN, evolutionary computation, fuzzy logic and multi-agent systems) to engineering problems such as robot control and vision.



Pedro Isasi received his Ph. D. in computer science from Universidad Politecnica de Madrid in 1994. He joined the Computer Science Department of Carlos III de Madrid University in 1991; he is associate professor of that department from 1997, and full professor from 2001. He is the head of the Neural Network and Evolutionary Computation Laboratory. His current research includes the application of soft computing techniques (NN, evolutionary computation, fuzzy logic and multi-agent systems) to engineering problems such as power plant control, robot control, cryptography or finances, mainly in the domains of prediction, classification, optimization and time series.

José M. Molina received his Ph. D. in telecommunication engineering from Universidad Politecnica de Madrid in 1997. He was a member of the System, Signal and Radio Coomunications group of the Universidad Politecnica de Madrid from 1992. He is associate professor of the Computer Science Department of the Carlos III of Madrid University, and a member of the Complex Adaptive Systems group. His current research includes the application of soft computing techniques (NN, evolutionary computation, fuzzy logic and multi-agent systems) to engineering problems such as RADAR, robot control and vision.

Inés M. Galván received a doctorate-fellowship, as research scientist, in the European Commission, Joint Research Centre Ispra (Italy) from 1992 to 1995. She received her Ph.D. in computer science at Universidad Politecnica de Madrid (Spain) in 1998. She has joined the Computer Science Department at the University Carlos III of Madrid in 1995 and she is associate professor of that department from 2000. Her current research includes artificial neural networks and other soft computing techniques such as evolutionary computation and multi-agent systems. Her research interests also include application fields such as time series prediction and control of dynamic process.