Grammars and Cellular Automata for Evolving Neural Networks Architectures

J.M. Molina, I. Galván, P. Isasi, A. Sanchis
Departamento de Informática. Universidad Carlos III de Madrid.

ABSTRACT. The class of feed-forward neural networks trained with back-propagation admits a large variety of specific architectures applicable to approximation pattern tasks. Unfortunately, the architecture design is still a human expert job. In the recent years, the interest to develop automatic methods to determine the architecture of the feed-forward neural network has increased, most of them based on evolutionary computation paradigm. From this approach, some perspectives can be considered, at one extreme, every connection and node of architecture can be specified in the chromosome representation using binary bits. This kind of representation schemes is called the direct encoding scheme. In order to reduce the length of the genotype, the search space, and to make the problem more scalable, indirect encoding schemes have been introduced. An indirect scheme under a constructive algorithm, by the other hand, starts with a minimal architecture and new levels, neurons and connections are added, step by step, by some sets of rules. The rules and/or some initial conditions are codified into a chromosome of a Genetic Algorithm. In this work, two indirect constructive encoding schemes based on Grammars and Cellular Automata, respectively, are proposed to find the optimal architecture of a feed-forward neural network.

1. INTRODUCTION

The Neural Networks (NN) architecture design is the "key" of the later efficient application of the net. Besides, still today is a process basically guided by the expert's knowledge about the system. These experts are guided by their previous experience and by a process of trial and error for deciding the best architecture for a given problem. The design of the architecture of a neural network can be seen as a search problem within the space of architectures, where each point of the search space is a defined architecture. Evidently, this search space is huge and the task of finding the simplest network that solves a given problem is a tedious and long task.

In the last years, many works have been centered toward automatic resolution of the design of neural networks architecture [1,2,3,4,5,6]. Two representation approaches exist to find the optimum net architecture using Genetic Algorithm (GA): one based on the complete representation of all the possible connections and 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 GA, and starts with Ash's work [7] and continues with other works, including [8].

The second one is the called Indirect Encoding Method that consists on codifiying, not the complete network, but a compact representation of it [1]. In 1990, Kitano [9] presents a new method for designing neural networks by means of Genetic Algorithms, where neural networks are represented throught graph grammars, codified in the chromosomes of the individuals. The grammatical approach is not the only proposed representation method. Merril et al. [10] introduced a fractal representation for encoding the architectures, arguing that this representation is more related with biological ideas than constructive algorithms. Valls et al. [11] proposed a Multiagent System to find optimal architectures in Radial Basis Neural Networks.

One of the most important lacks of Kitano's method is that for an NN of *"n"* neurons, an *"m x n"* matrix is needed, where *"n"* is the smallest power of 2 bigger than *"n"*, and only the upper triangle of the *"m x n"* matrix is used, which also decreases the efficiency of the encoding. In Kitano's method, the recursion is not included [5]. In this work, one of the first objectives is to extend and improve Kitano's method to make a general and powerfull method of designing NN, in particular Multilayer Perceptrons with a hidden layer, and bakpropagation algorithm [12]. For the Multilayer Perceptrons considered, a mechanism of representing NN by means of graph grammars, without restrictions in the matrix size and with recursion, has been developed. This grammatical mechanism is integred in a complete system, called GANET, for the design of NN applied to a considered problem.

The second method is based on Cellular Automata (CA) to find the optimal architecture of a feed-forward neural network, also trained with backpropagation. In this scheme, positions of several seeds in a bidimensional grid are represented in the chromosome. These seeds are used as the initial configuration of a cellular automata which is translated into a feed-forward neural network through a growing procedure as the cellular automata is evolved. Thus, the chromosome length is significantly reduced and the scalability of the method is increased.

These two new methods have proven to automatically design the architecture of an NN to forecast one step ahead in the logistic chaotic time series. The special features of this series make the problem a good test bed for testing the two methods.

2. GRAMMATICAL APPROACH

The GANET system is composed of three modules, following the general schema proposed by Kitano [9]. In Figure 1, the architecture of the system is shown.

0-7803-6583-6/00/$10.00 © 2000 IEEE

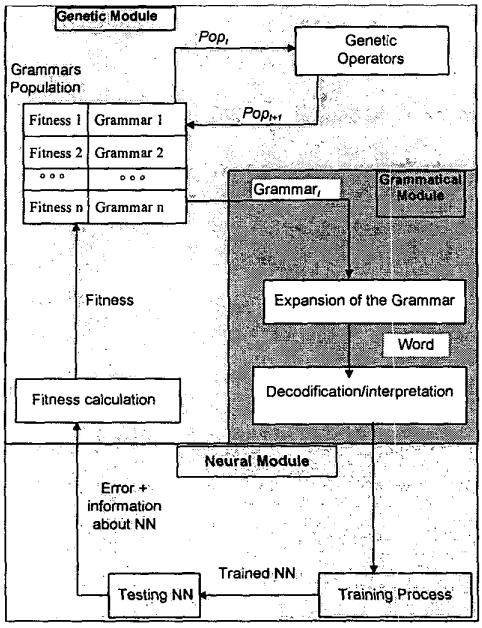


Figure 1: GANET architecture: Genetic, Neural and Gramatical modules.

The scheme shown in Figure 1 is cycled, and each cycle corresponds to a GA generation. By means of genetic operators, a population of grammars is obtained, except the first one that, as is usual [13], is randomly generated. These new individuals are evaluated through a fitness function. To calculate the fitness, the grammar codified in the individual chromosome is expanded, and a bidimensional matrix (word) is obtained. In a next step, this word is decodified as an NN matrix of connections. This matrix is transformed into an NN architecture that is trainned. After training the NN, it is tested and an error value is obtained. With this error and some other relevant information about the NN (size, learning cycles, etc.), the fitness value of the considered individual is computed. The process is repeated until all population is evaluated.

Grammatical Module

In this work, Context Free Chomsky Grammars, G2, have been employed [14,15], particularly bidimensional or graph ones [16]. These grammars are defined as a quintuple, $G = \{\Sigma_T, \Sigma_N, S, P, EM\}$, where Σ_T is the alphabet of terminals, Σ_N is the alphabet of non-terminals, S is the start symbol, $S \in \Sigma_N$, P is the production or rules set and EM is the Expansion Method. This kind of grammars generates words that are matrix of terminal symbols. An example of this kind of grammars is:

$G = \{\Sigma_T, \Sigma_N, S, P, EM\}$, where:

$\Sigma_T = \{0,1\}$ alphabet of terminals

$\Sigma_N = \{A,B,C,D\}$ alphabet of non-terminals

S = start symbol, $S \in \Sigma_N$

P = production or rules set = {

$$A ::= \begin{pmatrix} B & D \\ D & D \end{pmatrix}, \quad B ::= \begin{pmatrix} 0 & C \\ 0 & 0 \end{pmatrix},$$

$$C ::= \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}, \quad D ::= \begin{pmatrix} 0 & 0 \\ 1 & 1 \end{pmatrix} \}$$

EM = Expansion Method = {Up and Right,Up and Left, Down and Right, Down and Left}.

In order to carry out the derivation process in a bidimensional context free grammar and to generate words, sometimes it is necessary to insert a row, a column, both row and column or neither. The position of the space inserted depends on the position of the symbol and the expansion method. In this work, the expansion method employed is the Up and Right one. For generating a word useful for GANET system, a special derivation mechanism has been developed. This mechanism keeps the rectangularity of words during the derivation, inserting an auxiliary symbol that produces, when necessary, the space required before producing the desired derivation. For the Up and Right expansion method, the inserted row will be placed over the symbol and the inserted column will be placed at the right. In the production rule of Figure 2, the insertion of the row and column is made in first place, and the non-terminal symbol (in Figure 2 is the symbol B) is derived later.

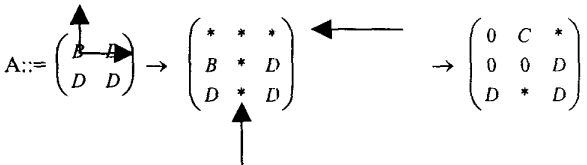


Figure 2: Derivation of non-terminal symbol B in a bidimensional production rule of a G2.

Then, the derivation sequence of all the non-terminal symbols is carried out "through levels". Firstly all the non-terminal symbols of the same level (that belong to the same previous derivation process) are derived and, secondly, the non-terminal symbols of next levels. In the example of Figure 2, the symbol D (that belongs to the first level) is derived before the derivation of the symbol C (that belongs to the second level). This derivation order assures that the number of auxiliary symbols is minimum. Finally, to stop the derivation process (when using recursive rules) two parameters ought to be specified: how many times a recursive rule is applied (maximum value of recursion, MNR) and the maximum number of levels with non-terminal symbols (number of levels, NL).

Neural Net Module

The architecture selected for this module is a Multi Layer Perceptron (MLP), with *backpropagation*, forward connections, and with one hidden layer. A hidden layer allows any solution without loosing generality, because is proven that any problem could be solved with only one hidden layer if there is not restrictions in the number of neurons in this layer. The method to obtain the MLP from the word obtained in the previous module is as follows:

1. 0 substitutes auxiliary symbols. Figure 3 shows this step in an example of word.

$$\begin{pmatrix} * & 1 & 1 & * & * \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & * & 1 & 1 \\ 0 & 0 & * & 0 & 0 \\ 1 & 1 & * & 1 & 1 \end{pmatrix} \rightarrow \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 \end{pmatrix}$$

Figure 3: Transformation of auxiliary symbols in 0's.

2. The matrix is divided in three zones, one for each layer. The first zone is the input layer, the second one is the hidden layer and the last one the output layer. The first and the third layer have a constant size defined by the problem. The second zone, the hidden layer, has a variable size and

depends on the word. The three zones appear in rows and column, as it is shown in Figure 44.

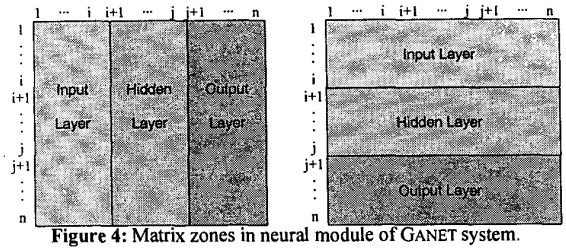


Figure 4: Matrix zones in neural module of GANET system.

Then, the first *i* components of the matrix will correspond with input neurons, the following *j* neurons with the hidden layer and the last (*n-j*) with the ouput layer. Values of *i* and *j* were fixed through the problem to solve and will be kept constant during all the process. However, *n* is not a constant value, so the number of neurons in the hidden layer depends on the concrete grammar applied and this concrete grammar is learned by GANET system. The final number of neurons in each layer is the number of neurons that are conected to other neurons. This number is obtained analyzing the connections that appear in the matrix of Figure 3 following the division of Figure 44 at it is showed in Figure 5.

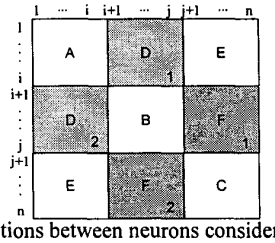


Figure 5: Conections between neurons considering three layers in the Neural Network.

Each zone in Figure 5 corresponds to a different type of connection:

- Zone 'A': connections between neurons of the input layer.
- Zone 'B': connections between neurons of the hidden layer.
- Zone 'C': connections between neurons of the output layer.
- Zone 'D' (D_1 AND D_2): connections between neurons of the input and hidden layers.
- Zone 'E': connections between neurons of the input and output layers.
- Zone 'F' (F_1 AND F_2): connections between neurons of the output and hidden layers.

In the proposed method, only zones D and F (Figure 5) are considered. The position (x, y) with a value of 0 in the matrix means that x and y neurons are not connected, a 1 value will mean that a connection exists. In order to evaluate the existence of a connection between two neurons a 1 value must appear in the two subzones (D_1 and D_2, F_1 and F_2). The logical AND operator has been used because the meaning of D_1 (F_1) is the same than the D_2 (F_2). An example is shown in Figure 6.

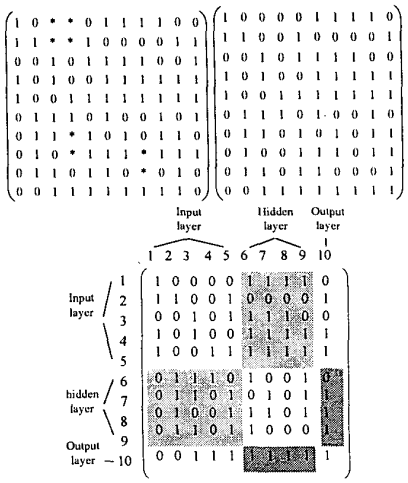


Figure 6: Example of interpretation of a matrix to obtain a Neural Network.

The NN interpreted from Figure 6 is shown in Figure 7.

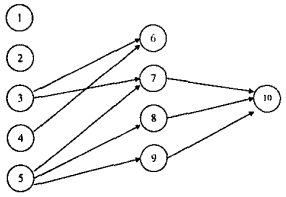


Figure 7: Neural Network obtained from Figure 6.

Genetic Module

GANET works with a population of grammars, so the chromosome codifies in binary a bidimensional grammar. The codification represents the five elements of a bidimensional grammar; in this work the expansion method is always the same (Up and Right), then only four elements are codified (Σ_T, Σ_N, S, P). The right side of the production rules is represented in the chromosome, as shown in Figure8 , where T represents terminal symbols and NT the non terminal ones. The genotype of Figure 8 represents grammars with four different NT symbols, and has been used in this work.

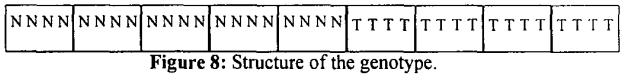


Figure 8: Structure of the genotype.

Each part of genotype shown in Figure8 is the right part of a production rule, in the same sequence that appears in the grammar definition, as shown in Figure 9.

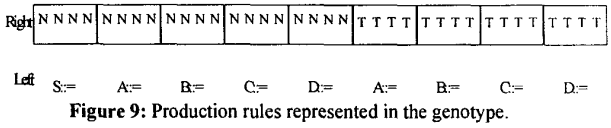


Figure 9: Production rules represented in the genotype.

The first four rules contain non-terminal symbols and the last ones are composed only by terminal symbols. Finally, the codified chromosome needs two bits for representing non-terminal symbols (four non-terminal symbols) and one bit for terminal ones (two terminal symbols, 0 and 1). In this work four non terminal symbols have been used because the first

production rule in these graph grammars (S ::= anything) has four non terminals in the right part. More non-terminal symbols could be considered only increasing the size of the genotype.

3. CELLULAR AUTOMATA APPROACH

The Cellular Automata approach is composed by three modules: the GA, the CA and the module responsible of neural network training, as is shown in Figure 10.

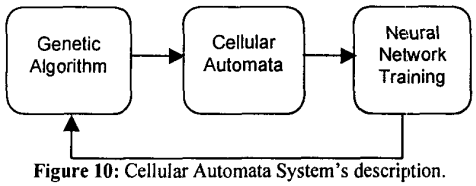


Figure 10: Cellular Automata System's description.

Cellular Automata Module

For generating neural networks architectures, a bidimensional CA has been included. The size of the bi-dimensional grid of the automata depends on the number of input and output neurons given by the problem and on the maximum number of hidden neurons to be considered. Let Dim_x be the number of rows in the grid, its value is equal to the number of input neurons plus the number of output neurons in the problem. On the other hand, Dim_y is the number of columns in the grid and corresponds with the maximum number of hidden neurons to be considered. Each cell in the grid could be in different states, active, inactive, or occupied by a seed. Two different kinds of seeds have been introduced: growing seeds and decreasing seeds. The growing seeds could be different $(s_1, s_2, ..., s_m)$, while the decreasing seeds are all the same (d). The number of seeds of each type is a parameter of the method. Each seed type corresponds with a different type of rule, so there are two rules called growing rule and decreasing rule respectively.

Growing rules: This type of rule reproduces a particular growing seed when there are at least three identical growing seeds in its neighbourhood. There are different configurations, growing seeds located in: rows, columns, or in a corner of the neighbourhood (see Figure 11). In Figure 11 and Figure 12 some of those rules are shown (the others are symmetrical). Here *s* is a specific growing seed, *d* is a decreasing seed, *i* is an inactive state for the cell and *a* means that the cell could be in any state or contains any type of seed (even a decreasing seed).

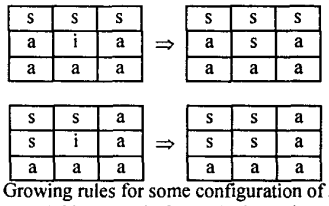


Figure 11: Growing rules for some configuration of seeds in the neighbourhood of a particular cell.

Decreasing rules: The growing rules previously specified allow obtaining feed-forward neural networks with a large number of connections. In order to remove connections in the network, decreasing rules are included in the system. These rules deactivate a cell in the grid when the cell has a decreasing seed and two cells in a row of its neighbourhood contain also a decreasing seed. One situation in which the decreasing rules can be applied is shown in Figure 12, the other rules can be obtained symmetrically.

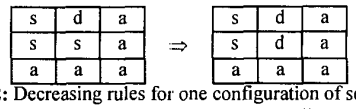


Figure 12: Decreasing rules for one configuration of seeds in the neighbourhood of a particular cell.

The CA is expanded as follows:

1) The growing seeds are located in the grid.
2) An expansion of the growing seeds takes place. This expansion consists on replicating each seed in turns, on its 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.
3) The growing rules are applied until no more rules could be fired.
4) The decreasing seeds are placed in the grid. If there are some other seeds in those places, they are replaced.
5) The decreasing rules are applied until the final configuration is reached.

Neural Net Module

As in the previously described method, an MLP with backpropagation has been selected. To relate the grid of the automata with an architecture of a neural network, the following meaning for a cell in the grid *(x,y)* is defined: if $x \leq n$, -with *n* the number of input neurones- *(x,y)* represents a connection between the *x-th* input neuron and the *y-th* hidden neuron; if $x > n$, *(x,y)* represents a connection between the *y-th* hidden neuron and the *(x-n)-th* output neuron. The conversion procedure is then as follows:

1) Obtaining a binary matrix *M*.- The places of the grid in which a growing seed appear, are set to 1 and the places corresponding with inactive cells or decreasing seeds are set to 0. Thus, a *Dim1xDim2* binary matrix, $M=(m_{ij})$, is obtained.
2) Processing the binary matrix *M*.- In the above binary matrix the one value is interpreted as a connection, and the zero value as the absence of connection, the rows and columns in the matrix with values 0 are removed. A new and shorter matrix , called processed matrix (*PM*), is obtained.
3) Generating the feed-forward neural network.- For the *PM* matrix, if $pm_{ij} = 1$ then a connection between the *i-th* input neuron and the *j-th* hidden neuron is created, or between the *j-th* hidden neuron and the *(i-n)-th* output neuron, as is previously described. If $pm_{ij}=0$, there do not exist connection between that neurones.

Genetic Module

The size of the chromosomes in the GA corresponds with the number of seeds, and it codifies all the possible locations of seeds in the grid. Chromosomes have been codified in base *b*, where *b* is the number of rows in the grid: the number of inputs plus the number of outputs. Each seed is determined by a coordinate *(x,y)*:

1) The first coordinate could be represented by an unique gen, indicating the row in which the seed is located.
2) The second coordinate will need more than one gen, if as usual the maximal number of hidden neurons is bigger than *b*.

In this case, two genes have been used to codify the *y* coordinate, what allows a maximum of *(b-1)xb+b* hidden neurons. For instance, if there are 3 inputs and 2 outputs, the

maximum size of the hidden layer is: 4x5+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.

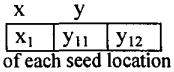


Figure 13: Codification of each seed location in the GA chromosome structure.

The chromosome will have 3 genes (Figure 13) for each seed to be placed in the grid, firstly the growing seeds are represented and finally the decreasing seeds. The number of seeds of each type has to be determined previously, and is a parameter of the method. If, for instance, there are five growing seeds, and five decreasing seeds, the size of the chromosome is 30, divided in ten trios of gens, the first five trios representing the growing seeds, and the final five trios representing the decreasing seeds. It is important to notice that the growing seeds belong to different types of seeds, while the decreasing seeds are all of the same type, as mentioned in the description of the automata rules.

The GA module takes charge of generating initial configurations of the cellular automata, and to optimize these configurations from the information obtained from the training module. The CA takes the initial configuration and generates a final configuration corresponding to an NN architecture. Finally, the generated architecture has to be trained and evaluated for a particular problem and the error is used as the fitness value for the GA.

4. EXPERIMENT FOR LOGISTIC TIME SERIES

These two methods have been applied to determine the simplest feed-forward neural network able to approximate the logistic time series. The logistic map is given by equation 1.

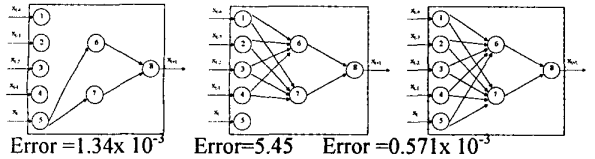
$$x_{t+1} = \lambda \cdot x_t \cdot (1 - x_t)$$

Where λ values determine the behavior of the serie. In this work a value of λ=3.97 and x(0) = 0.5 are used, and the map describes a strongly chaotic time series. The use of the logistic map has two main advantages:

- The chaotic behavior and its prediction is a non trivial task
- The optimal NN to predict the map is known. As the value of the map at instant *t* deppends only on the value of the map at instant *t-1*, the optimal network has to consider the input carrying the *t-1* signal.

In order to increase the complexity of the problem and to test the ability of the methods to generate good architectures, five input have been taken into account: x_{t-4}, x_{t-3}, x_{t-2}, x_{t-1}, x_t. Thus, the system must find the most relevant input variables in the dynamic behavior of the logistic time series, besides the minimum number of hidden neurons to solve the problem.

Several "a priori" nets have been trained with 95 patterns during 700 cycles. The results of testing (see Figure 14) show that the input x_t is necessary to solve the problem and that the other inputs allow to obtain better error values, but not in a significant amount.



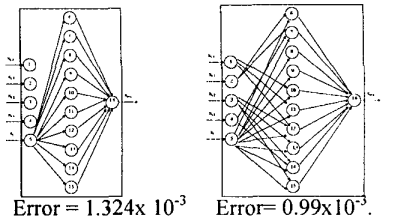


Figure 14: Testing errors of several NN for logistic time series.

The fitness value employed considers the accuracy of the NN that is given by two values: testing error and size of the NN. A fitness function that only considers the error does not allow finding the simplest NN. Then, the fitness function used in this work is given by equation 1. Where: p is the test patterns number, S_j is the network output for pattern j, d_j is the desired output for pattern j, *nconex* is the number of connections and k is a parameter that weights the influence of the net size in the fitness function. In this work the experimental value of k is 1.1.

$$fitness = \frac{1}{error} \cdot \frac{1}{size} = \frac{p}{\sum_{j=1}^{p} (S_j - d_j)^2} \cdot \frac{1}{k^{nconex}} \quad (1)$$

The experimental parameters in both methods are:

- 400 learning cycles.
- 200 generation of GA.
- Population size 50 individuals.
- Parent population 35 individuals.
- Elitism percentage 10%, increased a 2% each 40 generations.
- Mutation 1%.
- k=1.1
- 95 patterns (100 iterations of logistic series with x_0=0.5 and λ=3.97 with six decimals)
- Learning factor α=0.9.

The NN obtained by GANET and CA has two neurons in the hidden layer and a total of five connections (see Figure15). The error is 0.00129 and the fitness value is 481.26.

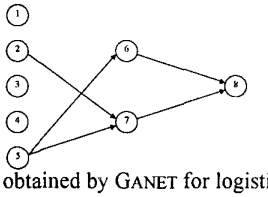


Figure15: NN obtained by GANET for logistic time series.

5. CONCLUSIONS

The election of good neural networks architectures is an important step in many problems where there is few knowledge about the problem itself. Evolutionary computation techniques are good approaches for automatically generate those good architectures. However the codification of the network is a crucial point in the success of the method. Direct codification's become inefficient from a practical point of view, making bigger and redundant the search space. To solve this problem an indirect encoding has to be used.

Indirect encoding is driven to reduce the search space in such a way that similar solutions are eliminated and represented by only one representative. In these cases, the codification makes the method able to find better architectures.

The first proposed method is a succesful extension of Kitano's works [9]. An automatic procedure to design neural networks architectures has been developed by means of the bidimensional grammar evolution. The considered grammars allow overcoming Kitano's restrictions that affect to the grammar contents, the architecture of NNs and the genetic operators. In this work these three restrictions are elimated. The grammars include recursivity. The number of NN nodes (constant in Kitano's work, determined through the word size and fixed also in the genotype) is evolved by GANET system in an optimal way (the size of the final word is variable). The genetic operators, in particular crossover, are generic and is not necessary to redefine them. Moreover, this generalization of Kitano's work extends the capability of the automatic procedure without loosing the quality of the obtained nets and without dismissing the results of GA.

The genetic evolution of bidimensional grammars (codifying the architecture of an NN) reduces the size of chromosomes and improves the convergence of the algorithm. The grammar codification does not reduce the search space and is more efficient than direct encoding methods.

In the second method, CA's are good candidates for non-direct codifications. The constructive representation introduced in this work also solves some of the problems for non-direct codifications. The final representations have a reduced size and could be controlled by the number of seeds used.

The results with the logistic map show how the complexity of the achieved network could be reduced in spite of the fitness function used. These methods have also been able to find the appropriate network for the logistic map, identifying the only useful input. This result could have a great practical interest because some times the identification or importance of the inputs is a primordial step in the solution of a problem.

6. REFERENCES

[1] S. Harp, Samad T. and Guha A. Towards the Genetic Synthesis of Neural Networks. Proceedings of the Third International Conference on Genetic Algorithms and their applications, pp 360-369, San Mateo, CA, USA, 1989.

[2] G.F. Miller, P.M. Todd and S.U. Hegde. Designing neural networks using genetic algorithms. In Proc. of the third international conference on genetic algorithms and their applications, pp 379-384, San Mateo, CA, USA, 1989.

[3] S. Harp, Samad T. and Guha A. Designing Application-Specific Neural Networks using the Genetic Algorithm, Advances in Neural Information Processing Systems, vol2, 447-454, 1990.

[4] F. Gruau. Genetic Synthesis of Boolean Neural Networks with a Cell Rewriting Developmental Process. Proc. of COGANN-92 International Workshop on Combinations of Genetic Algorithms and Neural Networks, pp. 55-74, IEEE Computer Society Press, 1990.

[5] F. Gruau. "Neural Network Synthesis Using Cellular Encoding and the Genetic Algorithm". PhD Thesis, Ecole Normale Supèrieure de Lyon, (1994).

[6] F. Gruau. Automatic Definition of Modular Neural Networks. Adaptive Behaviour, vol. 2, 3, 151-183, 1995.

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

[8] D.B. Fogel, Fogel L.J. and Porto V.W. Evolving Neural Network, Biological Cybernetics, 63, 487-493, 1990.

[9] H. Kitano. Designing Neural Networks using Genetic Algorithms with Graph Generation System, Complex Systems, 4, 461-476, 1990.

[10] J.W.L. Merril and R.F. Port. Fractally configured Neural Networks. Neural Networks, 4, 53-60, 1991.

[11] Valls J.M., Galván I.M. and Molina J. M. (2000),"Multiagent System for designing optimal Radial Basis Neural Networks", Information Processing and Management of Uncertainty in Knowledge Based Systems. Spain.

[12] Hartz J., Krough A. and R.G. Palmer (1991). "Introduction to the Theory of Neural Computation". Addison-Wesley.

[13] Goldberg D.E. (1989). Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, New York

[14] Chomsky N. (1959)."On Certain Formal Properties of Grammars", Information and Control 2,137-167.

[15] Hopcroft J.E. and Ullman J.D. (1979). Introduction to Automata Theory, Languages and Computation, Addison-Wesley.

[16] Ehring, Nagel, Rozemberg and Rosenfeld (1987)."Graph Grammars and their Applications to Computer Science", Lecture Notes in Computer Science.