Evolving Neural Arrays
A new mechanism for learning complex action sequences

Lic. Leonardo Corbalán, Lic. Laura Lanzarini
Universidad Nacional de La Plata, Facultad de Informática,
La Plata, Argentina, 1900
{corbalan,laural}@lidi.info.unlp.edu.ar

Abstract

Incremental evolution has proved to be an extremely useful mechanism in complex actions sequence learning. Its performance is based on the decomposition of the original problem into increasingly complex stages whose learning is carried out sequentially, starting from the simplest stage and thus increasing its generality and difficulty.

The present work proposes neural array applications as a novel mechanism for complex actions sequence learning. Each array is composed by several neural nets obtained by means of an evolving process allowing them to acquire various degrees of specialization. Neural nets constituting the same array are organized so that, in each assessment, there is only one in charge of its response.

The proposed strategy is applied to problems presented by obstacle evasion and target reaching as a means to show the capability of this proposal to solve complex problems. The measurements carried out show the superiority of evolving neural arrays over traditional neuroevolving methods that handle neural network populations – SANE is being particularly used as a comparative reference due to its high performance.

Neural array capability to recover from previous defective evolving stages has been tested, evincing highly plausible final successful outcomes – even in those adverse cases.

Finally, conclusions are presented as well as some future lines of work.

Key Words: Evolving Neural Nets, Learning, Complex Actions Sequence Learning, Incremental Evolution, Genetic Algorithms.

1 Introduction

Evolving Neural Networks (ENN) are a particular case of artificial neural networks (ANN), where the adaptation is carried out by training and, mainly, by evolution [15]. Evolution has been applied to define several aspects of the neural network: to obtain connection weights, architecture designs, initial parameters values, learning rules, etc. [16]. The common practice implies taking the best individual of the last generation leaving aside all those neural networks that have an inferior fitness value.

However, from the biological point of view, the brain's activity does not seem to come up from a single neural network. The studies carried out in humans by Jackson and Penfield in the '50s, revealed the existence of a significant functional allocation in the cerebral cortex. Later, other neuronal centers have been discovered in the brain related to emotions and the intellectual activities [13]. In this way, it is likely that the brain is composed by many different parallel and distributed systems that carry out well–defined functions. Thus, the emulation of the human brain by means of several artificial neural networks might be suitable.

Freeman and Skapura [4] argued the need to learn to combine small ANNs, putting them under the control of other networks in order to solve the scalability problem. X. Yao and Y. Liu [14] studied the advantages of using the complete population of neural networks obtained in the last generation –result of an evolutionary process– instead of that of better fitness. More recently, Bruce and Miikkulainen [1], working on the problem of free–hand written characters recognition, demonstrated that all the neural networks of a population, in combination with an effective specialization technique, could respond better collectively than individually.

Finally, Gómez and Miikkulainen demonstrated that incremental evolution could be successfully applied in the resolution of complex problems [6].

2 Objective

Making a relation among the researches previously mentioned, this paper proposes a new methodology to solve complex problems based on evolving neural networks. During the evolution, each individual of the population is composed by more than one neural network, and tries to solve the problem with the collective participation of its members and some functional division of the task. Learning is based on an incremental strategy that evaluates the population at several stages, upon different tasks on increasing complexity, until it reaches the final expected behavior. Section 3 describes in detail the proposed method.

Incremental evolution is applicable to problems that can be naturally decomposed in a sequence of tasks of increasing complexity [6].

Problems such as obstacle evasion and target reaching represent a real challenge for the evolving solutions since the behavior to be learnt varies according to the environment characteristics. For this reason, they have been chosen to measure the results of this proposal.

3 Evolving Neural Arrays (ENA)

ENA simulates the brain's proved functional allocation where various neural circuits related to cognitive, sensorial and motor functions are found. For this, an artificial neural network array population is evolved. Networks constituting an array learn to specialize themselves in different aspects of a problem and to delegate unrelated issues to their mates. In this way, from their coordinated behavior within an array, the solution of a complex problem arises more efficiently.

3.1 Neural Arrays Internal Organization

A neural array is an n–tuple of neural networks in the form $A=(nn_1,nn_2,..nn_n)$. The notion of empty array is defined as that that does not contain any element and is denoted as **( )**. The operator ***ins*** is also defined, which inserts a neural network as first element of a neural array and is denoted with the sign "**:**"

$$nn : (\,) = (nn) \qquad\qquad nn : (\, nn_1, nn_2, \ldots, nn_k\,) = (nn, nn_1, nn_2, \ldots, nn_k\,)$$

A neural array accepts an input data, evaluates it, and produces the corresponding output Like the brain, which activates precise cortical regions to process diverse kinds of information, neural arrays select a single network to produce output in response to a given stimulus. Thus, it is necessary to endow these structures with some mechanism that will perform this selection. Different alternatives have been analyzed and a decentralized solution was eventually chosen, in which each network self–qualifies itself with a confidence level so as to render a response to a given input.

In this way, array's networks have an extra output neuron in addition to neurons required by the problem, which takes part in the selection process. For each input data, this neuron produces a real number in the interval [0 , 1]. This value is considered the confidence level with which the network computes the response. Logically, the networks' self–qualification capacity is subject to evolution. Thus, it is unlikely that a network remains during several generations if it is incorrectly self–qualified.

The neural array's output is obtained by evaluating sequentially its member networks according to its position in the array, from left to right and with the same input data, until one with a confidence level higher than a given threshold (e.g. , 0.5) is found. This network will be used to compute the output. The last network of the evaluation sequence can do without the extra output neuron since, if it is chosen, it will have to provide the response anyway. The reason for this evaluation order is presented below, since it arises from the details in the implemented incremental evolution strategy.

Notice that the networks located in the lower positions of the array have a greater priority than the rest to produce the output. This is due to the fact that part of their learning consists in recognizing the moment in which they should delegate the responsibility of providing a response to higher level networks.

3.2 Incremental Learning in Neural Arrays

Incremental evolution is a very useful mechanism to learn complex behaviors. It works based on the following concept: if a population is first evolved in order to make a simpler version *t'*, of the complex task *t*, it is possible to discover a region in the search space from which it will be possible to reach a solution that will successfully solve *t*. If *t'* is still a complex task, it is possible that a simpler task *t''*, will be found from which *t'* will be accessible, and so on.

In this way, the goal task can be reached by incrementally evolving the solutions at a sequence of simpler tasks with growing complexity $\{t_1, t_2, ... , t_n\}$, being t_i easier to solve than t_{i+1} for all i: $0< i <n$, and t_n = goal task.

Consequently, the evolving process is divided in as many stages as intermediate tasks are already defined (Figure 1).

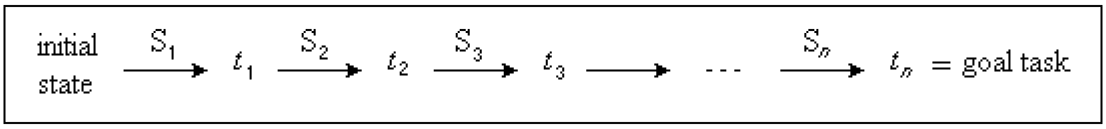


Figure 1

The quantity of necessary intermediate tasks varies in relation to the problem to be solved. Like with other neuroevolving algorithm’s parameters, this value should be determined experimentally.

Incremental evolution on a neural array population uses a mechanism that inserts in the arrays a new neural network at the same time that the evolution advances towards the next task.

In stage S_i, the population made up of arrays of exactly i neural networks evolves towards task t_i. In this way, the evolution starts with a population of arrays of length 1. Once the stage is completed, the best array is saved, and the following stage moves to the next task with the initialization of a new population where each chromosome codifies a neural network. The phenotypes (neural arrays) to be evaluated are created by decoding the chromosome neural network and inserting it as first element of the array saving the previous stage. At any of these stages, the phenotype associated to each chromosome of the population is not a neural network but an array of them partially codified.

During the necessary evaluation to assign fitness, input data are presented to the first network of each array, which is the only member subjected to evolution in this stage. This decides –by means of the extra output neuron– to respond by itself or delegate the task to the previous shaping of the array, passing the control to the second neural network. In this way, array's networks are evaluated from left to right, since this ordering represents the inverse order of their insertion in each of the stages.

Notice that in this way, in stage S_i of the evolving process, the network subject to evolution inserted in an array that already knows how to solve t_{i-1} must learn to delegate the functionality concerning task t_{i-1} and to solve by itself its own situation in order to reach the solution of t_i.

3.3 Evolving Algorithm Used

Let $\{t_1, t_2, ... , t_n\}$, t_n=goal task be the sequence of tasks

The algorithm used to evolve neural arrays is the following:

```
A=( )                                  //empty array
For each stage $S_i$ , with i=1,...,n
    generate randomly a population P of neural networks $P=\{ nn_1, nn_2, ... ,nn_k\}$
    build the population of neural arrays $P_A=\{ nn_1 :A, nn_2 :A, ... ,nn_k :A \}$
    evaluate each neural array of $P_A$ assigning fitness to the individuals of P          (*)
    While $t_i$ is not successfully solved
        P = next_generation(P)                                                          (**)
        build the population of neural arrays $P_A=\{ nn_1 :A, nn_2 :A, ... ,nn_k :A \}$
        evaluate each neural array of $P_A$ assigning fitness to the individuals of P
    End While
    A=best neural array obtained during stage $S_i$
End For
```

where *k* and *n* are parameters.

(*) The fitness value assigned to each neural network nn_i of *P* is that obtained by the array of which it constituted the first element.

(**) ENA does not assume any specific evolving algorithm to obtain the next generation in population *P* of neural networks. A simple genetic algorithm can be used –as that presented by Goldberg [5] – or others more sophisticated. It is usually convenient to use neuroevolving strategies specially designed to be applied to neural networks, such as SANE [9][10][11] or ESP [6][7][8][12], which have proved to outperform traditional algorithms. The adopted genetic codification, the used genetic operators, and the selection, reproduction and replacement mechanisms will depend on the chosen algorithm. Nor any restriction on the parameters of the network subjected to evolution (connection weights, architecture, transference function, etc.) is assumed, either. A set of variables can be found in [16], where several researches on neuroevolution are quoted, including hybrids with traditional learning algorithms.

4 Obstacle Evasion and Target Reaching

Intelligent behavior is expected to be attained in an agent that moves freely in two dimensions within the limits of a virtual environment, of rectangular shape, whose borders it cannot surpass. Objects such as obstacles –which it will have to learn to dodge– and mobile or static targets –which it will have to reach– interact with it in this virtual environment.

A neural array controls the movements of the agent in a temporal interval simulated by the succession of *n* time discrete instants (simulation steps). At each instant, the array is stimulated by a set of input signals. The output is made up of an ordered pair (Δx, Δy) that determines the displacement the agent makes on the surface. However, the present obstacles and the limits of the environment can impede the movement from being fulfilled.

4.1 Agent

Each agent has 13 sensors and an internal clock that work as follows (Figure 2):

- 8 sensors, uniformly distributed around the agent, capable of detecting targets independent of their distance.
- 4 sensors that allow it to detect obstacles, only at a short distance, in four possible directions (N, S, E, and W).
- 1 sensor indicating whether there has been any collision at any point of the agent's circumference.
- 1 internal clock. With this signal, we expect to endow the agents with a certain "conscience" about the course of time. A behavior might become unacceptable if it lasts more than a determined quantity of time. Also, it enables a change in the output even if the inputs coming from outside do not change.

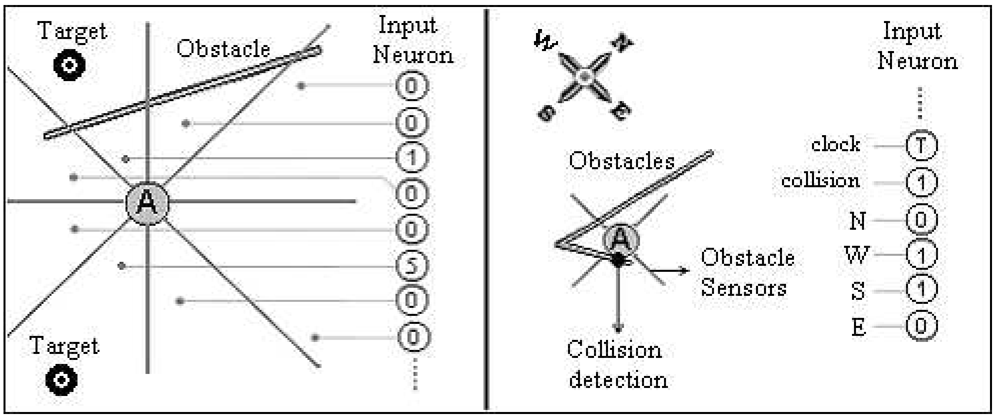


Figure 2

Target sensors provide three possible values (0; 1 or 5) respectively indicating the absence, presence behind some obstacle, or a direct target visualization. In this way, the direct visualization of an object is expected to be more significant, easing a potential change in the behavior of the agent which is capable of reaching this situation. Nevertheless, experiments with other triads of values (0,1,2), (0,1,10) etc. have been carried out with results that have not significantly changed.

Obstacle detectors can be considered as sensitive extension of the touch oriented in the four cardinal directions and, like the collision sensor, they convey two values 1 or 0 (presence or absence of stimulus). The clock value belongs to the real interval (0,1], and constitutes an appraisal of the simulation time, as for the current step p of a n steps simulation, $T = p/n$.

4.2 Fitness Designation

The problem of obstacle evasion and target reaching belongs to a generic class of problems called *Sequential Decision Tasks*. Its main characteristic is the difficulty to properly assign the goodness of an action, a decision sequence being necessary before measuring the effect of any of them. Games such as chess and checkers, among many, fall within this category. Examples of the real world could be: data routing in Internet routers, chemical flow control in a chemical reactor, air-traffic control, etc. In all of these cases, the effect of a simple decision arises when some time has passed, and even then, it is frequently difficult to establish which decisions are responsible, and to what extent, for what has occurred. [10].

Obstacle evasion and target reaching are within the range of this kind of problems and, thus, no error score is carried out until the simulation used to evaluate each individual of the population is finished.

Once the simulation is finished, the neural array fitness controlling the agent is computed as follows: Let $f(a)$ be the performance score assigned to agent a. If the target is not reached, $f(a)$ will take a value of interval [0,100) computed proportionally to the path covered towards the target. On the contrary, if the target is reached, $f(a)$ will belong to interval [100,150). If p is the number of steps used to reach the target, and P, the total number of the simulation steps, then fitness is computed as $f(a)=100+50(P-p)/P$. In this way, a value between 100 and 150 is obtained proportional to the speed with which the target is reached.

The fitness function thus defined rewards those individuals who come closer to the target, and those reaching it faster – if they can do so. As the first generations follow one another, the evolving algorithm will be capable of getting agents which come more and more near to the target. Then, once such target is reached, the same fitness function will enable the optimization of the solutions, obtaining agents that will reach it more rapidly.

5 ENA applied to Obstacle Evasion and Target Reaching

ENA method has been tested over obstacle evasion and static or mobile target reaching (prey capture) that represent *process control* problems difficult to be evolved. Also, they posed the inconveniences typical of *Sequential decision tasks*-type problems, reason why they turn out to be interesting issues to solve.

5.1 Implementation Details

ENA has been implemented by dividing the evolving process in as few stage quantities as possible. Although such number depends on the proposed scenario complexity, the tests carried out show that, in complex situations, the results can be successfully fulfilled in three stages.

$$\begin{tikzcd}
\text{initial state} \arrow[r, "S_1"] & t_1 \arrow[r, "S_2"] & t_2 \arrow[r, "S_3"] & t_3 = \text{goal task}
\end{tikzcd}$$

Figure 3

Each of these stages has a fixed length expressed in generation quantities. The final objective is to control the agents so that, starting from a determined origin, they reach their destiny in as few simulation steps as possible. Some fixed geographical location or some mobile target are considered as destiny.

As example, figure 4 shows a scenario in which the sequence of tasks $\{t_1,t_2\}$ has been defined. The task t_2 (goal task) consists in controlling the agent so that it can reach target, starting at the origin point (2). Task t_1 reduces the complexity of the voyage, starting at an intermediate place of the way between the origin and the target.

Thus, it is expected that in the second stage, the arrays of the form (nn_1,nn_2) will be capable of using nn_1 to lead the agent near the starting point of the previous stage, where the control could pass to nn_2, specialized in the journey from that point towards the final destination. It is worth mentioning that agents do not have any information about the place from where they departed in stage 1. The organization in the control delegation is determined in function of the value of the third output neuron of nn_1 , which solely depends of the input values sensed by the agent in each simulation step.

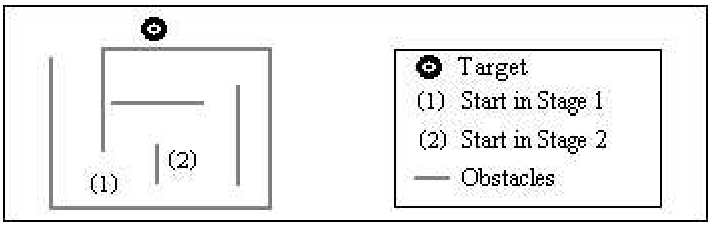


Figure 4

In the present implementation of ENA method, the following restriction has been set to the control change: once nn_1 delegates the task to nn_2, it does so until some collision arises or the simulation ends. If a collision takes place, nn_1 intervenes once again in the control decision now choosing to respond by itself or transferring the responsibility once again to nn_2.

This restriction responds to the need to stop the repetitive change of control that usually produces undesirable cyclic movements in the agents, and it is related to the particular implementation carried out on problems of obstacle evasion and target reaching.

5.2 Architecture and Codification

To shape the arrays, recurrent networks with free connection scheme, bias term, and transference function evolution were used, being each node capable of having one among four different sigmoids: $f_1(x)=1/(1+\exp(-0.5x))$, $f_2(x)=1/(1+\exp(-x))$, $f_3(x)=1/(1+\exp(-1.5x))$, $f_4(x)=1/(1+\exp(-2x))$ (figure 5). This type of networks has proved to render a better performance that the feedforward ones when they are applied to obstacle evasion and target reaching problems, allowing to obtain better individuals in fewer generations [2].

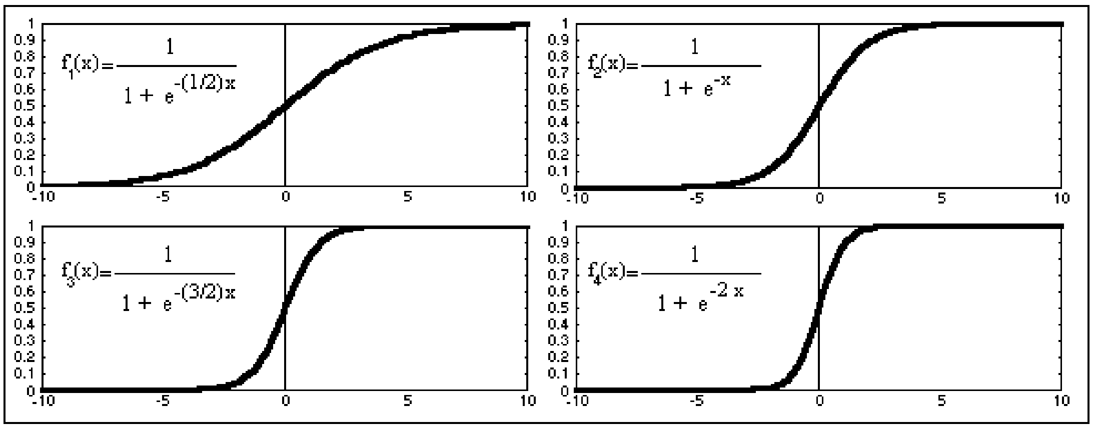


Figure 5

The variant SANE has been used, and was specially implemented to solve this problem. As an introduction to analyze the proposed extension, a short description of SANE method is presented next (consult [10] for a more details).

5.2.1. SANE

In conventional solutions that evolve Neural Networks, each individual of the population represents a complete neural network. This does not occur in SANE, where each individual is only a neuron of the network that must be combined to express the desirable solution. For this reason, SANE makes use of two populations, a population of neurons and a population of "blueprints" that represent the neural networks to be analyzed.

Within the population of neurons, each individual represents a hidden node of a feedforward ANN of only one hidden layer. The complete neural network is created by the combination of several neurons chosen from this population. In the chromosome, each neuron is codified as a set of connections of the form (*label*, *weight)* (Figure 6). The connection weight value is specified by the field *weight* (16 bits) interpreted as a real number. The neuron with which it is connected is determined by the field *label* (8 bits) as follows: if the label decimal value is D , n_E is the quantity of input neurons and n_S is the quantity of output neurons:

If $D > 127$ → the connection is towards the output neuron (D **mod** n_S)

If $D <=127$ → the connection is towards the input neuron (D **mod** n_E)

In the codification, the binary alphabet is used.

On the other hand, the knowledge of good neuron combinations is kept and exploited in the population of blueprints, from which the phenotypes are built. The chromosomes of this population are constituted by an array of L length (where L is the quantity of hidden neurons that the resulting network will have), of pointers to neuron chromosomes in the other population. Real numbers are used for the genetic codification (Figure 6).

The evolving algorithm works building neural networks from each blueprint chromosome. Each network's fitness, obtained by the performance of the neural network in which it participates, is assigned to the blueprint chromosome. The neuron population individuals' fitness is computed as the sum of the five better fitness obtained in the networks in which they have participated. The next generation in the population of neurons is obtained, and then the next generation in the population of blueprints.

SANE makes use of a highly elitist selection and replacement strategy in both populations. The best-ranked half of the population passes to the next generation. The best quarter is selected to be reproduced completing the remaining half with its descendants (Figure 7).

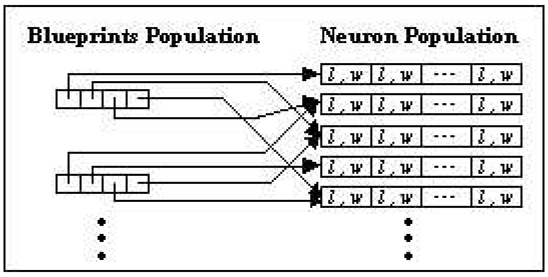


Figure 6

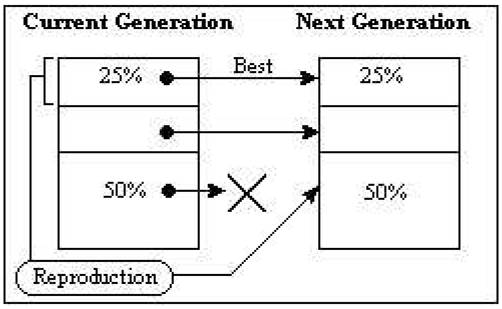


Figure 7

As regards reproduction, crossover of a point is used to get the first descendant, while the second is obtained by copying one of the parents. In the population of neurons, binary mutation with 0.001 probability per bit is applied. In the population of blueprints, two types of mutations are applied: i) change of pointer to other unit of the neuron population chosen at random with 0.01 probability, and ii) change of pointer to a pointed unit descendant with 0.5 probability.

5.2.2. Extension carried out on SANE:

By changing the genetic encoding in the population of neurons, SANE is extended to support networks with bias term and transference function evolution. The fields corresponding to the sigmoid type *TS* and the bias connection weight p_0 are set above the connection sequence that defines a neuron (Figure 8).

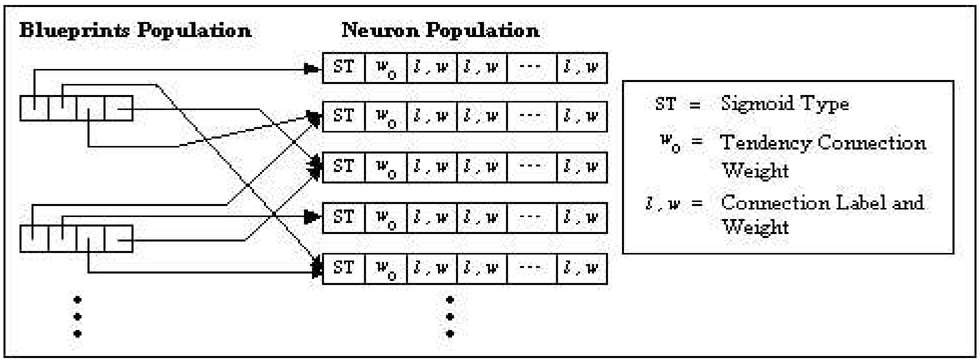

Blueprints Population
Neuron Population
ST
w_0
l, w
ST = Sigmoid Type
w_0 = Tendency Connection Weight
l, w = Connection Label and Weight

Figure 8

In order to support recurrent neural networks with free connection scheme, both hidden and output neurons are codified in the blueprints (Figure 9).

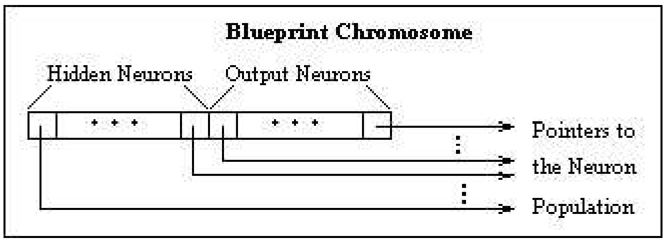


Figure 9

Also, it is necessary to modify the interpretation of the label of pair (l , w) at the moment of the phenotype construction. Numbering all the neurons from zero, placing first the input, then the hidden, and lastly the output ones, field l is interpreted as follows:

Let n_E, n_O and n_S be the number of input, hidden and output neurons respectively, and let D be the decimal value of field l:

If $D > 127$ → the connection is towards the numbered neuron (D **mod** $(n_O + n_S)) + n_E$

If $D <= 127$ → the connection is from the output neuron (D **mod** n_E)

Figure 10 shows the phenotype corresponding to a recurrent ANN with 5 input neurons, 2 hidden, and 2 output ones. For simplicity's sake, only two connections per neuron are considered, without bias term nor transference function evolution. Each blueprint is made up of four pointers to individuals to the population of neurons.

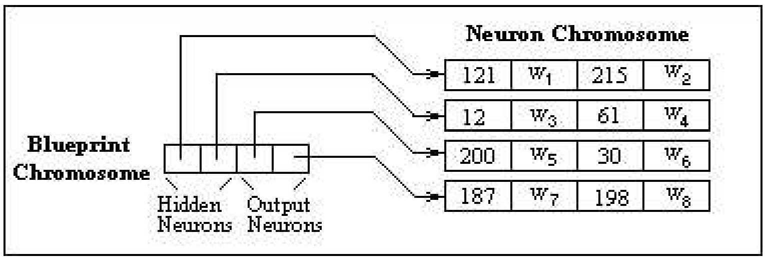


Figure 10

The phenotype decoding can be observed in Figure 11, which requires the following computations:

Computation	Connection
(121 mod 5) = 1	Connection from neuron 1
(215 mod 4) + 5 = 8	Connection towards neuron 8
(12 mod 5) = 2	Connection from neuron 2
(61 mod 5) = 1	Connection from neuron 1
(200 mod 4) + 5 = 5	Connection towards neuron 5
(30 mod 5) = 0	Connection from neuron 0
(187 mod 4) + 5 = 8	Connection towards neuron 8
(198 mod 4) + 5 = 7	Connection towards neuron 7

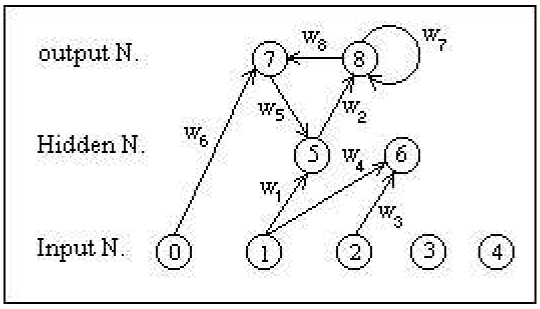


Figure 11

This example clearly shows the importance of the quantity of codified connections per neuron. In order to simplify the chart, only 2 have been chosen, but such lack of connections –although it yields simple networks (desirable characteristic)– increases the risk of leaving disconnected input neurons.

6 Obtained Results

ENA's performance was measured and compared to extended SANE and on various scenarios of different complexity (Figure 12). In all the cases, two and three stages were used.

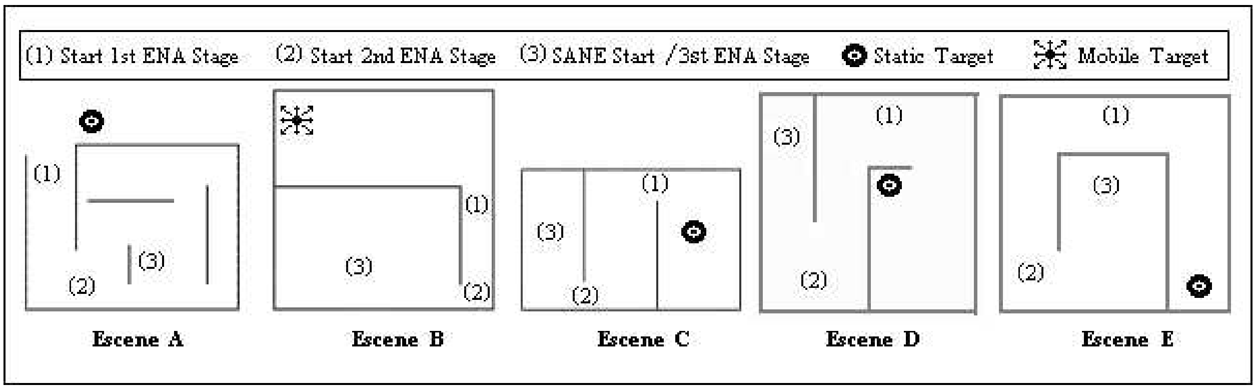
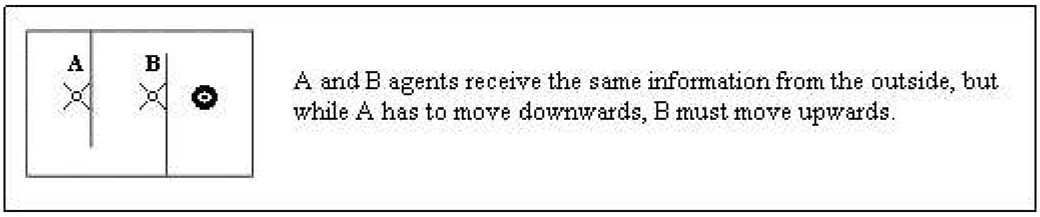


Figure 12

According to SANE's authors [9][10][11], their method has shown the advantages of cooperative coevolution in the search of solutions to control problems, outperforming the traditional strategies that codify, in a single chromosome, a complete neural network. For this reason, it has been chosen as comparative reference. On the other hand, [3] has proved that the extension of such method proposed in 5.2.1 has a far better performance.

Agents find scenarios C, D, and E especially difficult to dodge due to their limited "vision" of obstacles. Notice that there exists locations in which the sensed information from the outside is the same and, yet, the movement they have to make is completely different (figure 13).



A and B agents receive the same information from the outside, but while A has to move downwards, B must move upwards.

Figure 13

30 evolutions have been carried out on each scenario for each neuroevolutive method. In this way, it is possible to obtain an acceptable average of the best fitness obtained in each evolution generation for each method.

The quantity of successful evolutions, out of the 30 (in which the target is largely achieved), also represents an important point that characterizes the tested method performance. With this information, it is possible to estimate the likelihood of success when running an evolution using extended SANE or ENA.

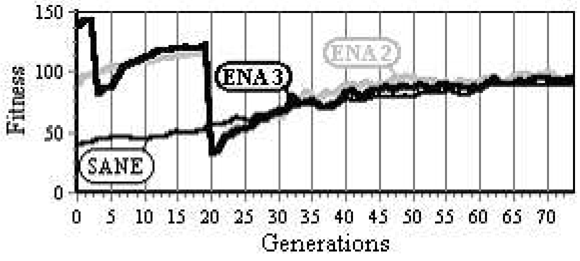
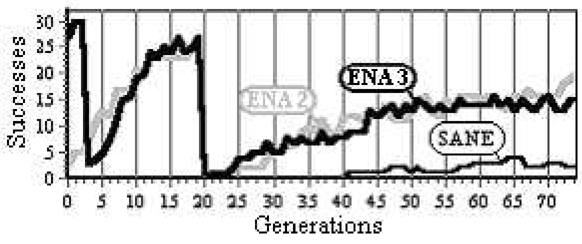
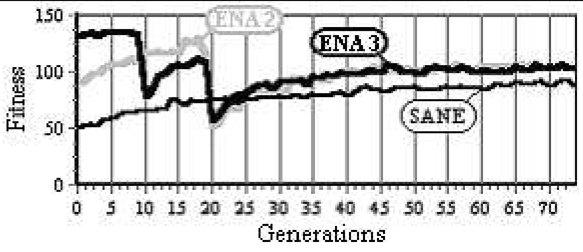
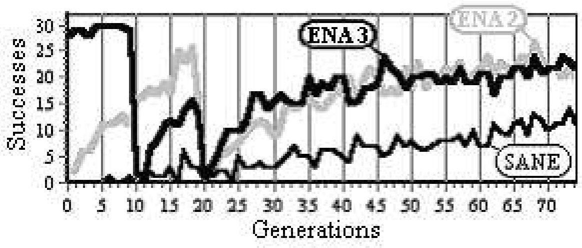
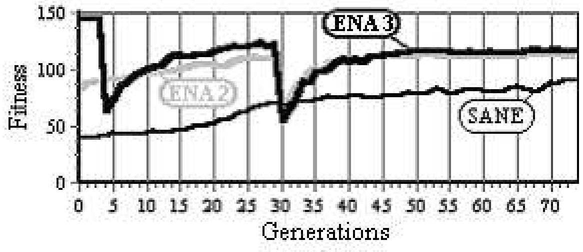
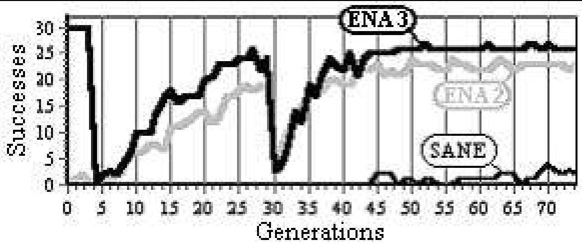
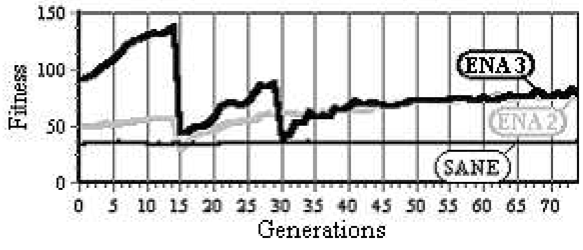
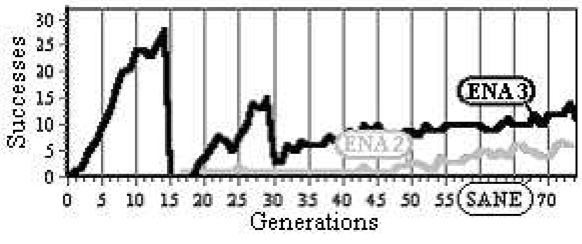
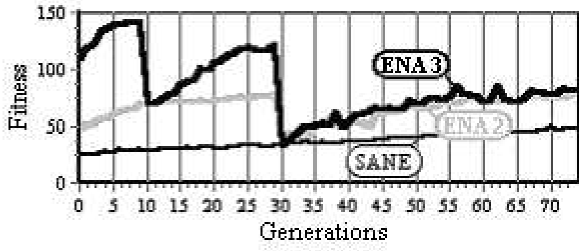
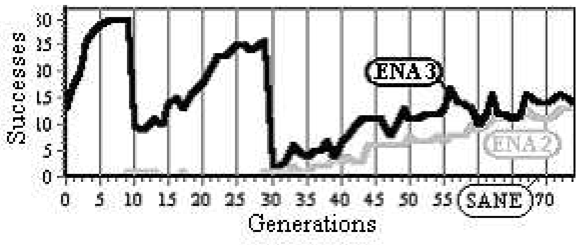
In all the tests carried out, recurrent networks with 14 input neurons, 2 output and 6 hidden ones, with bias connection and transference function evolution were evolved. Populations of 80 agents (80 blueprints) and 640 neurons were used. The type of sigmoid (2 bits), the bias connection weight (16 bits), and 18 connections per neuron were codified. Each connection was codified with 8 bits for the label and 16 bits for the weight. The evolutions were prolonged for 75 generations.

In simpler scenarios A and B, 20 generations were used for the first ENA evolving stages. In the rest of the scenarios, the last stage starts always in the generation number 30. These differences correspond to the different complexities that represent each scenario, though the use of other values is not discarded. When comparing with SANE method, the data captured from the start of the last ENA stage have to be taken into account, since in the first generations the data obtained are in function of simpler tasks rather than of the goal task.

Analyzing the results about the best fitness obtained in each generation (averaging the 30 evolutions), ENA has proved to outperform extended SANE in all the tested scenarios. Only in scenario A – of low complexity– extended SANE reaches ENA's performance in the last generations of the evolution. As the complexity of the problem to be solved increases (scenarios B, C, D, and E), the performance gap between them increases as well. This shows the importance of applying ENA –based on an incremental neuroevolutional strategy– to solve complex tasks (see Figures 14,16,18,20, and 22).

Also, the collected data about the quantity of successful evolutions (in which the target is effectively achieved) on the 30 carried out largely favors ENA method, showing a similar relation to that of the average fitness (see Figures 15, 17, 19, 21, and 23). This implies that there exists a higher possibility of reaching the target by means of a neural array than by extended SANE.

As regards the quantity of stages used in each case, notice that, as the scenario complexity grows, the use of three stages offers better results than two. This is clear in scenarios C, D, and E (Figures 18 to 23) where, even when the average fitness is similar, a better number of successes is obtained with three stages than with two. It is worth to mention that these last two scenarios could not be solved with extended SANE.

	Fitness Average	**Quantity of Successes obtained**
Scene A	 **Figure 14**	 **Figure 15**
Scene B	 **Figure 16**	 **Figure 17**
Scene C	 **Figure 18**	 **Figure 19**
Scene D	 **Figure 20**	 **Figure 21**
Scene E	 **Figure 22**	 **Figure 23**

The best fitness obtained from the 30 evolutions –for both of the methods – also shows ENA best performance and the advantages of using three evolving stages instead of two. Notice that the differences become more significant in more complex scenarios (Figure 24).

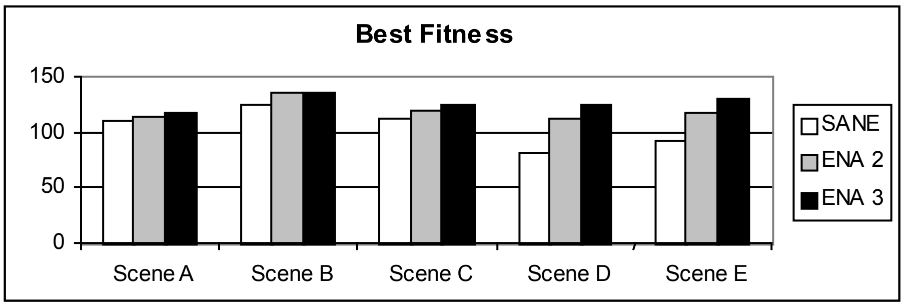


Figure 24

During the application of ENA to problems of target reaching and prey capture, it has been noticed in several opportunities that the method is capable of fixing the effects of a previous deficient stage. Figure 25 exemplifies ENA evolution with two evolving stages, where the first task is not properly solved (fitness < 100). This does not hinders the goal task from being solved successfully in the second evolving stage that begins at the 30th generation.

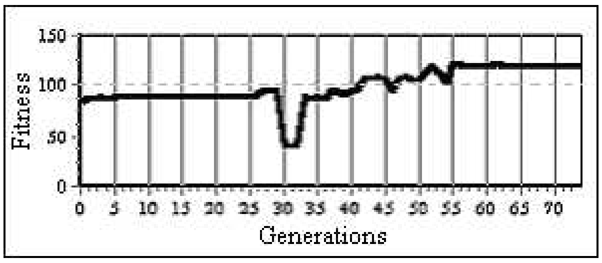


Figure 25

Within this line, work has been done on the most complex scenarios D and E, applying ENA with two and three evolving stages. Figures 26 and 27 sum up the results obtained.

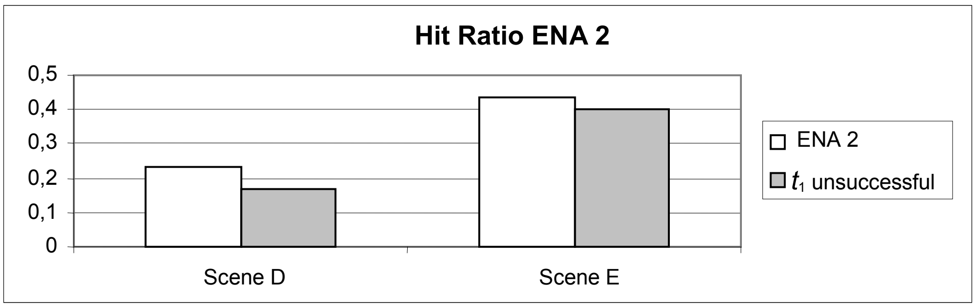


Figure 26

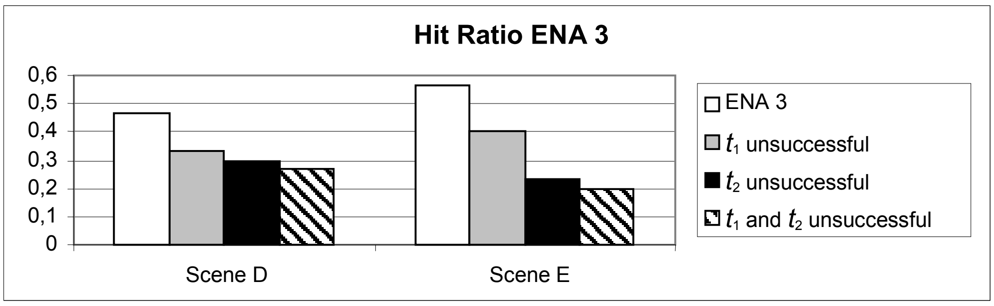


Figure 27

The white columns of both figures represent the relative frequency of success in order to solve the goal task adequately, computed in 30 evolutions, without making any consideration on previous stages which might have solved their corresponding tasks or not. Thus, these values estimate the likelihood of solving the posed problem by means of ENA method.

The remaining columns represent the relative frequency of success in order to solve the goal task, computed on the 30th evolution in which some or both (depending on the case) previous stages fail. In this way, these values represent an estimation of the likelihood of solving the posed problem bearing in mind that some or both previous stages have not been good at solving their corresponding tasks.

As it can be seen, although the degradation that takes place due to deficient initial stages, the recovery capability of ENA method to reach the final objective – solving the posed problem– is quite significant.

7 Conclusions and future lines of work

A new method for learning complex action sequences has been presented, based on evolving neural arrays (ENA) whose elements are incrementally evolved.

The results obtained applying ENA in the resolution of problems of target reaching and obstacle evasion have outperformed those obtained by employing extended SANE. The difference in the performance is enhanced as the complexity of the problem to be solved increases. This fact clearly indicates the use of the new method in complex problems resolution.

ENA method has also proved to be capable of fixing adverse effects of a deficient previous stage. This significantly increases the likelihood of success of the evolutions.

One of the inconveniences of incremental evolution is the need to establish *a priori* the sequence of intermediate tasks to be solved. This turns it into a technique dependent on the problem and difficult to generalize. For this reason, automatic segmentation of ENA method's evolving stages is posed as future line of research.

Nowadays, research is being done on the scopes of using non-incremental Evolving Neural Arrays. In this line, advances have been made in the resolution of control problems that involve complex action sequences. If no division in stages is done, networks integrating an array evolve simultaneously as they learn to distribute their activities in function of the process time.

For this reason, we can assert that evolving neural arrays have proved to be interesting structures to solve control problems, both in incremental and non-incremental evolving strategies. However, the discussion of the latter is beyond the scope of this paper.

The documentation of this paper – together with the ad-hoc developed environment to carry out the simulation and the measures here mentioned – is available at LIDI (Laboratory of Research and Development in Computer Science), and can be consulted at http://lidi.info.unlp.edu.ar.

References

[1] Bruce, J. and Miikkulainnen, R. Evolving Populations Of Expert Neural Networks. Department of Computer Sciences, The University of Texas at Austin. *Proceedings of the Genetic and Evolutionary Computation Conference*. (GECCO-2001, San Francisco, CA), (2001), pp. 251--257.

[2] Corbalán, L., Pisano, M., Osella Massa, G. y Lanzarini L. Criaturas Virtuales especificadas a través de Redes Neuronales Evolutivas. *VII Congreso Argentino de Ciencias de la Computación*. Argentina, Vol. 2, (October 2001), pp. 1105-1115.

[3] Corbalán Leonardo. Evolución de redes neuronales para comandar criaturas que alcanzan objetivos sorteando obstáculos en un entorno virtual 2D. Graduate Thesis corresponding to the Licentiature in Computer Science. UNLP. March 2002.

[4] Freeman, J. A. & Skapura, D. M. *Redes neuronales Algoritmos, aplicaciones y técnicas de programación*. Addison–Wesley, 1991. Spanish version of: Rafael García -Bermejo Giner. Addison–Wesley Iberoamericana 1993.

[5] Goldberg, D. E. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison–

Wesley, 1989. p. 10

[6] Gomez, F. and Miikkulainen, R. Incremental Evolution Of Complex General Behavior. Department of Computer Sciences, The University of Texas at Austin. *Adaptive Behavior*. Vol 5, (1997), pp.317-342.

[7] Gomez, F. and Miikkulainen, R. Solving non-markovian Control. Tasks with Neuroevolution. Department of Computer Sciences, The University of Texas at Austin. *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-99, Stockholm, Sweden)*, San Francisco, CA: Kaufmann. (1999), pp. 1356-1361.

[8] Han Yong, Ch. and Miikkulainen, R. Cooperative Coevolution of Multi-Agent Systems *Department of Computer Sciences, The University of Texas at Austin. Technical Report AI-01-287* (February 2001).

[9] Moriarty, D. E. & Miikkulainen, R. Efficient Reinforcement Learning through Symbiotic Evolution. Department of Computer Sciences, The University of Texas at Austin. Austin, TX 78712. *Machine Learning*, Vol. 22, (1996), pp.11-33.

[10] Moriarty, D. E. Symbiotic Evolution of Neural Networks in Sequential Decision Tasks. *Department of Computer Sciences, The University of Texas at Austin Ph.D. Dissertation. Technical Report AI97*-257, January 1997.

[11] Moriarty, D. E. & Miikkulainen, R. Forming Neural Networks Through Efficient and Adaptive Coevolution. Information Sciences Institute, University of Southern California. Department of Computer Sciences, The University of Texas at Austin. *IEEE Transactions on Evolutionary Computation*. Vol.5, (1997), pp.373-399.

[12] Pérez Bergquist, A. S. Applying ESP and Region Specialists to Neuro-Evolution for Go. *Department of Computer Sciences, The University of Texas at Austin. Honors Thesis, Technical Report CSTR01-24* (May 2001).

[13] Tapia, R. *Las Células de la Mente*. Impresora y Encuadernadora Progreso, S. A. de C. V. (IEPSA), Calz. de San Lorenzo, 244; 09830 Mexico, D.F. December 1996.

[14] Yao, X. and Liu, Y. Ensemble Structure of Evolutionary Artificial Neural networks. *Computational intelligence Group, School of Computer Science University College*. Australian Defense Force Academy, Canberra, ACT, Australia 2600. 1996.

[15] Yao, X. and Liu, Y. A New Evolutionary System for Evolving Artificial Neural Networks. *IEEE Transactions on Neural Networks*, Vol 8, n°. 3, pp 694-713, 1997

[16] Yao, X. Evolving Artificial Neural networks. School of Computer Science. The University of Birmingham Edgbaston, Birmingham B15 2TT. *Proceedings of the IEEE*. Vol.87, No.9, (September 1999), pp.1423-1447.