# ALENA

# Adaptive-Length Evolving Neural Arrays

**Leonardo C. Corbalan, Laura Lanzarini and Armando E. De Giusti**
**Instituto de Investigación en Informática LIDI, Facultad de Informática,**
**Universidad Nacional de La Plata, Buenos Aires, Argentina**
corbalan@lidi.info.unlp.edu.ar

**ABSTRACT**

Evolving neural arrays (ENA) have proved to be capable of learning complex behaviors, i.e., problems whose solution requires strategy learning. For this reason, they present many applications in various areas such as robotics and process control. Unlike conventional methods –based on a single neural network– ENAs are made up of a set of networks organized as an array. Each of them represents a part of the expected solution.

This work describes a new method, ALENA, that enhances the solutions obtained by solving the main deficiencies of ENA since it eases the obtaining of specialized components, does not require the explicit decomposition of the problem into subtasks, and is capable of automatically adjusting the arrays length for each particular use.

The measurements of the proposed method –applied to problems of obstacle evasion and objects collection– show the superiority of ALENA in relation to the traditional methods that deal with populations of neural networks. SANE has been used in particular as a comparative referent due to its high performance. Eventually, conclusions and some future lines of work are presented.

**Keywords**: Evolving Neural Networks, Evolving Neural Arrays, Learning, Genetic Algorithms, Subpopulations.

## 1. INTRODUCTION

Evolving Artificial Neural Networks (EANNs) are a particular case of the artificial neural networks (ANNs) in which the adaptation is carried out by way of training and, above all, evolution [13]. The evolution has been used in various ways: in order to obtain connection weights, architecture design, initial parameters value, learning rules, etc. [14].

Freeman and Skapura [6] discussed the need to learn to combine small ANNs, putting them under the control of other networks in order to solve the scaling problem. Xin Yao and Yong Liu [12] have studied the benefits of using the complete population of the neural networks obtained in the last generation –result of an evolving process–, instead of that of better fitness. More recently, Bruce and Miikkulainen [1], working on the problem of free-hand characters recognition, have demonstrated that all the neural networks of a population, together with an effective specialization technique, can respond better collectively than individually.

Under the general lines of exploring solutions based on multiple neural networks, [4] presents evolving neural arrays (ENA). This method has proved to be more efficient than other neuroevolving strategies as it combines the benefits of incremental evolution with the power of several neural networks integrated in a single controller. However, the need of defining explicitly the subobjectives in each substage –compulsory inheritance of incremental evolution [8]– constitutes the weakest point of the method hindering its generalization.

An alternative strategy to evolve neural arrays was presented in [5] to avoid the explicit definition of the subobjectives. In this way it is possible to build evolving algorithms easily adaptable to other types of problems. Nevertheless, the obtained solutions often lack tolerance for disturbances that may alter the normal process times.

This work proposes a new method of evolving neural arrays, ALENA, rectifying the mentioned drawbacks. ALENA does no work on the basis of subobjectives explicit definition; the networks activation strategy within the array does not depend on the processing time but exclusively on the data input into the controller, and the algorithm is capable of adjusting the length of the array to the most adequate size for each situation. In this way, there exists an upgrading of the solution efficiency and the algorithm robustness easily adaptable to other types of problems.

## 2. ALENA

ALENA allows to obtain controllers made up of neural arrays that solve process control problems more efficiently than traditional solutions. Member networks arising from the evolution of different subpopulations learn to become specialized in different subtasks of the total process to be controlled. Thus, from the coordinated operation of these networks, the solution of a complex problem arises more efficiently.

### Internal Organization of the Controller

The controller consists of a neural array, a tuple of neural networks in the manner of $C=(nn_1,nn_2,..,nn_n)$. Like a network, a neural network accepts an input of data, it is evaluated, and produces the corresponding output. At each instant *t*, the output will be provided by some of the member networks. Only one of them remains active at a time, thus the controller processing time is evaluated in such instant.

### Performance of the Controller

At the beginning of the process, the only active network of the controller is $nn_1$, which solves all the inputs up to its auto-deactivation. Once deactivated, the control moves to $nn_2$, still evaluated until it "decides" to be deactivated, passing the control to $nn_3$. This control delegation goes on until $nn_n$ (last component) is eventually activated, remaining in such state until the end of the process. In all the cases, the network deactivation is carried out after the output of the controller occurs, leaving the next neural network active for the next instant of evaluation.

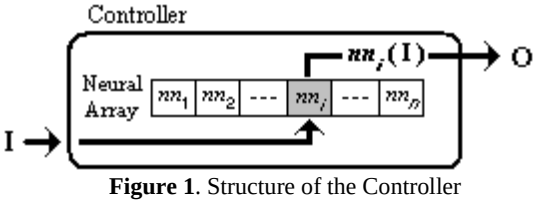


**Figure 1**. Structure of the Controller

Array networks have an output neuron additional to the quantity defined by the problem to be solved. The value of such neuron is not part of the controller output and is used to determine whether the evaluated network should be deactivated.

Let $d_i(t)$ be the value of the extra output neuron of the active network $nn_i$ at the evaluation instant $t$. The neural network $nn_i$ will be still active for instant $t$+1 if some of the following two conditions are fulfilled:

1) $Abs(d_i(t)-d_i(t\text{-}1)) < c$, where $c$ is an *a priori* defined constant.

2) $i = n$, being $n$ the array length.

On the contrary, if none of these conditions is fulfilled, $nn_{i+1}$ will be activated at instant $t$+1.

In this way, the activation/deactivation mechanism of the member networks is defined in function of the information provided to the array at two consecutive time instants. The sensitivity of this mechanism can be adjusted by the proper choice of the constant $c$ that appears in the first condition. For a range between 0 and 1 of the extra neuron output, good results with values for $c$ comprising 0.01 and 0.1 have been found. As with other parameters in the neural networks theory, previous experimentation may be the best option to determine this value properly.

## 3. EVOLVING ALGORITHM

Two phases are distinguished in the evolving algorithm:

1) *Exploration:* It comprises the first generations of the evolution. Beginning with single-component arrays, the algorithm approaches the solution to the problem at the same time the array length becomes larger with the incorporation of new neural networks.

2) *Exploitation*: Once the solution to the problem is close enough (high enough fitness), the array length remains fixed and the solution is achieved by means of the adjustment or optimization of the components.

If the array length established during the exploration is $n$, controllers like $C=(nn_1,nn_2,..,nn_n)$ will be obtained by concurrent evolution of $n$ subpopulations of neural networks, one for each $nn_i$ component of the controller.

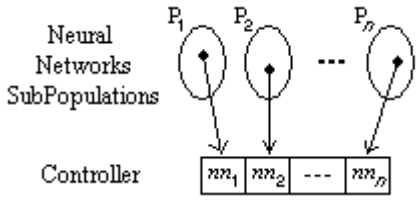


**Figure 2**. Neural Networks Subpopulations

In both phases –exploration and exploitation–, populations $P_i$ of neural networks are evolved one at a time, during variable length periods, measured in quantity of generations, and determined in function of the best fitness obtained per generation. As the process advances, the populations evolutions will succeed orderly in a circular line. Each $P_i$ will eventually undergo several evolution periods.

The algorithm pseudocode used is the following:

```
Begin Program
// exploration
n:=0
while not endExploration
  n:= n + 1
  Generate at random subpopulation Pn
  Repeat
    stepOfEvolution(Pn)
  Until fitness_Stationary
End while

// now n keeps the array length
// exploitation
i:=0
While not termination
  i:= i mod n + 1
  Repeat
    stepOfEvolution(Pi)
  Until fitness_ Stationary
End while

End Program.

//routine StepOfEvolution
StepOfEvolution(Pi)
begin
Build a controllers population Pc(i)
Evaluate each controller of Pc(i)
Assign fitness to networks of Pi
If not fitness_stationary then
  Pi:= next_generation(Pi)
End
```

The condition `endExploration` is related to the understanding, certainly vague, of the already mentioned "high enough fitness." This condition must be defined in function of the particular implementation of a given problem. As example, in the problem of obstacle evasion, objects search and collection presented in this work, `endExploration` takes the value true when a fitness equal to the 75% of the maximal value attainable by optimal controller is achieved.

The condition `fitness_Stationary` will be true when the best fitness cannot be upgraded over a given number of generations. This quantity is a parameter of the algorithm. This paper shows the results that suggest the use of a small number for this parameter (1 or 2, at most).

The condition `termination` becomes true when a given generation, or a value of previously fixed fitness, is achieved.

Controllers are built up with each of the networks of the population that undergoes the evolution and the best-ranked network according to its fitness, in each of the remaining populations. In this way, the evaluated controllers differ from each other, only in the i-th component.

$P_c(i) = \{(nn_1,nn_2,...nn_i,...nn_n)$: $nn_i \in P_i$ and
$nn_J = best(P_J)$ for $i \neq J\}$

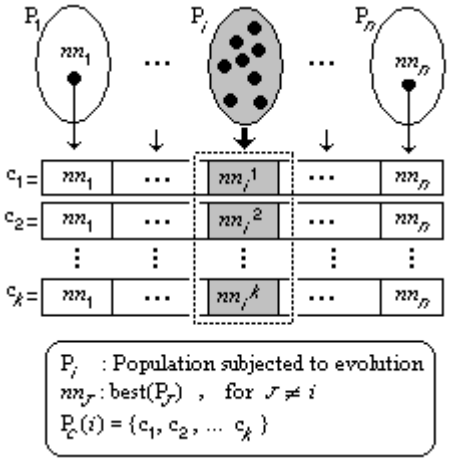


**Figure 3**. Subpopulations Evolution

In this way, at any moment, there exists a population $P_i$ that evolves to optimize the integration of the i-th component of the controller with its remaining networks.

The fitness attained by each controller during the evaluation is assigned to its i-th component (individuals of population $P_i$).

In order to obtain the next generation in the networks population $P_i$ no specific evolving algorithm is assumed. Simple genetic algorithms –such as the presented by Goldberg [7]– as well as other more sophisticated ones –typical of the neuroevolution paradigm– can be used. In particular, in this work, SANE [9] [10] [11] has been implemented, which is briefly described since it has also been used as comparative referent in the tests carried out.

No restriction is assumed on the network parameters undergoing evolution (connection weights, architecture, transference function, etc.). A set of variants can be found in [14], where several researches on neuroevolution are quoted, including hybridization with traditional learning algorithms.

Since a neural array evaluation makes use of the same computing resources as a single network, and since subpopulations are evolved in shifts, one at a time, the computational load of the algorithm here presented is similar to that of any conventional method. In addition, since the controller shares most of the components, a good implementation of its evaluation allows to upgrade the processing times, enabling, in some cases, the evaluation of a controller and the use the output for another, thus saving computing time. It should be bear in mind that, in neuroevolving algorithms, neural networks evaluation often consumes most of the running time.

A small variant to the algorithm here presented allows to reduce the storage requirements. Instead of keeping the whole subpopulations, only a given percentage of the best-ranked networks in them is to be kept. Once the shift for its evolution is obtained, the subpopulation is completed at random with chromosomes that provide genetic diversity.

**More on the exploration and exploitation phases**

During the algorithm exploration phase, it is important that the partially built controllers tend to deactivate all of their neural networks during their evaluation. Thus, the future next component will have chances of executing itself. For it, those controllers that have not deactivated all of their networks at the last instant of the evaluation (only during the exploration) are penalized with low fitness, and the evaluation time is extended every time that the algorithm adds a component to the neural arrays.

In the exploitation phase, the components optimization can result in the decrease of the array length. It is common to find that, while the algorithm adjusts the i-th component, subjecting the subpopulation $P_i$ to evolution, some controller solves the problem using the first $k$ neural networks with $i \leq k < n$, being $n$ the current length of the arrays. In this case, subpopulations $P_J$ are eliminated for $k < J \leq n$, and the arrays are cut establishing their new length in $k$.

## 4. SANE

SANE has demonstrated the advantages of cooperative coevolution in the search of solutions to control problems, being superior to the traditional strategies.

In the conventional solutions that evolve neural networks, each population individual represents a complete neural network. This does not occur in SANE, where the individuals of a neuron population are combined to make up the networks that express the searched solution. This combination undergoes evolution in a blueprints population.

Each member of the neuron population codifies, with binary alphabet, a sequence of connections (*label*, *weight*) that thoroughly defines a hidden node of a feedforward neural network with a single hidden layer. The field *label* identifies the input or output neuron with which a connection is established. Each member of the blueprints population consists of a series of pointers to the other population that identify the neurons building the neural network (fig. 4). The blueprints population uses the real alphabet.

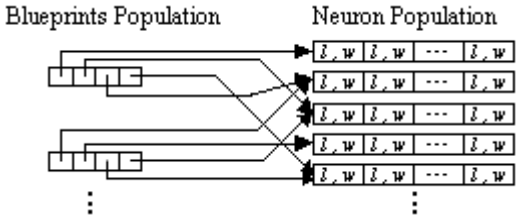


**Figure 4**. SANE Populations

The evolving algorithm operates building the neural networks from each blueprint chromosome. Each network fitness, obtained by its performance in the solution of the posed problem, is assigned to the blueprint chromosome. The fitness of the neuron population individuals is calculated as the sum of the five better fitness obtained in the networks of which they have been part. The next generation is obtained in the neuron population and, then, the next generation in the blueprints population.

SANE uses an elitist selection and replacement strategy in both populations. The best ranked half of the population moves towards the following generation. The best quarter is selected to reproduce itself completing, with its descendants, the remaining half (fig. 5).

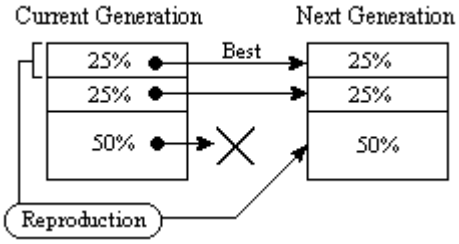


**Figure 5**. Replacement Selection in SANE

In the reproduction, one point crossover is used in order to obtain the first descendant whereas the second one is obtained by a copy of one of the parents. In the neurons population, binary mutation is applied with 0.001 probability per bit. In the blueprints population two types of mutations are applied: i) change of pointer to another unit of the neurons population chosen at random with 0.01 probability, and ii) change of pointer to a descendant of the unit pointed with 0.5 probability.

[9][10][11] can be consulted for a more detailed explanation of this method.

## 5. OBSTACLE EVASION, OBJECT SEARCH AND COLLECTION

**Problem Definition**

The aim consists in achieving intelligent behavior in an agent that moves freely in two dimensions within the boundaries of a virtual environment, interacting with obstacles, which the agent must learn to elude, and objects that it must learn to find and collect.

A controller directs the agent movements in a temporal interval simulated by the succession of $n$ discrete time instants (simulation steps). At each instant, the controller

is stimulated by a set of input signals. The output is made up of an ordered pair $(\alpha, \rho)$ that determines the rotating and displacement angle performed by the agent on the surface. However, the obstacles and the boundaries of the environment can frustrate the movement.
The goal of the problem is to control the agent so that it, from a given place, finds and withdraws from the scenario two objects of different kind, in a given order, in the lesser possible quantity of steps.

**Agent**
Each agent has 5 sensors to detect obstacles, at a short distance (twice its own diameter), distributed uniformly in front of the agent in order to achieve an vision angle of 144° (fig. 6).

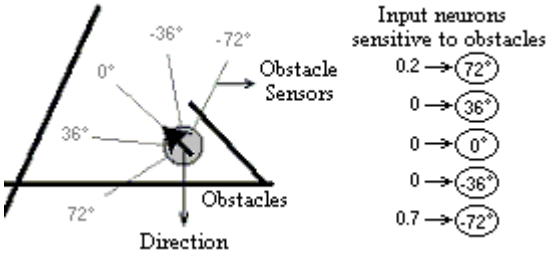


**Figure 6**. Obstacle Sensors

The obstacle sensors can be considered as touch-sensitive extensions oriented forwards in five directions. Each sensor provides a real value belonging to the interval [0,1] directly proportional to the closeness of the obstacle detected.
It also has 4 sensors, affected to the detection of the object 1, and others 4 for the detection of object 2. They show the absence or presence of the object in a radio of 90° around the agent, by means of the values 0 or 1. The objects detection is carried out at any distance, provided that no obstacle is in the way, impeding its visualization (fig. 7).
The four sensors affected to the detection of an obstacle are set to 1, at the moment in which the corresponding object is collected.

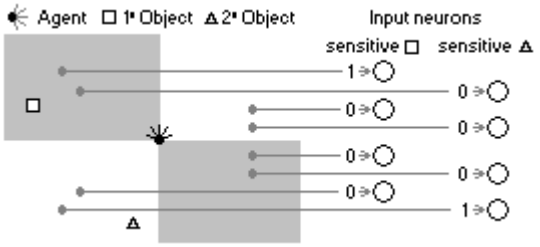


**Figure 7**. Object Sensors

**Defined Scenarios**
The complexity of the problem varies in function of the three defined scenarios (fig. 8).
Scenario A is the simplest one. Its reduced dimensions and the obstacles arrangement limit the agent mobility. An evolving algorithm will easily find a good solution. It is enough to evolve a controller that advances until an obstacle is found (labyrinth wall) and goes around the scenario without leaving it, keeping it always to the right (known heuristics to dodge given labyrinths).

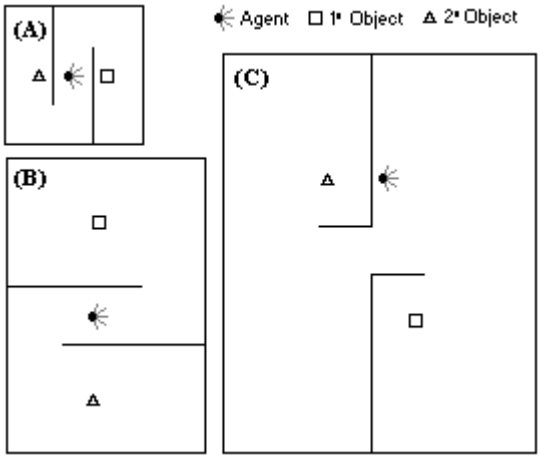


**Figure 8**. Scenarios

Scenario B represents an intermediate level of difficulty. The controller must be more elaborated than in the previous scenario. Here, it is necessary that the agent becomes separated from the walls in order to reach the objects. Once the first object is collected, the challenge consists in finding the most adequate direction to go on. It should be considered that, from that place, the second object is not "in sight", nor any of the obstacles. The agent only handles the information that the first object has been collected (four sensor set to 1) and, from this data, it must adopt the most adequate moving strategy to solve the scenario efficiently (lesser quantity of possible steps).
Scenario C is the most difficult of the three. To the difficulties of scenario B it must be added the need of carrying out several changes of direction to find an efficient run. This behavior is not easy to acquire, mainly if, as in this case, there are no many significant changes in the data input to the controller.

**Fitness Allocation**
The problem of obstacle evasion can be considered of the type of Sequential decision tasks. Its main characteristic is the difficulty to designate with precision the goodness of a decision taken, being necessary a decisions sequence before being able to measure that the effects of any of them have been. Some of the examples of the real world are: data routing in Internet routers, flow control in chemical reactors, air-traffic control, etc. In all of these cases, the effect of a simple decision is evidenced after some time has passed and, even then, it is frequently difficult to establish which, and to which extent, the responsible decisions were for what has occurred [10].
The obstacle elusion and objects search have been framed within this type of problem and, thus, there is no assessment of the controllers aptitude until the simulation used for its evaluation has not finished.
Once the simulation time finishes, the controller fitness is computed, which in turn directs the agent, in the following manner: Let *f(a)* be the aptitude value assigned to agent *a* for its performance in the simulation.

- If object 1 is not reached, *f(a)* will take a value of the interval [0,40) computed proportionally to the path run towards such object.
- If, otherwise, object 1 was collected but object 2 remains uncollected, *f(a)* will take a value of the interval [40,50) computed proportionally to the path run towards object 2.

- Eventually, if both objects are collected, *f(a)* will belong to the interval [50,60). If *s* is the number of steps used to complete the task, *f(a)*=50+10(*o*/*s*), being *o* an estimation of the quantity of steps necessary to solve the scenario for a optimum controller.

**Artificial Neural Networks Architecture**

In order to build arrays, feedforward networks are used with a single hidden layer made up of 8 neurons, with a free connection scheme (not thoroughly connected), with bias and transference function evolution letting each node have one out of four different sigmoids: $f_1(x)=1/(1+\exp(-0.5x))$, $f_2(x)=1/(1+exp(-x))$, $f_3(x)=1/(1+\exp(-1.5x))$, $f_4(x)=1/(1+exp(-2x))$. The transference function evolution has proved to have a good performance when applied to problems of obstacle evasion and reaching objectives [2][3], reason why it has been used in this work.

## 6. EXPERIMENTATION

The performance of this new evolving strategy was measured with and compared to SANE.

In all the tests carried out, feedforward networks were evolved with 13 input neurons, 2 output (3 in the case of the neural arrays), and 8 hidden neurons, tendency connection, and transference function evolution. In the neuron population, the sigmoid type (2 bits), the tendency connection (16 bits), and 15 connections per neuron were codified. Each connection was codified with 8 bits for the label and 16 bits for the weight. Blueprints subpopulations chromosomes were codified with real numbers. Populations of 80 blueprints and 640 neurons were used, both in SANE and ALENA subpopulations.

For ALENA method, the condition of achieving a fitness equal to 75% of the maximum value attainable by an optimum controller (estimated over every scenario) was taken as the finalization of the exploration phase.

On the whole, 15 tests were carried, testing SANE, $ALENA_1$, $ALENA_2$, $ALENA_5$, and $ALENA_{10}$ on each of the described scenarios (fig. 8). The sub-index accompanying ALENA method refers to the parameter that determines the condition `fitness_Stationary` of the algorithm. Thus $ALENA_i$ is an instance of the ALENA method in which the condition `fitness_Stationary` becomes true when the fitness curve is not upgraded during *i* generations.

Each of the tests consisted in 30 evolutions extended through 300 generations. In order to analyze the tested methods performance, the functions *fitnessAvrg* and *HitRatio* are used and defined as follows:

$$FitnessAvrg(g) = \frac{1}{30}\sum_{i=1}^{30} F_i(g) \quad , \quad 1 \leq g \leq 300$$

Being $F_i(g)$ the fitness value of the best individual of the generation *g*, in the evolution number *i*.

$$HitRatio(g) = \frac{100}{30}\sum_{i=1}^{30} H_i(g) \quad , \quad 1 \leq g \leq 300$$

Being $H_i(g)$ a function that returns 1 or 0, according to the success or failure of the evolution number *i* in finding a controller that solves the task (collecting both objects) in at most *g* generations.

**Results obtained**

An important parameter of the new proposed method is that which indicates how many consecutive generations without progresses make the `fitness_Stationary` condition come true. The impact of this parameter is especially significant during the algorithm exploration phase.

In principle, one may think that a large value will lead to invest too many generations in the optimization of a component, whereas it would be of better profit to add a new network to the array. On the other hand, a small value would not give the chance to the algorithm to find a nearly-good component when a new network is being added (in the exploration phase) to the array, which would undergo the same fate. The experimentation carried out does not support this last assertion. Figure 9 and 10, for scenario A, show that the lesser the value of this parameter, the better the method performance. The tests on scenarios B and C yielded very similar results.

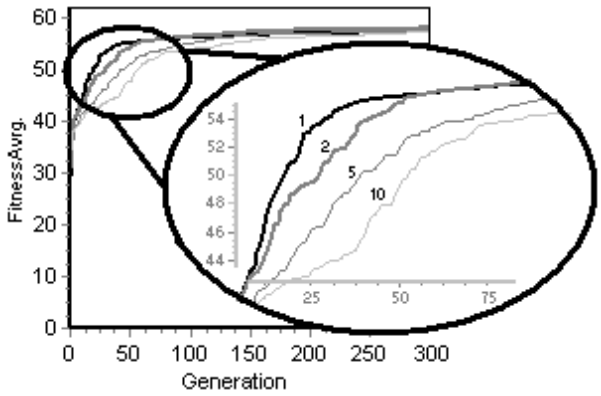


**Figure 9.** Average Fitness. $ALENA_1$, $ALENA_2$, $ALENA_5$ and $ALENA_{10}$ on scenario A

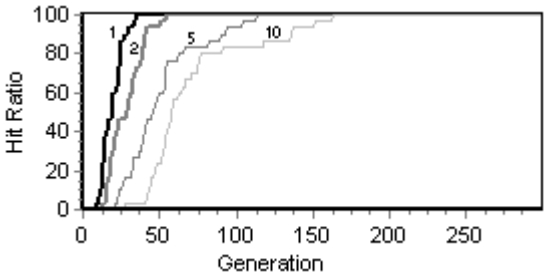


**Figure 10.** Hit Ratio. $ALENA_1$, $ALENA_2$, $ALENA_5$ and $ALENA_{10}$ on scenario A

$ALENA_{10}$ needed 157 generations in order to obtain the 100% of efficiency (collection both objects in the 30 test evolutions), while $ALENA_1$ only needed 51 for the same result.

The use of a small value for this parameter results in controllers with greater number of components. Table 1 shows the average length of the neural controller that have solved each scenario.

| | $ALENA_1$ | $ALENA_2$ | $ALENA_5$ | $ALENA_0$ |
|---|---|---|---|---|
| Scen. A | 8.93 | 8.23 | 6.30 | 6.03 |
| Scen. B | 7.20 | 5.83 | 5.07 | 4.83 |
| Scen. C | 8.10 | 6.57 | 5.57 | 4.87 |

**Table 1**. Average Length of the neural array

It must be recalled that most of the neural networks in a controller does not assume a greater computational load,

since within the array only one network remains active at each instant. The genetic algorithm does not suffer overload either, since only one populations is evolved at a time.

Next, the comparative graphics between $ALENA_1$ and SANE on the different scenarios are presented.

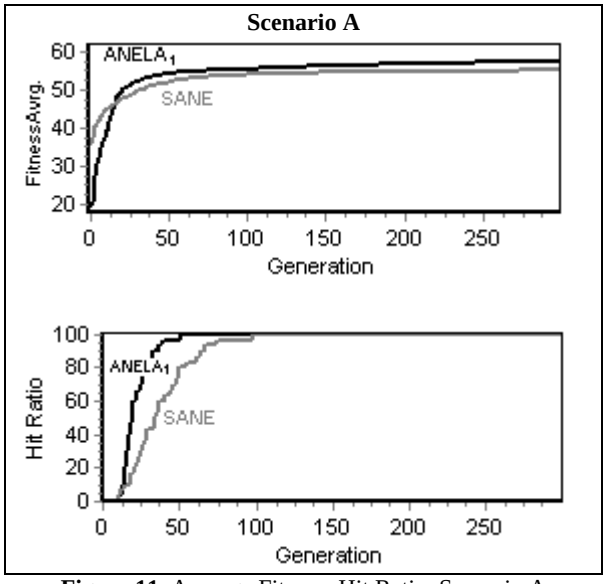


**Figure 11**. Average Fitness. Hit Ratio. Scenario A

Scenario A is successfully solved by both methods. However, in order to achieve a 100% efficiency, SANE used 100 generations, whereas $ALENA_1$ only 51. In addition, the best fitness always favored $ALENA_1$ which found the speediest controllers (fig. 11).

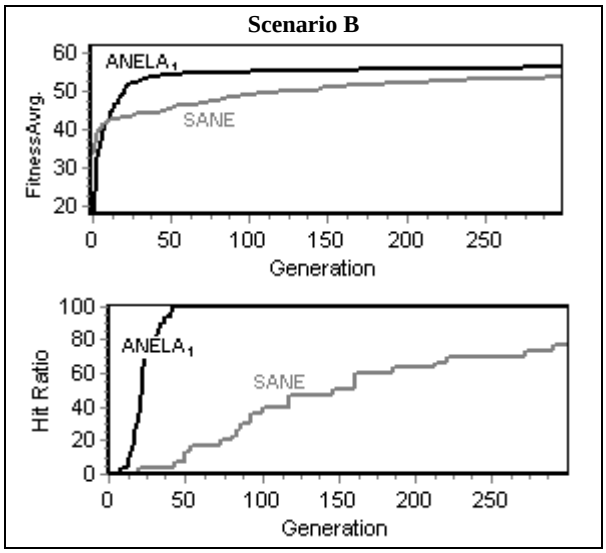


**Figure 12**. Average Fitness. Hit Ratio. Scenario B

Scenario B, more complex than A, emphasizes the performance differences of both methods. In only 42 generations, $ALENA_1$ reached the 100% of efficiency, whereas SANE only obtained the 77% in 300 generations (fig. 12).

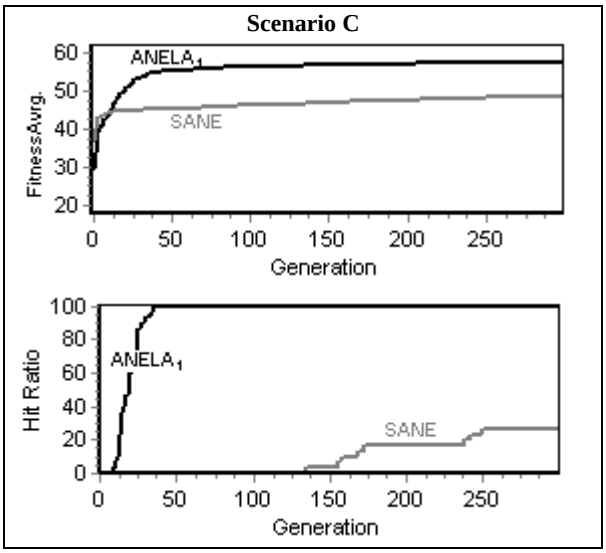


**Figure 13**. Average Fitness. Hit Ratio. Scenario C

Finally, the most difficult of the three Scenarios evidences that $ALENA_1$ obtains a significantly higher performance than SANE in the solution of this problem.
In the tests carried out on the three Scenarios, notice that SANE performance is affected notoriously by the Scenario on which the test is carried out. On the contrary, $ALENA_1$ behaves rather similarly in all of them. This is clearly depicted in figures 14 and 15.

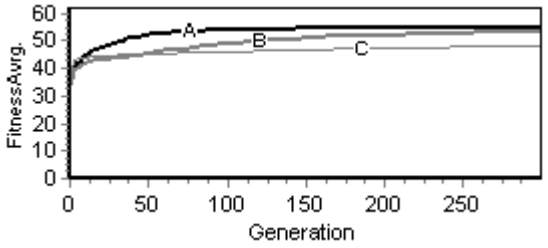


**Figure 14**. SANE on Scenarios A B and C

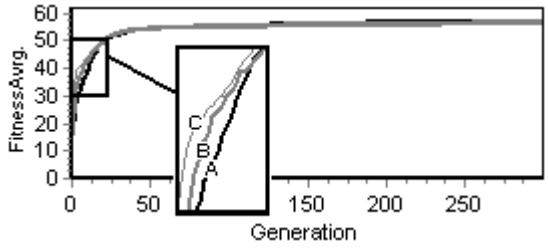


**Figure 15**. $ALENA_1$ on Scenarios A B and C

## 7. CONCLUSIONS AND FUTURE LINES OF WORK

A method evolving neural arrays to control process more efficiently than the conventional methods but with similar processing requirements has been presented.
ALENA also upgrades other neural arrays evolution strategies eliminating the explicit definition of subobjectives and adjusting automatically the array length for each situation. In this way the solutions are easier to be generalized with other type of problems.
The results obtained to the present are expected to be applied in the area of robotics in order to evolve neural

arrays dominating a robot arm with five degrees of freedom from an external image acquisition system.

## 9. 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, (Octubre 2001), pp. 1105-1115.

[3] Corbalán Leonardo. Evolución de redes neuronales para comandar criaturas que alcanzan objetivos sorteando obstáculos en un entorno virtual 2D. *Tesis de grado correspondiente a la carrera Lic. en Informática*. UNLP. Marzo 2002.

[4] Corbalán Leonardo y Lanzarini Laura. Arreglos neuronales evolutivos aplicados a la evasión de obstáculos y alcance de objetivos. *VIII Congreso Argentino de Ciencias de la Computación CACIC 2002* (octubre 2002) - *XXVIII Conferencia Latinoamericana de Informática CLEI 2002* (noviembre 2002).

[5] Corbalán Leonardo y Lanzarini Laura. An ENA-Based Strategy Replacing Subobjectives Definition in Incremental Learning. *International Conference on Computer Science, Software Engineering, Information Technology, e-Business, and Applications, (CSITeA'03).* Junio 2003, Río de Janeiro, Brasil.

[6] Freeman, J. A. & Skapura, D. M. Redes neuronales Algoritmos, aplicaciones y técnicas de programación. Addison–Wesley, 1991. Versión en español de: Rafael García -Bermejo Giner. Addison–Wesley Iberoamericana 1993.

[7] Goldberg, D. E. Genetic Algorithms in Search, Optimization, and Machine Learning. Addison–Wesley, 1989. pág. 10

[8] 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.

[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] 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.

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

[14] 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.