ARTÍCULO

Modular Creation of Neuronal Networks for Autonomous Robot Control [1]

Germán L. Osella Massa, Hernán Vinuesa, Laura Lanzarini

Instituto de Investigación en Informática LIDI
Facultad de Informática, Universidad Nacional de La Plata
50 y 115, 1er piso. (1900) La Plata, Buenos Aires, Argentina
{ gosella, hvinuesa, laural } @ lidi.info.unlp.edu.ar

Abstract

In general, complex control tasks can be solved by dividing them into simpler ones which are easier to handle. Several authors have developed different solutions that combine Layer Evolution techniques with Evolving Neural Networks, giving rise to controllers made up by several networks. In this type of solution, the selection of the module to be used in each case is not an easy problem to solve. This paper is focused on a new evolutionary mechanism that allows combining modules which solve the different parts of a problem, giving place to a single recurrent neural network. In this way, simple modules which are trained independently of the problem to solve are used. The communication among them is established by evolution, which gives rise to a single neural network representing the expected solution. The proposed method in this paper has been used to solve the problem of obstacle evasion and target reaching using a Khepera II robot. The tests carried out, both in the simulated environment and over the real robot, have yielded really successful results.

Keywords: Evolutionary Neural Networks, Evolutionary Robotics, Modular Evolution

1. Introduction

Evolutionary Algorithms have proved to be highly useful to solve control problems. However, when dealing with complex tasks, it is difficult to find a good solution in a reasonable time.

A complex task refers to that whose solution is not simple but involves learning a strategy to achieve the expected objective. Problems like prey capture and target reaching belong to this category [4].

In addition, there exist situations which cannot be solved by a single agent. Such is the case of robotic football or the prey capture problem, in which the predator is slower than the prey. In both cases, beyond the differences among agents, the group is in charge of carrying out the strategy [11].

In order to solve this type of problems, different approaches that divided the original problem into simpler parts have been proposed [6] [8]. Even though the existing methods vary in the way they acquire knowledge, most of them adopt a strategy based on the evolution and combination of different modules.
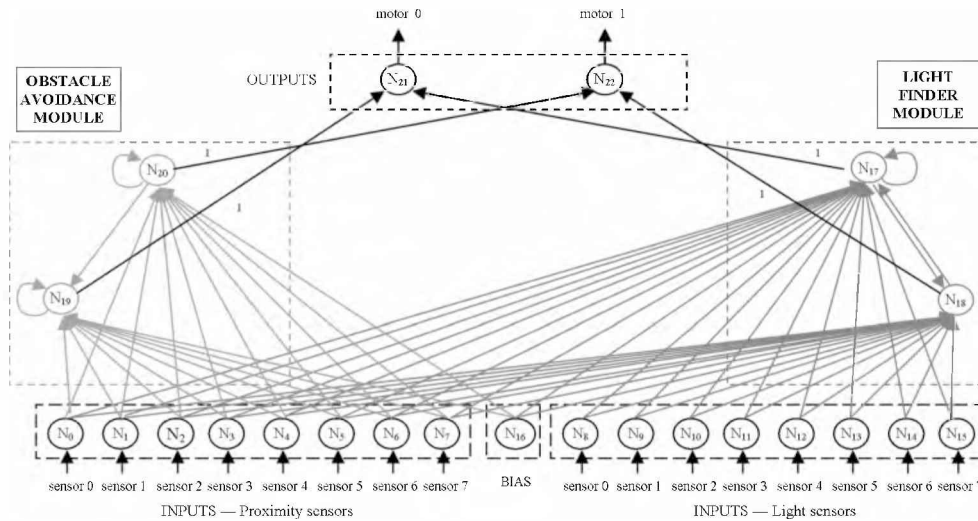
Figure 1: Unified Neural Network obtained after combining two modules with some inputs in common.

In this direction, solutions have been developed to offer adaptive mechanisms that minimize the necessary knowledge to obtain a good controller, giving rise to neural networks composed by several others [1]. As the controller is composed by several modules, the way to determine which neural network should be run at each time instant is important [12][13]; in this line, there exist different alternatives, which range from the use of an ad-hoc designed decision tree [5] to mechanisms that organize the structure automatically [2].

2. Objective

This proposal is based on the decomposition of the original problem into simpler independent tasks. Each task is solved by a different module which, after being individually obtained, is combined into a single structure.

This paper presents a new strategy based on NeuroEvolution that integrates these independent modules into a unified neural network, generating the necessary interfaces between each module to produce a controller that is capable of solving a complex task.

This approach allows reducing the training time necessary to obtain a solution to the complete problem, since integrating each part implies an easier adaptation. Also, as a consequence of this decomposition, general reusable neural modules are produced.

Section 3 describes the proposed method giving also a description of the method on which is based. Section 4 presents the problem to solve including some implementation aspects. Both of the independently evolved modules are described in Section 5. The results of applying this method to solve an obstacle evasion and target reaching problem using a Khepera II robot are presented in Section 6. Finally, some conclusions and future work are presented in Section 7.

3. Modular NeuroEvolution

As previously described, certain type of complex tasks may be considered as the combination of several, simpler tasks. If each of these simpler tasks is successfully and independently solved, it should be possible to combine such solutions to complete the complex task. On this assumption, this paper presents an extension to the NeuroEvolution of Augmenting Topologies (NEAT) method [9], which incorporates the concept of modules.

This proposal assumes that there exists a set of neural networks in which each of them, called a module, is capable of solving one of the simple tasks. The objective of this work is focused on getting a unified neural network, constituted by the combination of all of these modules, capable of solving the complex task. A brief description of the original NEAT method will be presented in order to understand the proposed extension.

3.1. NeuroEvolution of Augmenting Topologies

The standard NEAT implementation has been shown to be a highly effective NE method in several domains [10]. It addresses three problems commonly found in ANN systems: 1) how to crossover topologically disparate chromosomes, 2) how to protect new topological innovation, and 3) how to keep topologies as simple as possible throughout evolution [9]. This is accomplished through historical markings, speciation, and incremental complexification.

First, each genome in NEAT includes a list of connection genes, each of which refers to two node genes being connected. In order to perform crossover, the system must be able to tell which genes match up between any two individuals in the population. For this reason, NEAT keeps track of the historical origin of every gene. Two genes that have the same historical origin represent the same structure (although possibly with different weights), since they were both derived from the same ancestral gene from some point in the past. Tracking the historical origins requires very little computation. Whenever a new gene appears (through structural mutation), an innovation number is incremented and assigned to that gene. The innovation numbers thus represent a chronology of every gene in the system, and allows crossover of diverse networks without extensive topological analysis. With historical markings the problem of having to match different topologies [7] is avoided.

Second, NEAT networks are speciated so that individuals compete primarily within their own niche. This way, topological innovations are given time to optimize their structure before they have to compete with the entire population. Also, networks share the fitness of their species [3], to prevent one species from taking over the entire population.

Third, NEAT networks are built from a minimal configuration and complexified incrementally to ensure that solutions of minimal complexity are searched first. This procedure has two advantages: First, it minimizes topology bloat, and second, it improves the efficiency of evolution by complexifying the search space only as needed.

For more details about NEAT, see Stanley and Miikkulainen [9].

3.2. NEAT with Modules

The incorporation of neural modules to the NEAT method implies carrying out several modifications. The first is related to the neural networks that compose the initial population. In the original proposal, it is assumed that there is no enough knowledge of the problem to specify the topology of those networks. In addition, starting with minimal networks allows the method to explore first the simpler solutions. In this extension, networks solving different parts of the problem are known and it is possible to fill the initial population with variations of a unified neural network. This network is built up from merging each of the available modules within a same structure.

Since the tasks solved by each module are part of a single complex tasks, it is expected that more than one module will use the same inputs or produce the same output. The unified neural network will have as input the union of the inputs of each module. The modules are connected to those inputs without undergoing any modification. The unified network outputs depends on the task to solve and for this reason the network will have as many output neurons as the problem needs.

More than one module may generate the same output of the network. It is also possible that different modules produce opposite stimuli for similar inputs, since the tasks solved by each of them may be contradictory. To allow the evolution to adjust the contribution of each module to the unified network outputs, rewarding expected responses and making compatible opposite stimuli, each module output neurons become hidden neurons. To each of these converted neurons, a new connection is added that link this neuron to the output neuron that produces the response which was originally yielded by former neuron. The connection is established with weight 1.0, so the original stimulus reaches the output neuron without being affected. This new connection is not considered as part of any module, but belongs to a unified neural network. Figure 1 shows the combination process of two modules to produce a unified neural network.

During the building process of the unified network, each connection and neuron integrated into the network is marked with an identifier associated to the module which it belonged. This is done to simplify the tracking of the modules that composes each network once the evolution has started.

Figure 2: Khepera Robot II next to a coin of € 2.

Figure 3: Detail of the sensors.

Another proposed modification to NEAT is the way in which genetic operators are applied to produce new genomes. Originally, the mutation operator was in charge of generating innovations, perturbing weights, establishing new connections among existing neurons, or inserting a new neuron after dividing an existing connection.

In this paper, the mutation operator scope has been restricted. It is only possible to modify the weight of a connection if it didn't originally belong to any of the modules making up the network undergoing mutation. In the same way, it is not allowed to establish new connections among neurons of the same original module, being only valid to do so among neurons of different modules. Eventually, it is only possible to add a neuron if an existing connection is previously divided, which, once again, should not be a connection contributed by any of the modules. These restrictions force the evolutionary method to generate the necessary structure to allow the original modules to interact so they together can reach the solution of the posed complex task.

The rest of the evolving method is not different from standard NEAT; historical markings are kept in the genomes of the population, the original crossover operator is used, and the population is divided into species according to a compatibility criterion, dividing the fitness of each member proportionally to the number of genomes belonging to the same niche.

The reason for which the topology and connection weights of each module cannot be changed is due to the fact that, since these are fixed, the evolutionary algorithm will search in a more reduced space than if it were to do it over an entirely mutable network. This should favor a faster convergence towards better solutions.
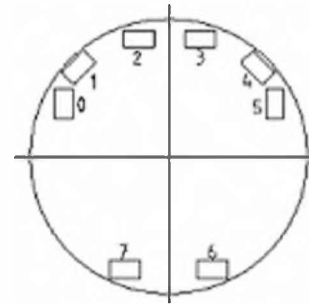
It is worth noting that bigger structures could be generated compared with the ones that could be obtained if started from a minimum topology. However, when the difficulty of the task increases, the complexity of the neural network proportionally increases, and generating a structure which acts as interface between the modules is simpler than solving the whole problem.

4. Experimental Setup

4.1. Khepera II Robot

The robot used is a Khepera II, see Figure 2, which has 8 sensors and 2 DC motors controlling each wheel. The 8 sensors are placed as Figure 3 shows, and each of them can be used in two different ways: as an object proximity sensor and as an environment lightness sensor. In the first case, sensors are used as senders and receivers of infra-red pulses, and in the second, only the reception function is used. In both cases, the returned values are between 0 and 1023. The closer the proximity sensors are to an object, the higher the value obtained. The detection range is between 0 and 10 cm, depending on the infra-red reflection capacity of the object sensed.

The higher the lightness intensity, the lower will be the value yielded by the light sensors and 1023 represents completely darkness. The detection range depends on the source intensity. For example, a source of 50W at 40 cm can produces significant values in the sensors.

Motors' speed is given as an integer value between -128 and 127, in which negative values make the robot move back, positives, move forward, and 0 makes the robot remain still. The higher the value applied to a motor, the higher the speed.

```
function EvaluatePopulation
  for each genome of the population
    Create a controller from this genome
    for i = 1 : 4
      Place the robot at position i
      Compute the controller's score
      using the appropriate Eval
      function.
      If during this evaluation the robot
      collides, the trial is interrupted
      and the current Eval value is
      returned with what is gathered up
      to this point.
    end for
    Calculate the genome's fitness as the
    average of the 4 previous scores.
  end for
end function
```

**Figure 4: Algorithm used to evaluate
the population in 4 different trials.**

4.2. YAKS: Yet Another Khepera Simulator

YAKS is a simulator of a Khepera II written in C++, which not only has a simulated environment but it can also be used to control a real robot connected to a PC through an RS-232 interface. To simulate the object or light recognition process, as well as robot movements, the simulator has samples tables, where each of them is used for recognizing a particular object. These tables were generated from measurements made on a real robot, which gives more realism to the simulator. Moreover, the simulator allows adding noise to the sensors.

The simulated robot, as well as the real one, has 8 proximity sensors whose values are normalized within range $[0, 1]$. It also has 8 light sensors with values also normalized between $[0, 1]$, where darkness is represented by 0. For security reasons, motors are not allowed to move at a speed higher than 10. The values for the motors are also normalized within range $[0, 1]$ where 0 means it runs backwards at speed 10, 1 means it runs forward at speed 10, and 0.5 makes the motor stop.

4.3. Problem Description

The goal is to evolve a controller based on recurrent neural networks to provide to a Khepera II robot with the capacity of avoiding obstacles and reaching specific targets. Since only light and proximity sensors of the robot have been used, the targets will be represented by white lights placed at any position within their environment. The robot is expected to move freely, without colliding with obstacles, until it reaches a lighted area in which it will try to remain.

Using the proposed extension of NEAT previously described, two modules independently obtained will be combined: one allowing evading obstacles, and another allowing reaching the nearest light. Each of these modules is represented by a recurrent neural network generated by the standard NEAT method [9].
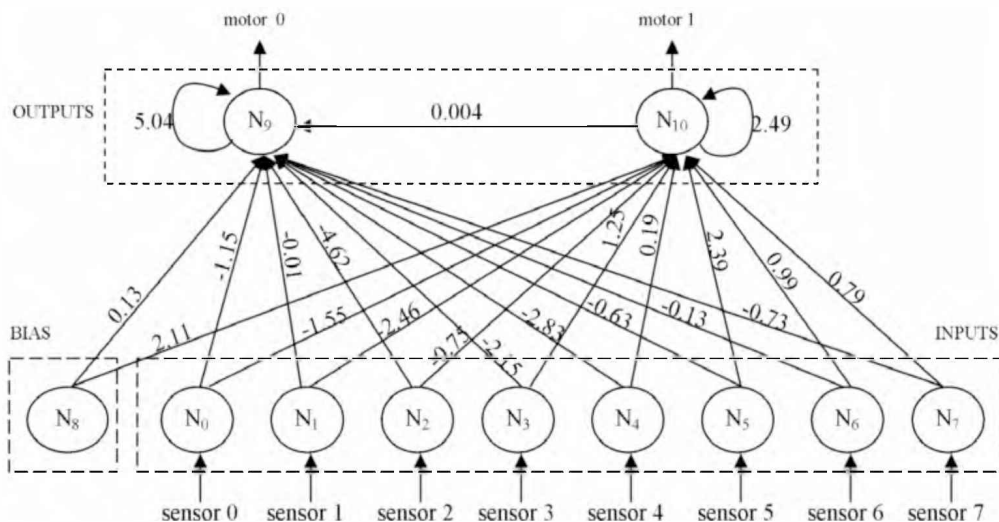
Figure 5: Topology of the best performing controller for the obstacle evasion problem.
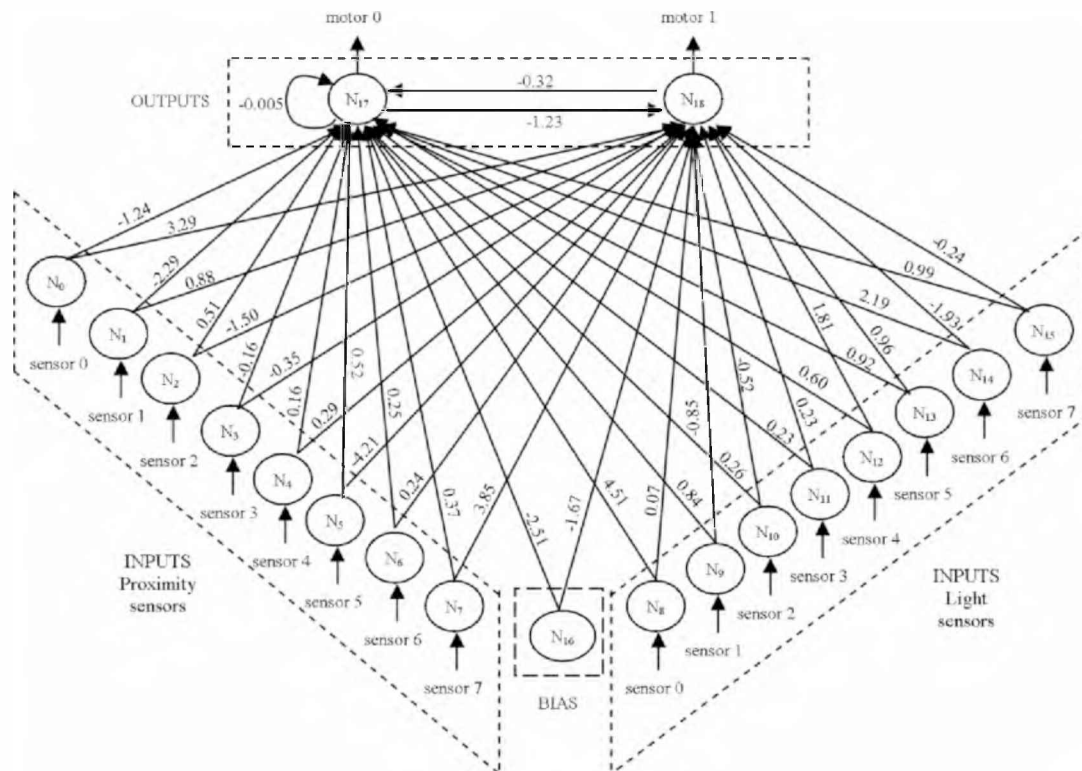
Figure 6: Topology of the best performing controller for the light finder problem.

The strategy proposed in this article, even though applied to obtain a controller from two previously generated behaviors, can be easily extended by incorporating additional modules with their corresponding fitness functions. In this way, having previously trained independent modules eases obtaining more complex controllers.

5. Evolved Modules

Both the obstacle evasion module and light finder module have been evolved using the standard NEAT method running for 500 generations each, using the same parameters: 150 genomes in the population, 90% crossover rate, 4% chance to add a hidden node, 7% chance to add a link with a 5% chance of adding a recurrent link, 20% weight mutation with a 10% probability of replacing the value.

In both cases, the fitness of a genome is computed averaging the score obtained in four different trials.

Each trial differs in the initial position of the robot within the environment. Since tests have been carried out in a rectangular maze, each trial starts with the robot placed at a different corner. Figure 4 contains the pseudo code of the algorithm used to evaluate the population. In the next subsections, a description of each module with the utilized evaluation function will be presented.

5.1. Obstacle Evasion Module

The initial neural network is composed by 8 linear input neurons, two non-linear output neurons, and an additional bias neuron which can connect itself to any other neuron with the exception of an input one. The inputs to the network are linearly scaled to the range [0, 1] from the values captured by the sensors. The outputs of the network are scaled between [−1, 1] to control the speed of the motors driving each of the robot wheels to fit the simulator requirements.

The following evaluation function is used to measure the score obtained in each trail.

$$Eval_{Obs} = K \times \sum_{t=1}^{500} \left[(M_l + M_r) \times (1 - |M_l - M_r|) \times (1 - S_{ir}) \right] \quad (1)$$

where

M_l and M_r are values in the interval $[-1, 1]$ corresponding to the left and right motor speeds, respectively. These are the network outputs.

S_{ir} is the maximum value of the proximity sensors in the interval $[0, 1]$.

K is a value proportional to the area covered by the agent during the training.

The optimization of the term $(M_l + M_r)$ pushes the controller to maximize its movement since the highest value is obtained when the robot goes forward at the maximum speed. The term $(1 - |M_l - M_r|)$ refers to the robot's rotation. If the robot is spinning on its axis, the speeds of the motors are opposite. The higher the rotation, the lower the value of this term is. The controller needs to minimize this effect in order to increase its score. Finally, the term $(1 - S_{ir})$ forces the robot to move away from the obstacles to increase its score.

To obtain controllers capable of covering large distances, the environment was divided into a grid of 100x100 equal sectors, and the coefficient sectors was used to measure the territory which the robot covered throughout the test.

$$K = \frac{\sum_{x=1}^{100} \sum_{y=1}^{100} sector_{xy}}{100 \times 100} \quad (2)$$

$$sector_{xy} = \begin{cases} 1 & \text{if the agent covered } sector(x, y) \\ 0 & \text{otherwise} \end{cases}$$

In summary, the robot's run is weighted in (1) along 500 steps and scaled proportionally to the number of covered sectors. Figure 5 shows the topology of the best performing controller obtained after 30 independent runs.

5.2. Light Reaching Module

This module has 16 inputs, 8 for each robot's proximity sensors and 8 for each of the light sensors, and two output neurons, one for each of the motors.

Like in the previous module, the network inputs are linearly scaled to the range $[0, 1]$ and the outputs, between $[-1, 1]$. Again, a minimal topology is used when the evolution begins.

In order to measure the controllers' score during each trial, the next evaluation function is used:

$$Eval_{Light} = \sum_{t=1}^{500} \left[\alpha \times (M_l + M_r) \times (1 - |M_l - M_r|) + \beta \times S_{light} \right] \quad (3)$$

where

M_l and M_r are the same values described in the previous subsection.

S_{light} is the maximum value of the light sensors in the interval $[0, 1]$, where 0 corresponds to the absence of light and 1 to a full detection.

α and β are constants used to regulate the dominant behavior.

In (3), the robot's movement is weighted the same way as in the previous module: The controller will attempt to maximize its movements minimizing rotations. However, adding a quantity proportional to the highest value sensed from the light sensor permits a controller to be acceptable even if it does not move. This allows the robot to stop moving and stay near the light. Figure 6 shows the architecture of the best controller obtained after 30 independent runs.

6. Results

Modules represented in Figures 5 and 6 are integrated to a single neural network following the method described in section 3.2. Figure 1 shows the resultant topology. It can be noted that both modules share the same information provided by the proximity sensors input neurons.

The unified neural network combines the output of each module through two new neurons which are in charge of controlling the robot's motors. In this way, the network is composed by 16 inputs, 8 for each robot's proximity sensor and 8 for each light sensor, 4 hidden neurons and 2 output neurons, one for each motor.
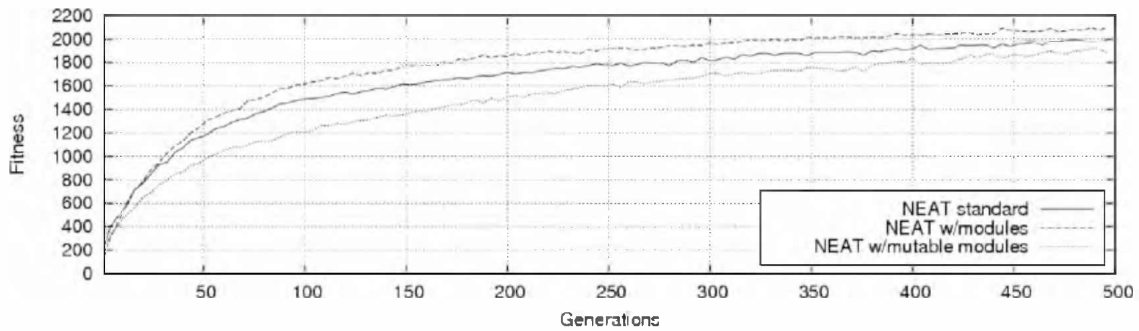
Figure 7: Average best fitness values per generation for each of the tested methods
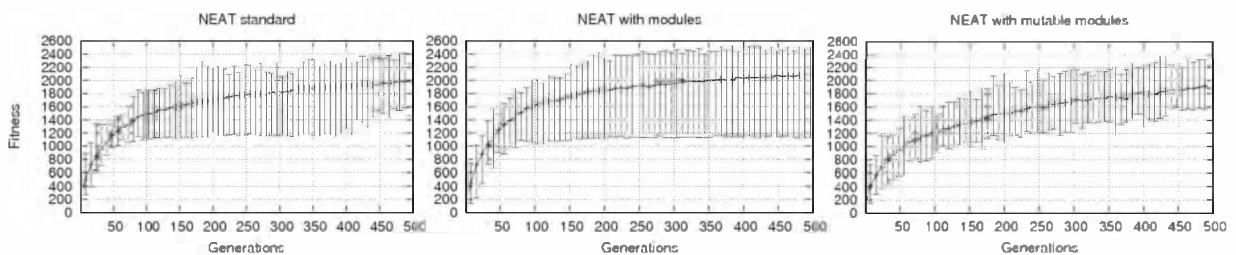

Figure 8: Average best fitness values per generation with minimum and maximum values for each of the tested methods.
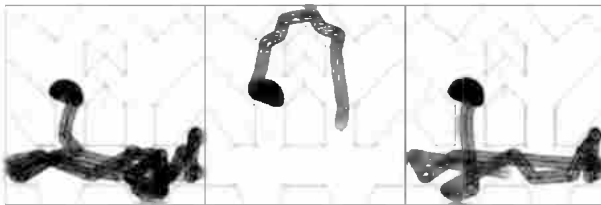

Figure 9: Behavior of the best individual generated using standard NEAT.
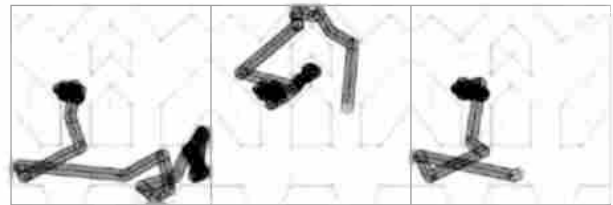

Figure 10: Behavior of the best individual generated using NEAT with modules.

The initial population is created with networks whose genomes are produced applying the mutation operator defined in section 3.2 to the unified neural network.

As explained in that section, the population is evolved according to the NEAT method with restrictions imposed to the mutation operator.

The fitness of a controller is calculated combining the evaluation functions of each of the original modules into a single expression. In this way, both of the original behaviors are measured simultaneously. The resulting evaluation function is:

$$Eval = c_{obs} \times Eval_{Obs} + c_{light} \times Eval_{Light} \quad (4)$$

where

$Eval_{Obs}$ and $Eval_{Light}$ correspond to the controller's score for the obstacles evasion and light reaching tasks, respectively.

c_{obs} and c_{light} are constants that balance the relative weight of each of the scores of the original tasks.

In this work, c_{obs} and c_{light} are 1 and 1.3 respectively, to put more emphasis on light seeking (+30%) over obstacle avoidance because to reach a light, there is an implicit pressure in its evaluation function to traverse the environment. The same happens with the obstacle avoidance function.

If both evaluation functions are added without being scaled, the controller which travels more distance will have a better score, diminishing the importance of being near a light.

The method proposed in this paper was compared to standard NEAT and to an intermediate version which makes use of the same initial topology than the version with modules does (like the example of Figure 1), but allows the entire architecture to evolve, i.e., all the genetic operators of the standard NEAT are used without restrictions.

In order to compare the performance of the three alternatives presented, 30 runs were carried out, independent one from the other; each of them for 500 generations using the same parameters as before. Figure 7 shows the average of the best fitness values obtained per generation in each kind. Figure 8 shows each of these results separately, including the minimum and maximum fitness obtained, to easily visualize the dispersion of these values per generation.

It is worth to mention that the proposed method outperforms the standard NEAT method; both in the

number of generations necessary to reach a given fitness value and in the maximum value reached. Also, starting the evolution with unified networks that are completely mutable is not better than the standard NEAT performance but worse.

This can be justified if the initial topology is taken into account. When an evolution is started with a more complex mutable neural network, the search space is larger that when started with a simpler one. Thus, finding the suitable combination for the topology and weights is a more difficult task and it may require more time to reach the same fitness.

Figure 9 shows the performance of the best individual found in the 30 runs of the standard NEAT method while Figure 10 shows the best of the proposed extension. In the three trials, the robot began from positions different from the ones used during the evaluation of its fitness.

Figure 11 shows the topology of the controller generated by this proposal. Three new hidden neurons have appeared interconnecting both modules and interacting with the neural network's inputs and outputs.
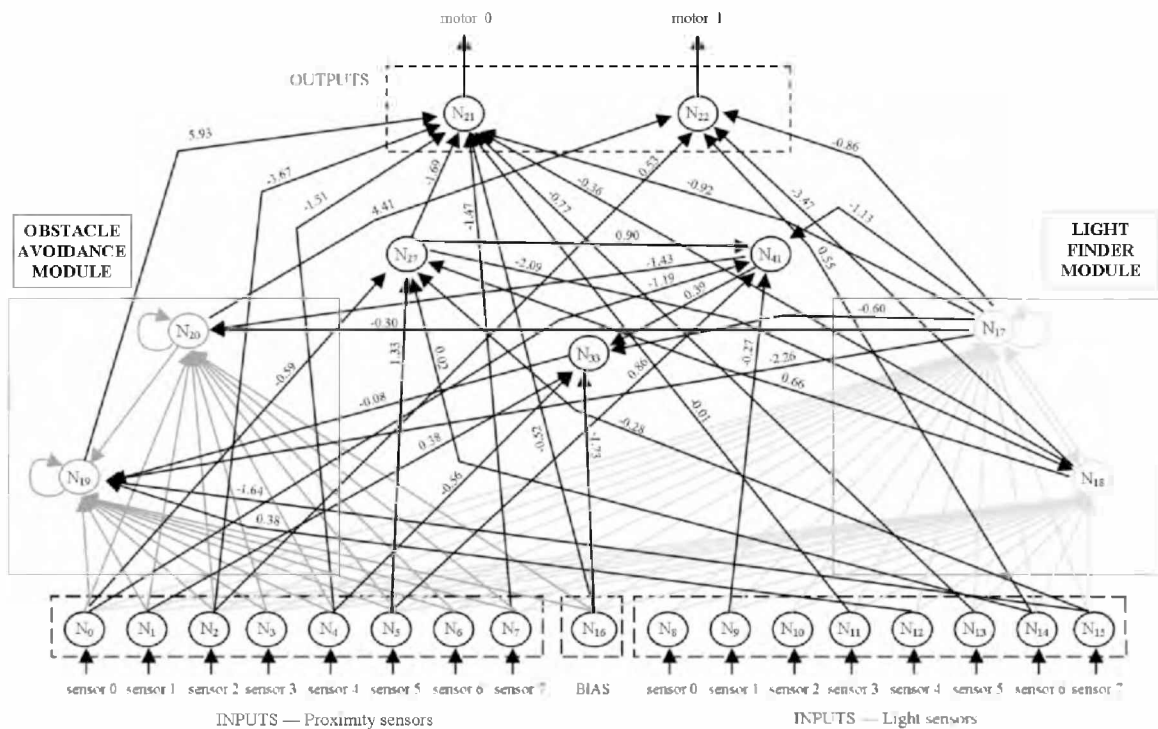

Figure 11: Best controller generated with the proposed method.

7. Conclusions and Future Work

Evolutionary techniques, though capable of providing excellent results in several areas, present some unfavorable characteristics: the adaptation process may be slow and costly in running time for some problems; and it is difficult to reuse the acquired knowledge.

This paper has presented a strategy which aims at solving both problems, proposing the combination of simple modules, based on recurrent neural networks, generated independently of the problem to solve. Finally, the communication among them is established by evolution, which gives rise to a single neural network representing the expected solution.

The results obtained for the obstacle evasion and light reaching problems using a Khepera II robot have proved that the proposed strategy is more efficient. The reason for this improvement may be attributed to the fact of incorporating partial solutions, instead of not using any previous knowledge.

An automatic mechanism suitable for combining each module's fitness function still has to be established. In this paper, each evaluation function had to be scaled to make them all comparables (within the same interval), to avoid the fact that one of the functions may diminish the others. In addition, the election of the coefficient was manually made.

In future works, the performance of the proposed method will be measured in problems with higher complexity which can be decomposed in a higher number of modules. In order to simplify the definition of the fitness function for the unified network, the possibility of incorporating multi-objective evolution techniques will be studied. This will permit to obtaining a set of neural networks that are in the Pareto frontier. As they all represent variations of the expected behaviors, each maximizing a different aspect, it will be useful to select the most appropriate one.

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. pages 251-257, 2001.

[2] Corbalán L., Osella Massa G., Lanzarini L., De Giusti A. ANELAR. Arreglos Neuronales Evolutivos de Longitud Adaptable Reducida. *X Congreso Argentino de Ciencias de la Computación.* CACIC 2004. Universidad Nacional de La Matanza. Buenos Aires. Argentina. ISBN: 987 9495 58 6. 2004

[3] Goldberg, D.E., Richardson, J. Genetic Algorithms with Sharing for Multimodal Function Optimization. *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application.* Cambridge, Massachusetts, United States, pages 41-49. 1987

[4] 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, pages 317-342, 1997.

[5] Olivera J., Lanzarini L. Cyclic Evolution. A new Strategy for Improving Controllers Obtained by Layered Evolution. *Journal of Computer Science and Technology.* Vol 4, nro. 1, pages 211-217, 2005.

[6] Lara, B, Hüsle, M., Pasemann, F. Evolving Neuro-Modules and their interfaces to Control Autonomous Robots. MPI-MIS-Preprint 21, accepted paper for "*The 5th World Multi-Conference on Systemics, Cybernetics and Informatics*" (SCI2001), Orlando, Florida USA, July 22-25, 2001.

[7] Radcliffe, N.J. Genetic Set Recombination and its Application to Neural Network Topology Optimization. *Neural computing and applications,* Vol. 1, pages 67-90, 1993.

[8] Reisinger, J., Stanley, K.O., Miikkulainen, R. Evolving Reusable Neural Modules. *Proc. of the Genetic and Evolutionary Computation Conference,* 2004.

[9] Stanley, K.O., Miikkulainen, R. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, Vol. 10, pages 99-127, 2002.

[10] Stanley, K.O., Miikkulainen, R. Competitive CoEvolution Through Evolutionary Complexification. *Journal of Artificial Intelligence Research*, 21. 2004.

[11] Stone P., Veloso M. Multiagent Systems: A Survey from a Machine Learning Perspective. *Autonomous Robots*. Vol. 8, n. 3, pages 345-383, 2000.

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

[13] Yao, X. Evolving Artificial Neural networks. School of Computer Sciencie The University of Birmingham Edgbaston, Birmingham B15 2TT. *Proceedings of the IEEE*. Vol. 87, No. 9, pages 1423-1447, 1999.