Improving Controllers based on Neural Networks obtained by Layered Evolution

Hernán Vinuesa[1], Laura Lanzarini[2], Germán Osella Massa[3]

III-LIDI (Institute of Research in Computer Sciences LIDI)
School of Computer Sciences. National University of La Plata
La Plata, Argentina, 1900
{hvinuesa, laural, gosella}@lidi.info.unlp.edu.ar

Abstract. Complex task solving can be carried out by decomposing the original problem into more specific and simpler parts. Several authors have shown that those cases where incremental learning may occur can be successfully solved through the incremental layered evolution paradigm. This paper is focused on the presentation of a new mechanism, which allows improving controllers based on neural networks obtained through layered evolution. Its functioning is based on the combination of an evolving strategy capable of generating minimal structure neuronal networks through a tournament selection mechanism. The proposed method has been applied to the generation of a controller allowing a robot to find a ball, correctly stand behind it and hit it towards a specific place. Tests performed both in the simulated environment and upon the real robot have given quite satisfactory results.

Keywords: Evolving Neural Networks, Layered Evolution, Evolutionary Robotic.

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 reasonable time. Several researches have demonstrated that certain complex tasks may be solved by using layered evolution [1][2].

A complex task refers to one 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 [3]. In these cases, it is hard to set in advance the controller to be used, and here is where layered evolution becomes important. This process consists in decomposing the original problem into simpler parts, called subtasks, thus allowing for a gradual learning of the expected response [4].

On the other hand, unless we count with the necessary initial information to solve each subtask, it is ideal to count with some mechanism that allows carrying out the

[1] Becario CIC. Auxiliar Docente. Facultad de Informática. UNLP
[2] Profesor Titular DE. III-LIDI Facultad de Informática. UNLP
[3] Becario Doctoral de CONICET. Jefe de Trabajos Prácticos - Facultad de Informática. UNLP

adaptation as automatically as possible. In this way, different solutions combining techniques of Incremental Evolution with Evolving Neural Networks have been developed with the aim of providing an adaptation mechanism that minimizes the needed previous knowledge to obtain an acceptable performance giving raise to controllers made up of several networks [5]. Another aspect to take into account is the way of determining which neural network should be run at each instant of time [6][7]; thus, there are several alternatives ranging from the use of an ad-hoc design decision tree [8] to mechanisms automatically organizing the structure [9].

2 Objective

This research is based on works previously carried out in the fields of layered evolution [10] [11] through neuroevolving algorithms and proposes an alternative which allows obtaining improvements in the proposed solutions.

The purpose of this paper is to present a new evolution-based strategy through which controllers for solving each part of the problem can be efficiently obtained. The adaptation process not only allows achieving the expected behavior but also automatically determines the needed minimal structure for each controller.

This paper is organized as follows: Section 3 specifies the proposed strategy in detail; Section 4 describes the problem to solve; Section 5 presents some implementation aspects; Section 6 summarizes the results obtained; and Section 7 shows the conclusions together with some working future lines.

3 Proposed Strategy

The adaptation strategy proposed in this paper permits to obtain a controller formed by as many recurrent neuronal networks as defined subtasks. Each network is obtained through a layered evolution based on the dependency established among subtasks. The method used to carry out this adaptation process not only allows achieving the expected behavior but also automatically determines the needed minimal structure for each case.

Earlier studies [12] have shown that NEAT (NeuroEvolution of Augmenting Topologies) has enough capacity to solve this type of situations. However, the computation time used to obtain the proper neuronal network to solve each subtask may be excessive. In Subsection 4.1 a brief summary of the most significant features of the first method has been included.

Hence, this paper proposes to carry out the evolution in two parts; the first one by the NEAT method and the second one by a Binary Tournament applied to all individuals in a population. The following pseudocode specifies this process.

```
Be C=[$c_1$,$c_2$,..,$c_n$] the list of controllers to obtain
ordered according to their dependencies.
Let $O_i$ be the target of controller $c_i$ with i=1:n
For each controller $c_i$, with i of 1 a n.
  Generate a random initial population.
  Evolve using NEAT for a minimal number of
  generations.
  While (a minimal number of generations is not
  achieved) and (objective $O_i$ is not accomplished)
    Carry out tournaments between pairs of individuals
    randomly selected from the whole population.
    The number of pairs corresponds to 45% of the
    population size.
    The new population will be made up of a 10% of the
    individuals with best fitness from the previous
    populations (elitism).
      The winners of the binary tournaments.
      The new individuals obtained when applying
      uniform mutation to the arcs of the networks of
      each of the winners of the tournament.
  End While
End For
```

3.1 NeuroEvolution of Augmenting Topologies

NEAT implementation has proved to be a highly effective Neuro-Evolution method in several domains [13]. It addresses three problems commonly found in Neural Network 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 [14]. This is accomplished through historical markings, speciation, and incremental complexification.

First, each genome in NEAT includes a list of connection genes, each of which referring 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 (though 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. Thus, the innovation numbers represent a chronology of every gene in the system, and allow crossover of diverse networks without extensive topological analysis. With historical markings the problem of having to match different topologies [15] is avoided.

Second, NEAT networks are speciated so that individuals compete primarily within their own niche. In this way, topological innovations are given time to optimize their structures before they have to compete with the entire population. Also, networks share the fitness of their species [16] 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 [14].

4 Problem Description.

The method proposed in this paper has been applied to the generation of a controller allowing a Khepera II robot to find a ball in a play field and put it in the goal area. The play takes place in a rectangular field from which neither the ball nor the robot can come out and finishes when the robot is able to make a goal.

Figure 1 shows the field where the play takes place. Two independent runs followed by the robot to reach the position allowing it to hit the ball toward the goal or interest area are illustrated.

4.1 Problem Decomposing into Simpler Subtasks

This play can be decomposed into three subtasks. Each task is carried out by a different neuronal network obtained by evolution:

- Search: The purpose of this neuronal network is to provide the robot with the capacity to explore the field until locating the position of the ball and then come closer to it.
- Position: This neuronal network is responsible for adequately positioning the robot. Since the Khepera II used does not have any additional support to “hook” the ball, it is fundamentally significant that it remains correctly in line with the ball and the goal area.
- Hit: the purpose of this neuronal network is to hit the ball as strongly as possible so as to put it inside the goal area.

4.2. Layered Learning

Once the subdivision of tasks is carried out, a dependence order is established among them, which indicates the training sequence. Figure 2 shows these dependencies for the proposed problem.

Each rectangle represents a subtask and the arrows indicate the dependencies among them. A subtask could be learnt once the rest of the subtasks on which it depends have been learnt as well.

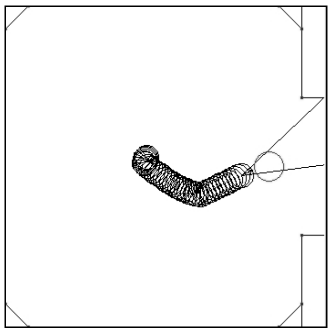

Fig. 1. Simulated Environ.

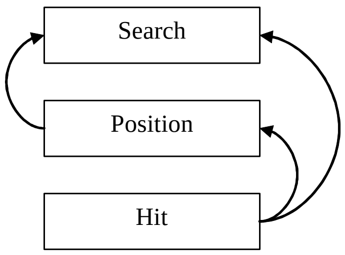

Fig. 2. Dependence layer order.

This is called layered learning based on the dependence existing in the order of learning of different subtasks. From another point of view, it could be regarded as a structure having an initial layer made up of those subtasks which do not need others to be learnt. Then, in the following layer, those subtasks that can be learnt from previous ones are placed, and so on.

Notice that this learning does not show how to solve the whole problem, but the way of learning to carry out each of the expected subtasks.

4.3. Problem Solving

Once the networks are obtained, a decision tree is in charge of selecting the network that should be used at each instant. In this way, a single controller is obtained based on specific controllers for each subtask. Figure 3 shows the decision tree used to solve the game.

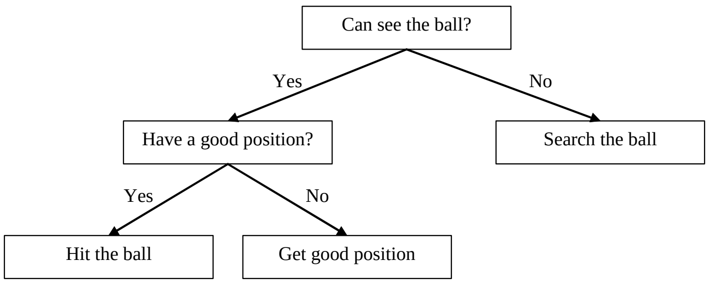

Fig. 3. Decision tree used to solve the problem.

5. Implementation Aspects

The Khepera robot used in all the trials only has one K213 vision camera capable of distinguishing a 64 pixel line corresponding to the grey shades located in its angular vision. For this reason, the rectangular walls of the closed rectangular environment used were painted in black, the ball in white and the goal area in grey. The collisions are detected through the proximity sensors.

The controller of each subtask is commanded by a neuronal network made up of 72 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 neuron. The inputs to the network are linearly scaled to the range [0, 1] from the values captured by the sensors where the first 8 values correspond to the proximity sensors and the remaining 64 correspond to the k213 camera. 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 final architecture to be used is determined following NEAT application.

To establish the fitness of an individual, the performance of its controller is measured starting from 4 different positions. Since the trials have been carried out in a rectangular area, each one starts with the robot at a different corner. The ball position also changes in each case. Finally, the fitness value of an individual is given by the average of the results of the four trials the individual underwent. The following pseudocode shows the algorithm used.

```
for each individual of the population
  for i = 1 : 4
    Locate the individual in position i
    Carry out 500 iterations with the current
    controller.
    Calculate the individual fitness at this stage
    represented by Eval as the addition of the fitness
    of each generation. 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 individual fitness as an average of the 4
previous trials.
end for
```

There follows a detail of how the corresponding fitness has been calculated.

5.1. Search Module

To measure the score achieved in each trial the following evaluation function is used.

$$Eval_{search} = \sum_{t=1}^{500} \left[\left(M_{left} + M_{right} \right) \times \left(1 - S_{ir} \right) \times \sum_{i=1}^{64} \left(camera_i \times vector_i \right) \right] \qquad \textbf{(1)}$$

where:

- *camera*$_i$ is position *i* in the value array corresponding to the k213 camera. It is the interval value [0,1] corresponding to the grey scale where 0 represents black and 1 white.
- *vector*$_i$ is position *i* in the scalar value array with normal distribution. This vector aims at increasing the importance of the central pixels.
- M_{left} and M_{right} are values in the interval [-1, 1] corresponding to the left and right motor speeds, respectively. These are the network outputs.
- Sir is the maximum value of the proximity sensors in the interval [0, 1].

The term *(camera*$_i$ *x vector*$_i$*)* gets its highest value when the robot gets as close as possible to the ball and the ball is located as close as possible to the center of the camera vision angle. The term pushes the controller to maximize its movement since the highest value is obtained when the robot goes forward at maximum speed. Finally, the term forces the robot to move away from the obstacles to increase its score.

5.2. Position Module

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

$$Eval_{position} = sectors \times \sum_{t=1}^{500} \left(\left[\left(M_{left} + M_{right} \right) \times \left(1 - \left| M_{left} - M_{right} \right| \right) \times \left(1 - S_{ir} \right) \right] + dist_{ball} + dist_{goal} \right) \tag{2}$$

where

- M_{left}, M_{right} y S_{ir} coincides with 5.1.
- *sectors* is a value proportional to the area covered by the agent during the training.
- $dist_{ball}$ is a value in the interval [0,1] indicating distance to the ball.
- $dist_{goal}$ is a value in the interval [0,1] indicating distance to the goal area.

The term $\left(1 - \left| M_{left} - M_{right} \right|\right)$ 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. The controller needs to minimize this effect in order to increase its score.

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

$$sectors = \frac{\sum_{x=1}^{100} \sum_{y=1}^{100} sector_{xy}}{100 \times 100} \qquad sector_{xy} = \begin{cases} 1 & \text{if the agent covered } sector(x, y) \\ 0 & otherwise \end{cases} \tag{3}$$

In summary, the robot's run is weighed in (2) along 500 steps and scaled proportionally to the number of covered sectors.

5.3. Module to hit the ball

$$Eval_{kick} = \sum_{t=1}^{500} \left[\left(M_{left} + M_{right} \right) \times \left(1 - \left| M_{left} - M_{right} \right| \right) \times \left(1 - S_{ir} \right) \times \sum_{i=1}^{64} \left(camera_i \times vector_i \right) \right] \tag{4}$$

where:

- $camera_i$ is position *i* in the value array corresponding to the k213 camera.
- $vector_i$ is position *i* in the scalar value array with normal distribution.
- M_{left} and M_{right} 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].

6. Results

In order to determine the efficiency and efficacy of the proposed method the following alternatives have been taken into account.

a) **Controller based on feedforward neuronal networks:** In each case, neuronal network structure used was the most efficient feedforward architecture that could be manually defined. Training was carried out through a binary tournament.

b) **Controller obtained using NEAT only:** this way of determining controller does not require any previous knowledge of neuronal network architecture since it has the capacity to determine it during adaptation.

c) **Controller obtained by combining NEAT and tournament selection:** this alternative is the proposal of this paper and corresponds to what has been detailed in Section 3.

In this case, 30 independent runs of the process needed to obtain the whole controller were carried out. This implies that in each run the correspondent controllers, applying 100 generations for each one, were generated.

In order to measure the capacity of each method to generate the controller, 30 trials were taken into account to choose the individual that better solved the subtask of hitting the ball for each one.

With these three controllers, 40 trials, considering the number of times in which the robot was able to make a goal were carried out. Figure 4 shows the success percentage out of the 40 runs mentioned above.

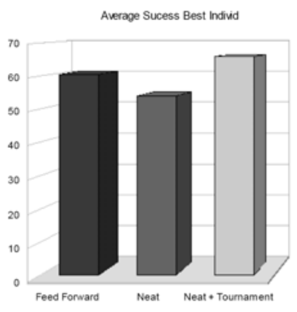

Fig. 4. Average success of the best individual.

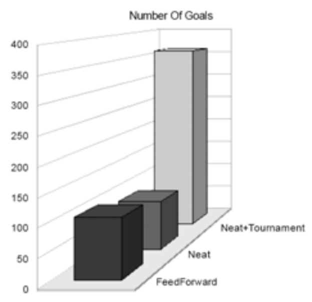

Fig. 5. Number of goals in the last generation.

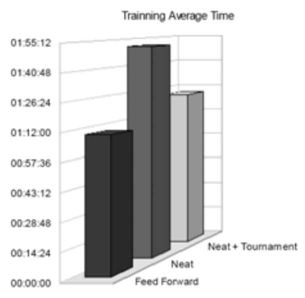

Fig. 6. Average evolution time.

If the behavior of each controller population is analyzed, regarding the number of goals each controller has made, it can be noted that the NEAT + Tournament behavior is quite superior. Figure 5 shows the amount of success of the whole population in the last generation.

Regarding computation time, figure 6 shows the average evolution time of the different methods. It is clearly seen that while NEAT evolution time is closer to two hours, NEAT + Tournament has a 1 ½ hour average. These time measurements correspond to the algorithm execution in 2.4 GHz Pentium 4 machines.

In order to measure the improvement introduced by the method proposed in this paper, the behavior of the best controllers obtained with NEAT and NEAT + tournament was analyzed every 10 generations of the evolution process, using them to hit the ball 100 consecutive times, trying to introduce it in the goal area.

Figure 7 shows the average values corresponding to the goals made by the controllers in the previously indicated generations during 30 independent runs.

As it can be seen, NEAT + tournament behavior is clearly superior to the standard method during the second half of the evolution process.

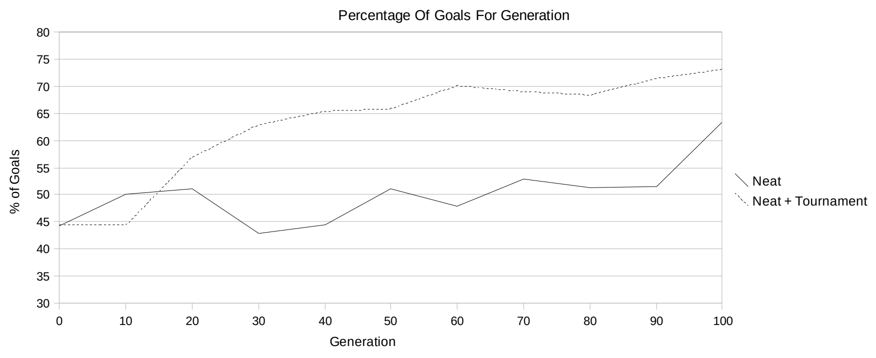

Fig. 7. Average goals for generation.

7. Conclusions and Future Working Lines.

A new strategy that allows improving the behavior of controllers obtained by applying layered evolution, thus considerably reducing computation time without markedly affecting the final controller quality has been presented. Its functioning is based on applying an evolution process in two parts; the first one is commanded by NEAT and the second one by a Binary Tournament applied to all individuals of the population.

Its application in solving a concrete problem has been tried both in the simulated environment and on the real robot with quite satisfactory results.

Different experiences carried out with NEAT have permitted to establish that 20% maximum generations are enough to obtain a population with a basic behavior upon which it is feasible to apply tournaments, thus optimizing execution time.

At present, work is being done on the possibility of installing a mini population of controllers in the robot and that this population evolves along its useful life [17]. To this aim, different genetic operators are being studied [18].

References

1. Whitson, S. Kohl, N. Miikkulainen, R. Stone P. Evolving. Soccer Keepaway Players through Task Decompositions. Machine Learning, 59(1): 5--30, May 2005.
2. Corbalán, L. Lanzarini, L. Evolving Neural Arrays A new mechanism for learning complex action sequences. Special Issue of Best Papers presented at CLEI 2002. Volumen 6. Number 1, December 2003. Montevideo, Uruguay.
3. 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.
4. Corbalán, L. Lanzarini, L. An ENA-Based Strategy Replacing Subobjectives definition in Incremental Learning. Publicado en 25th International Conference on Information Technology Interfaces. ITI 2003. IEEE catalog number 03EX645, pp. 383--390. ISBN 953-96769-6-7. ISSN 1330-1012.
5. Bruce, J. 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.
6. 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.
7. 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.
8. 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. 2005. pp. 211--217.
9. Corbalán, L. Lanzarini, L. De Giusti, A. ALENA. Adaptive-Length Evolving Neural Arrays". Journal of Computer Science and Technology. Vol 5, nro. 4. (http://journal.info.unlp.edu.ar). 2005. . ISSN: 1666-6038. pp. 59--65.
10. Stone, P. Layered Learning in Multiagent Systems. PhD Thesis. CMU-CS-98-187. School of Computer Science. Carnegie Melon University. 1998.
11. Whiteson S., Stone, P. Concurrent Layered Learning. Second International Conference on Autonomous Agents and Multiagent Systems - AAMAS'03 pp 14--18.Julio 2003.
12. Osella Massa, G. Vinuesa, H. Lanzarini, L.Modular Creation of Neuronal Networks for Autonomous Robot Control. Journal Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial. Vol.11. Nro.35. pp.43--53. 2007.
13. Stanley, K.O. Miikkulainen, R. Competitive coevolution through evolutionary complexification. Journal of Artificial Intelligence Research 21. 2004.
14. Stanley, K.O. Miikkulainen, R. Evolving neural networks through augmenting topologies. Evolutionary Computation 10. 2002. pp. 99--127.
15. Radcliffe, N.J. Genetic set recombination and its application to neural network topology optimization. Neural computing and applications 1. 1993. pp. 67--90.
16. Goldberg, D.E. Richardson, J. Genetic algorithms with sharing for multimodal function optimization. 1987. pp. 148--154.
17. Vinuesa, H. Osella Massa, G. Corbalán L. Lanzarini, L. Continuous Evolution of Neural Modules of Autonomous Robot Controllers. Jornadas Chilenas de Computación 2007. Iquique, Chile. Noviembre de 2007
18. Vinuesa, H. Lanzarini, L. Neural Networks Elitist Evolution. Proceedings del 29th International Conference on Information Technology Interfaces. ITI 2007. Dubrovnik, Croacia. 25 a 28 de junio de 2007. ISBN: 978-953-7138-09-7 / ISSN: 1330-1012 (CD Rom). Artículo completo – Publicado por IEEE Computer Society Press – pp. 457--462.