Continuous Evolution of Neural Modules for Autonomous Robot Controllers

Hernán Vinuesa[1], Germán Osella Massa[2], Leonardo Corbalán[3], Laura Lanzarini[4]
{hvinuesa, gosella, corbalan, laural}@lidi.info.unlp.edu.ar

III-LIDI (Instituto de Investigación en Informática LIDI)
Facultad de informática. Universidad Nacional de La Plata.
La Plata. Argentina. 1900

Abstract

In recent years, research on techniques for developing controllers for autonomous robots has been conducted. Evolutionary Algorithms are among the most popular tools used in this type of problem, mostly for its capacity to adapt to the environment. Nevertheless, they are usually applied to produce a controller that will not continue its adjustment after concluding this process. This causes trouble to a controller when it is used in a dynamic environment. In this paper, the combination of a state-of-the-art modular neuro-evolution algorithm with a specific evolutionary algorithm is proposed. The former method is used to generate the controller while the later is used to adjust it during its operation. As a result, an adaptable controller based on a minimal topology neural network is obtained. The method proposed was tested in a goal-reach problem with satisfying results. Finally, conclusions are presented.

Keywords: Modular Evolution, Neural Networks, Evolutionary Algorithms.

Resumen

En los últimos años se han realizado diversas investigaciones en técnicas para el desarrollo de controladores para robots autónomos. Los Algoritmos Evolutivos son una de las herramientas más utilizadas en este tipo de problemas por su capacidad de adaptación al entorno. Sin embargo, en su mayoría, la aplicación se concentra en la fase de generación del controlador no permitiendo realizar adaptaciones posteriormente. Esto perjudica la aplicación del controlador en ambientes dinámicos. Este artículo propone extender la evolución del controlador a lo largo de su vida útil combinando un método basado en evolución de módulos neuronales con un algoritmo evolutivo específico. El primer método es utilizado para producir los controladores mientras que el segundo ajusta al controlador durante su funcionamiento. Como resultado, se obtiene un controlador adaptable en la fase de ejecución basado en una red neuronal de arquitectura mínima. La propuesta de este artículo fue medida en la resolución de problemas del tipo alcance de objetivos con resultados satisfactorios. Finalmente, se exponen las conclusiones.

Palabras Claves: Evolución de Módulos Neuronales, Redes Neuronales, Algoritmos Evolutivos.

[1] Becario III-LIDI. Auxiliar Docente. Facultad de Informática. UNLP.
[2] Becario de Doctorado CONICET. Dirigido por Ing.De Giusti y Lic.Lanzarini. JTP. Fac. Informática. UNLP
[3] Becario Formación Superior UNLP. Dirigido por Ing.De Giusti y Lic.Lanzarini. Auxiliar Docente. Fac. Informática. UNLP
[4] Profesor Titular DE. III-LIDI. Facultad de Informática. UNLP

1. Introduction

Evolutionary Robotics (ER) is a methodology that uses Evolutionary Algorithms (EA) to develop controllers for autonomous robots, usually in the form of Artificial Neural Networks (ANN). Artificial Neural Networks are chosen partly because of their ability to learn and adapt to the environment and partly because they are easily represented in an Evolutionary Algorithm. [11].

Although the development of controllers using this kind of strategies is very useful, it is rarely applied to the adjustment of the controller after it enters in operation. When acting in a dynamic environment, like the real world for example, an autonomous robot must adapt itself to the changing conditions or it will not be able to correctly perform its tasks.

In this paper, the combination of a state-of-the-art modular neuro-evolution algorithm with a specific evolutionary algorithm is proposed. The former method is used to generate controllers while the later is used to adjust them during their operation. As a result, an adaptable controller based on a minimal topology neural network is obtained.

This task is divided into two stages: First, a population of controllers is produced using information from the environment. Then, a small subset of that population is selected and submitted to a second Evolutionary Algorithm inside the robot, allowing it to adjust itself to changes in the environment.

Controllers produced by the first stage consist of a combination of different neural modules that are simpler than the controller they form part of and were evolved previously and independently. This seeks to minimize the time required to obtain an acceptable performance. [5].

If the environment in which the autonomous robot acts does not suffer any changes, the best controller generated by the first stage will be good enough to solve the problem because they have precisely evolved to do it. However, in the real world changes occur all the time: light conditions vary, obstacles are not always in the same place or the target to reach is a different one. Owing to it, the controller must do some adjustments when it is in operation. This is the reason for the second stage: to adapt the autonomous robot interaction with its environment. To do so, a small population formed by the best three controllers found in the first stage is evolved using another Evolutionary Algorithm designed to run inside the robot.

This paper is organized as follows. Section 2 describes the evolutionary algorithm used to produce the initial controllers. Section 3 introduces how the continuous adaptation of the robot to its changing environment is achieved. Section 4 shows the results obtained. Finally, Section 5 presents our conclusions.

2. First Stage Evolutionary Algorithm

In order to solve complex problems, different approaches that divide the original problem into simpler ones have been proposed. 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.

In this direction, methods that combine Incremental Evolution and NeuroEvolution have been developed to offer adaptive mechanisms that minimize the necessary knowledge needed to obtain a working controller, giving rise to Neural Networks composed by several other networks [1]. As the controller is composed by several modules, it is important to define which module should be active at each time-step [13]. There exist different alternatives: from the use of an ad-hoc designed decision tree [4] to mechanisms that embed the selection into the controller [2].

If each of these simpler tasks is successfully and independently solved, it should be possible to combine these solutions to complete the complex task. On this assumption, an extension to the NeuroEvolution of Augmenting Topologies (NEAT) method which incorporates the concept of modules is used.

It is assumed 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 stage is to produce a **Unified Neural Network** constituted by the combination of all of these modules and capable of solving the complex task.

A brief description of the original NEAT method alongside the proposed extension will be presented in the following subsections. For more details, refer to [5].

2.1. Standard NEAT

The standard NEAT implementation has been shown to be a highly effective NE method in several domains [8]. 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 that are connected. In order to perform a 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 allow the 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. In 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 up 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].

2.2. NEAT with Modules

The incorporation of neural modules to the NEAT method implies carrying out several modifications. The first one is related to the neural networks that compose the initial population. In the original proposal, it is assumed that there is not enough knowledge of the problem to specify the topology of those networks. In addition, starting with minimal networks allows the method to explore simpler solutions first. 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 the union of the inputs of each module as input. The modules are connected to those inputs without undergoing any modification. The unified network outputs depend 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 opposite stimuli compatible, each module output neurons become hidden neurons. To each of these converted neurons, a new connection is added that links this neuron to the output neuron that produces the response which was originally yielded by former neuron. The connection is established with a 1.0 weight, 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 with the module that it belongs to. This is done to simplify the tracking of the modules that compose each network once the evolution has started.

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 did not 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 that they can reach the solution of the posed complex task together.

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 that acts as an interface between the modules is simpler than solving the whole problem.

3. Second Stage Evolutionary Algorithm

The second stage of the evolution takes place on the autonomous robot and it should run for an indefinite period as long as the controller is operational. On this stage, a small population consisting of three controllers generated in the first stage is maintained. The selection of the controllers that belong to the initial population may be accomplished in several ways: The three best controllers of the entire population of the first stage can be chosen to integrate the new population. Alternately, the fittest controller of each of the three most performing species may be selected. In this work, the first strategy was adopted. The selected controllers provide the genetic pool that will allow the second stage algorithm to adapt the robot behavior when changes in its environment occur.

The Evolutionary Algorithm running on this stage works by performing small modifications in the controllers, causing variations in its behavior. To do so, each controller in the current population is evaluated and a new controller is produced using the two best controllers found. This new controller replaces the worst one in the current population. This guaranties that the most performing controller will never be lost.

An Extended Linear Recombination operator applied to the best controllers found is used to produce the new controller. This operator was selected based on the results obtained in [10]. Let P1 and P2 be the selected controllers to be used in the recombination and O the new controller obtained by applying the Extended Linear Recombination operator to them. Equation 1 shows how O is generated.

$$O_i = P_{1i} + \frac{s_i \cdot r_i \cdot a \cdot \left(P_{2i} - P_{1i}\right)}{\left\|P_1 - P_2\right\|} \quad (1)$$

Sub-index $i \in [1, N]$ represents each of the network's weights and N, the number of weights. a defines the step size to use, and its value is computed as 2^{-ku}. k determines the precision of such step, taking values between 4 and 20 as suggested in [6]. $a \in [0,1]$ is a uniformly distributed random number, $r_i \in [-0.5,0.5]$ is a uniformly distributed random number and it represents the maximum variation that can appear between parent and offspring in the *i-th* weight. s_i is -1 or $+1$, random uniform. *s* is the direction of the step.

Inside this possible area the offspring are not uniform at random distributed. The probability of creating an offspring near the parents is high. Only with low probability offspring are created far away from the parents. For more details on this operator, see [6].

4. Results and Discussion

This work proposal can be used to produce a controller based on Evolutionary Artificial Neural Networks that is capable of guiding a Khepera II robot to reach a certain target while avoiding obstacles. The controller must also be able to adapt to changes in its environment when it enters in operation.

As only the robot proximity and light sensors were used, the target that the robot must reach is actually a light source located at random places inside a maze. The goal of the robot is to navigate freely without crashing until it approaches a bright zone where it will try to stay.

The initial population of controllers was generated according to the methodology explained in section 2: Two independently produced modules were combined: one of them gives the robot the capacity to avoid obstacles while the other seeks for the nearest light source. Each of these modules contains a Recurrent Artificial Neural Network generated using conventional NEAT as described in

section 2.1. Figure 1 shows the network obtained from merging the two modules. Both modules use the same input data from the proximity sensors. The outputs of each module were combined using two newly added neurons that commands the motors. This Unified Neural Network has 16 input neurons, 8 from each one of the proximity sensors and 8 from each one of the light sensors, and has 2 output neurons, each one connected to each of the motors.

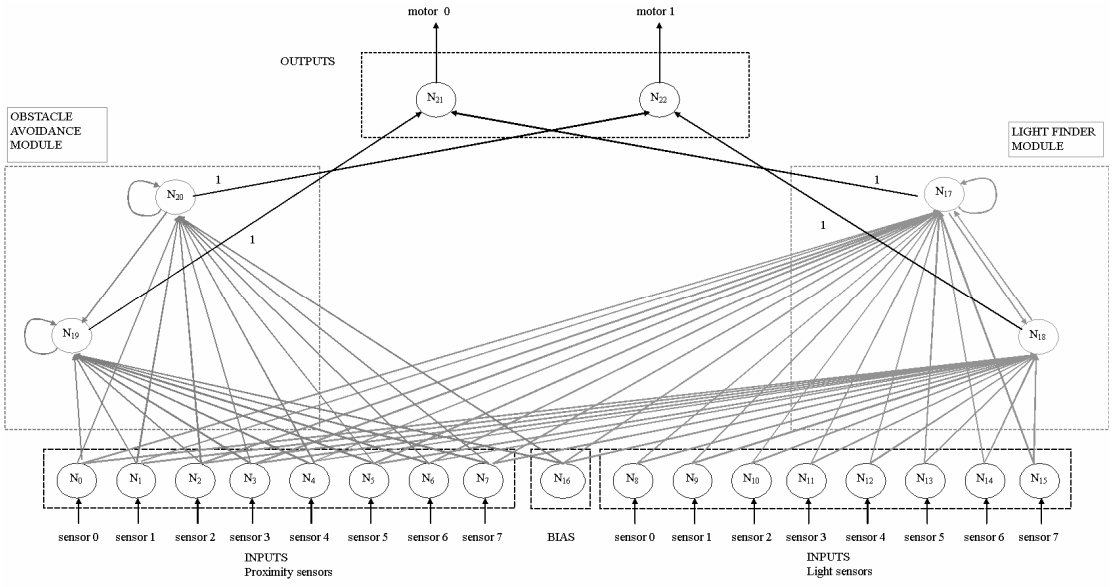

Figure 1

Slightly mutated copies of the original Unified Neural Network genome were used to form the initial population for the first stage. This population was evolved using *NEAT with modules* in the manner explained in section 2.2. In this stage, the fitness of a controller was computed according to its capacity to combine the original modules, as expressed by equation 2.

$$fitness = coef_{obs} \times Fit_{Obs} + coef_{lig} \times Fit_{Lig} \quad (2)$$

In equation 2, Fit_{Obs} and Fit_{Lig} are the fitness of each controller measured according to the fitness functions of the obstacle evasion problem and the light reaching problem, respectively. $coef_{obs}$ and $coef_{lig}$ are constants used to regulate the importance of each behavior in the resultant fitness value. In this work, 1 and 1.3 were respectively used to put more emphasis on the light reaching part.

Figure 2 depicts the fittest Unified Neural Network obtained as a result of the evolutionary process.

The best three performing controllers produced by the first stage were selected to constitute the initial population of the second stage algorithm. These controllers were continuously evolved as long as the robot was operative. A reduced population size was chosen to minimize the computational time required to evaluate the fitness of the population. Each of the three controllers was independently evaluated from the others when controlling the real robot.

After the evaluation of the population was completed, the controllers were sorted according to their fitness. The best two controllers were selected to produce a new one using the Extended Linear Recombination operator. This operator takes two parents and produces an offspring similar to the best parent, but changed in direction of the other parent. The newly produced controller replaced the worst controller of the population.

The frequency in which this process has to be done depends on how much the environment changes. If there are no changes, it may even not be necessary to do it at all. Nevertheless, it is advisable to run the algorithm in the first generations to allow the robot to adapt to the differences in the simulator compared to the real world.

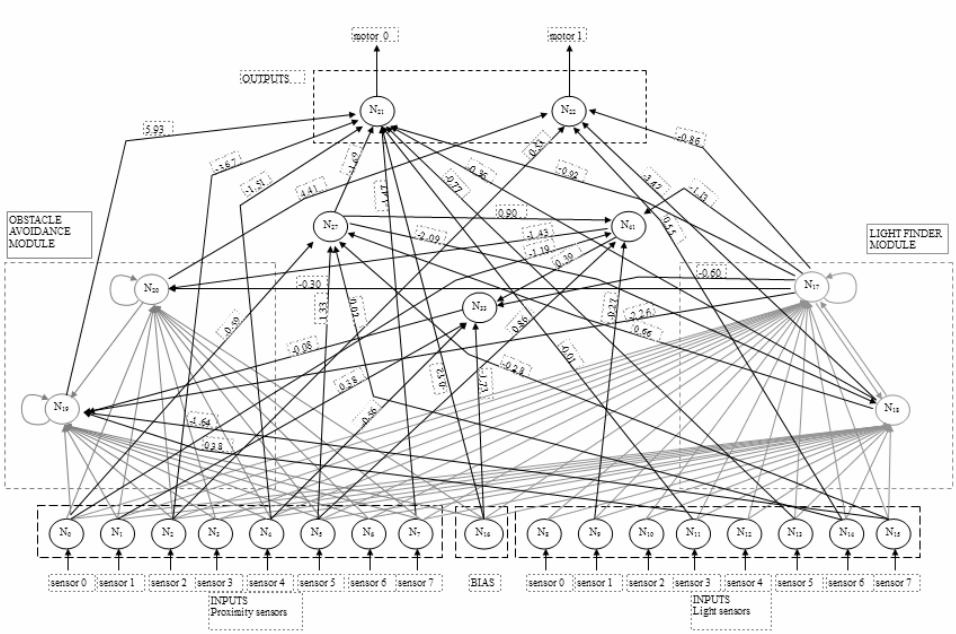

Figure 2

Tests to analyze the ability of the robot to adapt to changes in the environmental conditions were conducted. Figure 3 shows the mean fitness of two different robots generation by generation: one using the second stage algorithm (solid line) and the other not using it (dashed line). In the 60th generation, a sudden change in the environmental conditions was introduced and maintained until the 225th generation.

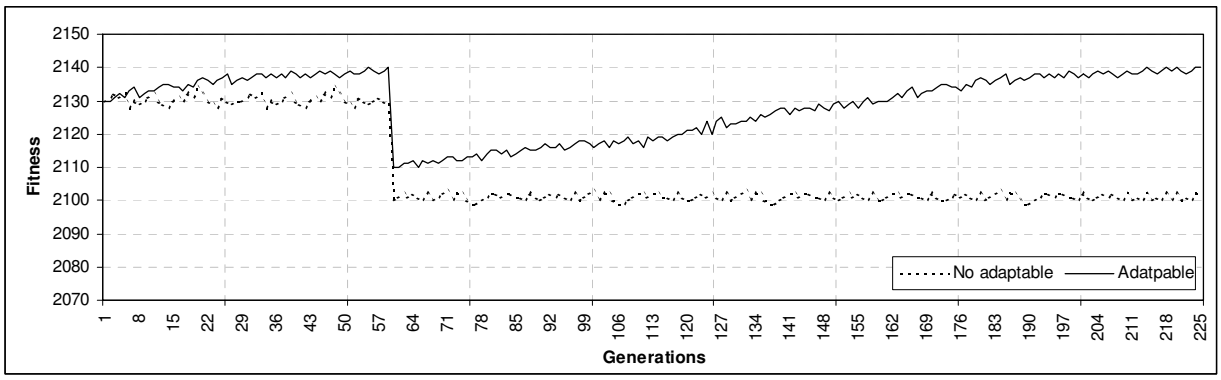

Figure 3: Mean fitness of the adaptable (solid line) and non-adaptable (dashed line) robots.

In the first 60 generations, an improvement in the fitness of the adaptable controller can be noticed compared to the non-adaptable controller. Fitness variations on both lines are due to different conditions in which the controllers were evaluated. This explains why the non-adaptable controller varies its fitness even if the controller does not change.

After the environment changed in generation 60, a drop in the fitness value of both controllers can be seen. This is because none of the controllers were appropriated for the new environmental conditions. However, as the second stage evolutionary algorithm of the adaptable robot continued to adjust its controllers, an improvement in its fitness value can be noticed. After generation 200, its fitness is almost as good as it was in the previous environment. On the other hand, the non-adaptable robot did not improve at all.

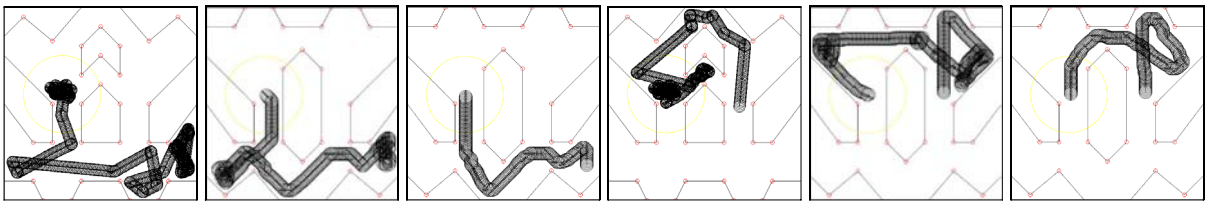

Figure 4: Performance of two adaptable robots in different periods: a) and d) in the original environment, b) and e) immediately after the changes, and c) and f) when the adjustment occurred.

5. Conclusions

Evolutionary Algorithms, though capable of providing excellent results in several areas, have one unfavorable characteristic: the adaptation process may be slow and costly in time for some problems. Because of that, it is rarely employed after an acceptable solution is found.

This paper applies a strategy that reduces the required time to find a suitable controller, taking advantage of the combination of simple neural modules, based on recurrent neural networks generated independently of the problem to solve. Furthermore, a second strategy allows the autonomous robot to continue its adaptation to changes in the environment after entering in operation.

The continuous evolution of a controller on a Khepera II robot capable of reaching a certain target while avoiding obstacles has proved to be successfully in adapting to changes in its environment. Adaptation is a crucial task for an autonomous robot and it may be the difference between a successful controller and a failed one.

References

[1] 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), pp. 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. Bs.As. Argentina. ISBN 987-9495-58-6. October 2004.

[3] Goldberg, D.E., Richardson, J. Genetic algorithms with sharing for multimodal function optimization. pp. 148-154. 1987.

[4] Olivera J. Lanzarini, L. Cyclic Evolution. A new strategy for improving controllers obtained by layered evolution. VI Workshop de Agentes y Sistemas Inteligentes 2005. Concordia, Entre Ríos. Argentina. ISBN: 950-698-166-3. October 2005.

[5] Osella Massa G., Vinuesa H., Lanzarini L. Modular Creation of Neuronal Networks for Autonomous Robot Control. 3rd IEEE Latin American Robotics Symposium. LARS 2006. Chile. October 2006.

[6] Pohlheim H. Evolutionary Algorithms: Overview, Methods and Operators. GEATbx version 3.7. Genetic and Evolutionary Algorithm Toolbox for use with Matlab. pp: 28-29. November 2005.

[7] Radcliffe, N.J.: Genetic set recombination and its application to neural network topology optimization. Neural computing and applications 1. pp. 67-90. 1993.

[8] Stanley, K.O., Miikkulainen, R. Competitive coevolution through evolutionary complexification. Journal of Artificial Intelligence Research 21. 2001.

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

[10] Vinuesa, H. Lanzarini, L. Neural Networks Elitist Evolution. 29th Internacional Conference Information Technology Interfaces (ITI 2007). Dubrovnik. Croatia. 2007.

[11] Walker J. Garrett, S. Wilson, M. The Balance Between Initial Training and Lifelong Adaptation in Evolving Robot Controllers. IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS—PART B: CYBERNETICS, VOL. 36, NO. 2. APRIL 2006.

[12] Yao, X. 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, pp.1423-1447. September 1999.