Cyclic Evolution
A new strategy for improving controllers obtained by layered evolution

A.C. Javier Olivera, Lic. Laura Lanzarini

III-LIDI (Institute of Research in Computer Sciences LIDI)
Facultad de Informática. Universidad Nacional de La Plata.
La Plata, 1900, Argentina.

ABSTRACT

Complex control tasks may be solved by dividing them into a more specific and more easily handled subtasks hierarchy. Several authors have demonstrated that the incremental layered evolution paradigm allows obtaining controllers capable of solving this type of tasks.

In this direction, different solutions combining Incremental Evolution with Evolving Neural Networks have been developed in order to provide an adaptive mechanism minimizing the previous knowledge necessary to obtain a good performance giving place to controllers made up of several networks.

This paper is focused on the presentation of a new mechanism, called Cyclic Evolution, which allows improving controllers based on neural networks obtained through layered evolution. Its performance is based on continuing the cyclic improvement of each of the networks making up the controller within the whole domain of the problem.

The proposed method of this paper has been used to solve the Keepaway game with successful results compared to other solutions recently proposed.

Finally, some conclusions are included together with some future lines of work.

Key words: Evolving Neural Networks, Incremental Evolution, Layered evolution.

1. INTRODUCTION

Several researches have demonstrated that certain tasks may be solved using layered evolution.

By complex task we understand that whose solution is not direct but involves the learning of a strategy in order to achieve the expected objective. Problems such as prey capture and target reaching belong to this category [3].

In addition, there exist situations which cannot be solved by a single agent. Such is the case of prey capture, in which the predator is slower than the prey, or the robot football. In both cases, beyond the differences among the agents, the team is the one which should carry out the strategy [5].

When the situation to solve is complex, it is hard to establish a priori the controller to be used, and here is where layered evolution becomes important. This process consists in dividing the original problem into simpler parts, called subtasks, thus allowing a gradual learning of the expected response.

On the other hand, unless we count with the initial information necessary to solve each subtask, it is ideal to count with some mechanism allowing carrying out the adaptation as automatically as possible. In this direction, different solutions combining techniques of Incremental Evolution with Evolving Neural Networks have been developed with the aim of providing an adaptive mechanism minimizing the previous knowledge necessary to obtain an acceptable performance giving place to controllers made up of several networks [1]. Another aspect to take into account is the way of determining which neural network should be run at each time instant [9][10]; in this direction, there exist several alternatives ranging from the use of an ad-hoc design decision tree [4] to mechanisms automatically organizing the structure [2].

2. OBJECTIVE

This research is based on the works previously carried out in the fields of layered evolution [6][8] through neuroevolving algorithms and proposes an alternative which allows obtaining improvements in the proposed solutions.

The objective of this paper is to present a new adaptation strategy, called Cyclic Evolution, through which the collective behavior of the controllers obtained by traditional layered evolving methods can be improved.

In particular, the results of the adaptation of this method for solving the KeepAway game will be shown.

Section 3 describes the KeepAway game together with the way of obtaining an initial controller capable of solving it. Section 4 presents the algorithm used to implement the cyclic learning. Section 5 details some of the implementation aspects. Section 6 describes the results obtained and Section 7 presents the conclusions as well as some future lines of work.

3. KEEPAWAY

Keepaway is a subtask of Robot Soccer game in which one team of agents, the keepers, attempts to maintain possession of the ball while the other, the takers, tries to get it. The game is carried out in a fixed, circular region and ends when the ball exits the bounding circle or when the taker grabs it [7].

There exist machine learning applications that successfully solve different subtasks of the Robot Soccer; however, the solution of the complete game seems to be beyond the capacities of modern techniques. As a consequence of the diversity of applications, which solve different parts of the Robot Soccer, the task of coherently comparing the different machine learning applied techniques becomes quite difficult. [7] has recently proposed the Keepaway game as a proper domain for contrasting different machine learning approaches to the resolution of the Robot Soccer.

Keepaway is a challenging machine learning task for several reasons:

- The state space is far too large to explore exhaustively;
- Each agent has only partial state information;
- The action space is continuous;
- Multiple teammates need to learn simultaneously;
- The keepers are relatively large when compared to the playing area, which makes moving and positioning difficult around the ball;
- The ball does not move much faster than the players, which prevents the keepers from being able to quickly make passes around the taker;
- The keepers do not possess any abilities for handling the ball. They are modeled as simple cylinders, and lack any way to "grab" the ball and move with it. If they run into the ball, the ball will bounce away.

For these reasons, the KeepAway game requires complex behavior, ranging from the input data processing about each keeper, the teammate – the taker – and the ball, to the decision-making as to the best course of action decided upon at each moment of the game, and the acquisition of the ability needed to carry it out.

In our implementation of KeepAway, each robot receives noise-free sensory input describing the current state of the game. All these inputs are scaled to [-1, 1] and presented to each player in relative coordinates.

Each robot is round, like the ball. A really simple physics engine is used to allow the ball to bounce in the players once it makes contact with them. As a result, the only way in which the players can "kick" the ball is to approach it in the precise direction and at the proper speed so that the ball bounces or is thrown in the right direction.

Figure 1 shows the region in which the game is carried out. Here are three keepers, one of them has the ball and the taker is placed in the center of the field.

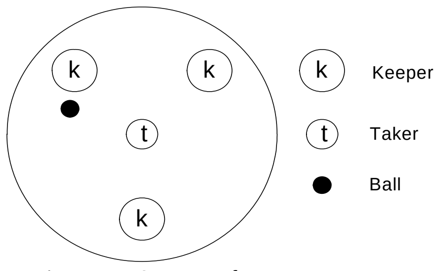

Figure 1. A game of Keepaway

3.1. Decomposition of the problem in simpler subtasks

This game can be decomposed in four subtasks, each of which will be commanded by a feedforward neural network obtained by evolution and made up of three layers: input, hidden, and output. In all the cases, the hidden layer consists of two neurons:

- **Intercept**: The goal of this network is to get the agent to the ball as quickly as possible. The network has four inputs: two for the ball's current position and two for the ball's current velocity. Two output neurons are used, which control the keeper's heading and speed.
- **Pass:** The pass network is designed to kick the ball away from the agent at a specified angle. The difficulty of the learning lies in that the angle with which the player kicks the ball depends on its relative position to it. Hence, in order to learn the proper behavior, the keeper should approach the ball in the proper way. The network has three inputs: two for the ball's current position and one for the target angle. It makes use of two output neurons with the keeper's direction and speed.
- **Pass Evaluate:** This network allows the keeper deciding to which teammate to pass. It has six inputs: two for the position of the ball, two for the taker's position, and two for the teammate whose potential as a receiver it is evaluating. It has an only output neuron which allows obtaining a real value between 0 and 1, indicating its confidence that a pass to the given teammate would succeed.
- **Get open**: The objective of this network is that the agent should get to a good position where it can receive a pass. It has three inputs: two for the ball's current position, two for the taker's current position, and one indicating how close the agent is to the field's bounding circle. It has two output neurons which control the agent's heading and speed.

3.2. 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 KeepAway game.

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.

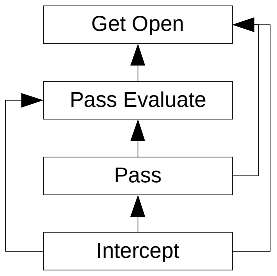

Figure 2. A layered learning hierarchy for the keepaway task. Each box represents a layer and the arrows indicate dependencies between layers.

This is called layered learning based on the dependence existing in the order of the 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 are placed, and so on.

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

3.3. Resolution of KeepAway

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 controller for a KeepAway player is obtained based on specific controllers for each subtask. Figure 3 shows the decision tree used by the keepers to solve the KeepAway game.

The behaviors specified in the tree-leaves are carried out by the neural networks proposed by this paper, as well as the node “Teammate #1 Safer?”. The remaining nodes have fixed behavior.

In each turn, keepers make use of the decision tree to select the proper subcontroller for that moment. If a keeper is less far from a certain length of the ball, it tests which is the teammate that is more likely to successfully receive the pass, and attempts to kick in such direction. If it is beyond a certain distance, the keeper tries to get the ball if it is directed to it, or otherwise it tries to get open to a proper area for future receptions.

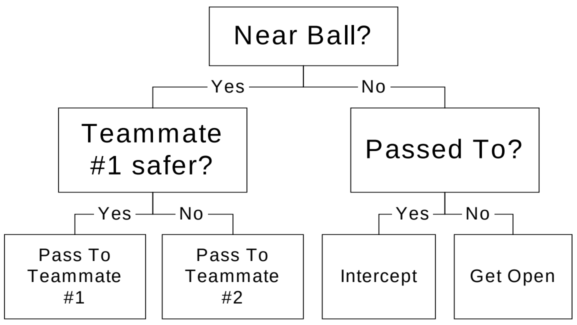

Figure 3. A decision tree for controlling keepers in the keepaway task.

The behaviors specified in the tree-leaves are carried out by the neural networks proposed by this paper, as well as the node “Teammate #1 Safer?”. The remaining nodes have fixed behavior.

In each turn, keepers make use of the decision tree to select the proper subcontroller for that moment. If a keeper is less far from a certain length of the ball, it tests which is the teammate that is more likely to successfully receive the pass, and attempts to kick in such direction. If it is beyond a certain distance, the keeper tries to get the ball if it is directed to it, or otherwise it tries to get open to a proper area for future receptions.

It is worth to mention that the behavior of every node could be controlled by neural networks, instead of having a fixed, manually programmed behavior. The decision tree could also be replaced by a neural network capable of carrying out the same function, i.e. selecting the most adequate subcontroller at each instant. These options were not taken into account for time reasons, since the additional, required trainings for each new network would take much more time.

Next, a new learning method – called cyclic evolution – is presented, whose implementation makes great use of the concepts previously mentioned and grounds this paper. The method has been created in order to improve the behavior of controllers obtained by other learning methods, layered learning, in particular.

4. CYCLIC EVOLUTION

This training mechanism proposes a strategy to improve the performance of neural networks controlling each subtask. These networks are initially obtained conventionally using layer evolution, each allowing the resolution of a part of the problem [4]. For this, they are trained in the resolution of simpler and more specific tasks. This allows them to be part of the resolution of other similar problems by just modifying the way they interact.

The method is cyclic since it establishes an order in the adaptation of each subtask, allowing –after the last one is trained – the evolution of the first one to start once again, thus generating a continuing cycle. During subtask learning, the networks controlling the rest of the activities remain still. This allows each network to improve its behavior and integration with a set of already trained networks.

Let T = $\{t_1,t_2,t_3,...,t_n\}$ be a set of subtasks, C = $\{c_1,c_2,c_3,...,c_n\}$ be a set of subcontrollers solving each subtask ti, and E() be a function representing a c_i subcontroller training; then, figure 4 graphically represents the proposed strategy in this paper.

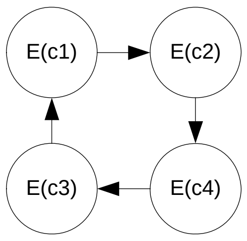

Figure 4. A brief graphic representation of cyclic evolution. Each oval represents the training of a subcontroller and the arrows making up the circle represent the order in which the trainings are carried out.

It is important to highlight that all the trainings are carried out in the final domain, not existing an environment specially prepared for each task, as it is usually the case in the conventional methods. This allows the networks to keep their training within the domain in which they are finally solved.

Cyclic-Evolution Method Algorithm

```
Begin { main program}
   Let C={c1,c2,c3,..,cn} be the initial
   subcontrollers set.
   Let O be the final objective.
   Let Z be the maximum cycle quantity to carry
   out.
   Let G be the maximum number of generations
   per cycle.
   Cycles = 0  {until now no cycle has been
   carried out }
   While (objective O is not accomplished)
            and (Cycles < Z)
      For each subcontroller ci, with i of 1 a n.
                  Evolve(C, i)
      End For
      Cycles = Cycles + 1;
   End While
End {main program}
```

The *Evolve* Process is in charge of improving the performance of the i-th subcontroller. It has two parameters: **C** representing the controller made up of a set of neural networks, and **i** indicating the number of subcontroller to be evolved. This process returns the modified **C** controller since the i-th subcontroller has been replaced by an improved version.

It is worth mentioning that in order to implement this process, any type of evolving algorithm can be used. In particular, this paper has made use of ESP (Enforced Subpopulations). For a clear description of this method, see [3].

Next, details of the algorithm used in this paper to implement this process are presented.

```
Process Evolve(C,i)
{Evolves the i-th C subcontroller using ESP}
Begin
  Repeat
      Repeat
         ▪ Build ci' selecting at random a
           c/subpopulation hidden neuron.
         ▪ Evaluate the fitness of the controller
           composed by
         ▪ {c1, c2, ...,ci', ...,cn} in the domain of
           the complete problem.
         ▪ Accumulate the fitness obtained in the
           ci hidden neurons
      Until (each neuron of each subpopulation
               has taken part in a 10 KeepAway test
               average)
      Obtain the following neurochromosomes
      generation for each population through
      genetic operators.
  Until (reaching a number of generations) or
        (until obtaining a ci' optimum)
  {replace ci by the best ci' found up to the
   moment }
  C = {c1, c2, ..., ci',…, cn}
End { evolve process }
```

5. IMPLEMENTATION ASPECTS

In order to obtain each of the neural networks controlling the subtasks mentioned in section 2, the following considerations have been taken into account:

- In all the cases, feedforward networks with a single hidden layer made up of two neurons have been used. The decision of using two neurons in the hidden layer is made by other authors [4]. They have tested other configurations, though the best results were obtained with just two neurons.
- The training algorithm used in all the cases is ESP [3] with two subpopulations of 100 individuals each, where the used stagnation factor for delta coding is 20.
- For each of the networks, the training was carried out in 100 generations.

For a detailed description of how the initial neural networks controlling each subtask were obtained, see [4].

Each of these networks builds a controller according to the tree shown in Figure 3, which will be replicated in each keeper. For its application, we have considered that:

a) The ball is close to the keeper if it is located not farther than a D distance equivalent to the sum of 3 player's diameters.

b) The decision whether a keeper is receiver or not depends on the result of the pass network of the player keeping the ball. At the moment in which a keeper decides to pass the ball to a teammate, this last one "knows" he is the receiver.

As regards the taker's behavior, even though it is possible to use the neural network controlling the Intercept subtask, it was awarded with the capacity of getting to the ball at a determined speed. This allows effectively measuring the improvement in the keepers' behavior.

Independently of the method used to improve this initial method, all the measurements carried out have taken into account the following aspects:

- They begin with the taker moving at some percentage of the keepers' speed. As the evolution goes on, each time keepers complete 20 passes in an average of 3 KeepAway games, the taker's speed is increased in 5%. This increase makes the controller, replicated in each keeper, adapt itself in order to overcome this difficulty.
- The fitness of each controller made up by the neural network set is computed as the average number of completed passes during 3 runs of the KeepAway game.

6. RESULTS

In order to determine the efficiency and efficacy of the Cyclic Evolution method, the following comparisons have been carried out during N generations:

a) Layered Evolution

This method has been used in [4] and allows obtain the four initial networks trained independently in specific environments according to the dependency order indicated in figure 2. Only the last subtask is trained in the complete domain of the problem and evolved during the successive generations while the three first networks remain still.

b) Concurrent layered evolution of the different networks making up the controller

This is the method proposed by [4][8]. Here, all the networks composing the controller are allowed to evolve simultaneously. In order to keep the uniformity of the measuring process, ESP has been employed in the evolution of each network.

The sole difference found in the paper presented in [4] lies in that the taker is not directly controlled by the intercept neural network, but has the capacity of getting directly to the ball. This is justified in several observations in which the keepers' controller receives a high fitness, not due to its good performance but for the lack of the taker's controller training. This is what causes the difference in the results obtained in this paper and those presented in [4]

c) Cyclic Evolution with a fixed quantity of cycles carried out in N generations

The Cyclic Evolution method has been measured for different quantities of cycles. Given a number K of cycles to be applied in the N generations, each of the four neural networks controlling a subtask has been trained during N/(4*K) generations respecting the dependency order indicated in figure 2.

d) Cyclic Evolution with a variable quantity of cycles carried out in N generations

In this method, each network training within a cycle is carried out in a non-predefined quantity of generations. The stage from the training of a network to the next one is given by a stagnation factor EF indicating that a network ends its training if after EF generations it does not improve its fitness.

Figure 5 shows the average of the results obtained for 100 generations for methods a), b), c) y d) during 10 runs of the KeepAway game. In the case of method c) N=100 and K=1.3 and 5. were used; and for method d) EF=3. It is important to mention that 100 generations for each method have been used for the coherence of the comparisons. In the case of cyclic evolution, apart from the fact that cycles increase, generations remain constant; all of which means that there exist less generations per cycle as the cycles increase.

This allows clearly showing that when using the same quantity of generations, differences come to light as the cycles are varied.

As can be observed in figure 5, the layered evolution method, described in a) and used for obtaining the initial controller of the remaining methods, is not capable of improving itself from generation 40. Even though this result is influenced by the behavior of the first three networks, it does not count with the capacity of the remaining methods to properly evolve in the solution of the general problem.

From figure 5 we can also see that the Cyclic Evolution method provides better results than the Concurrent Layered Evolution method. Notice that this relation is independent of the quantity of used cycles. Moreover, the efficiency of the controller improves as the quantity of cycles increases. Method d) further proves that the quantity of cycles improves the efficiency of the controller because, in this scheme, a greater quantity of cycles than in the case of method c) - for the same quantity of generations - is generally achieved.

Tests carried out with 150 and 200 generations show that this relation is kept. Only from the 250 generations, the differences between the Cyclic Evolution and the Concurrent Layered Evolution start to be considerably reduced.

7. CONCLUSIONS AND FUTURE WORK

A new strategy, called Cyclic Evolution, has been presented. It allows improving the behavior of the controllers obtained through Layered Evolution with really successful results in the resolution of the KeepAway game.

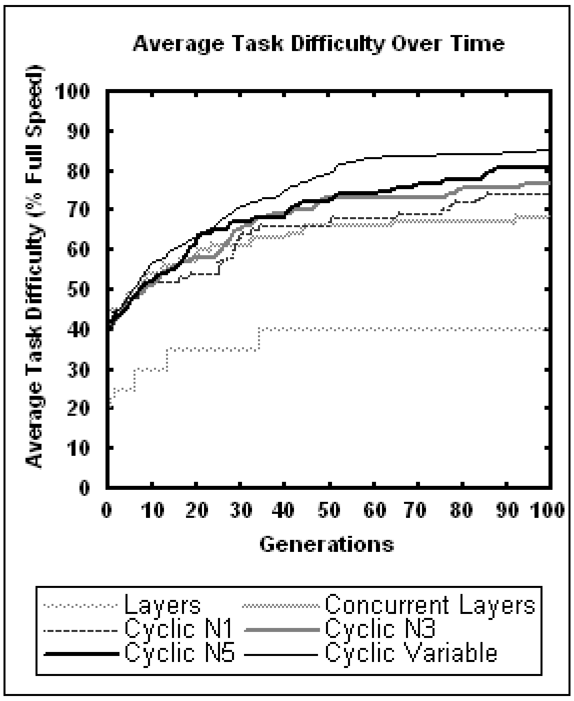

Figure 5. The graphic shows the improvements obtained in the behavior of the taker's controllers through the different evolving methods. As generations advances, these improvements allow overcoming certain levels of difficulty.

As it can be drawn from the previously mentioned differences, the improvements in the controller efficiency introduced by the Cyclic Evolution – even though they vary with the size and quantity of cycles used- outperform those provided by Concurrent Layered Evolution allowing obtaining good controllers in fewer generations.

Even though the cyclic evolution method bounds the search space with respect to the concurrent layered evolution method, it allows us to obtain local improvements in the controller with less generations. The concurrent layered method has the potential capacity of finding good controllers; however, due to the fact that the search space exponentially increases, the task of finding an optimum controller is more expensive in terms of computing time.

At present, works are being developed on the definition of a mechanism allowing identifying the efficiency degree of each network in the solution of a subtask. This would allow emphasizing the training of the most inefficient networks, reducing the running time of the cyclic training.

At a further stage, we expect to apply the results of this research into more complex domains, such as the robot football game, an environment into which a great part of what has been learnt as regards the obtaining of KeepAway controllers could be applied.

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., 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. Oct/04. ISBN 987-9495-58-6.

[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] S. Whitson, N. Kohl, R. Miikkulainen, P. Stone. Evolving. Soccer Keepaway Players through Task Decompositions. Machine Learning, 59(1): 5-30, May 2005.

[5] Stone P., Veloso M. Multiagent Systems: A survey from a Machine Learning Perspective. Autonomous Robots. Vol.8, nro. 3, pp. 345-383. 2000.

[6] Stone, P. Layered Learning in Multiagent Systems. PhD Thesis. CMU-CS-98-187. School of Computer Science. Carnegie Melon University. 1998

[7] Stone, P. and R. S. Sutton: 2002, 'KeepAway Soccer: a Machine Learning Tesbed'. In: A. Birk, S. Coradeschi, and S. Tadokoro (eds.): RoboCup-2001: Robot Soccer World Cup V. Berlin: Springer Verlag, pp. 214-223.

[8] Whiteson S., Stone P. Concurrent Layered Learning. Second International Conference on Autonomous Agents and Multiagent Systems - AAMAS'03 pp 14-18.Julio 2003.

[9] 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.

[10] 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