The Path to Complexity: Incremental Generalization by Evolution in Neural Control Systems

Juan J. Figueredo

Departamento de Ingeniería de Sistemas e Industrial, Universidad Nacional de Colombia.

jjfigueredou@unal.edu.co

Abstract

Evolutionary methods show a persistent limitation achieving global and sustainable optimization. In the artificial life approach to evolutionary robotics, it is expected that control and planning patterns and techniques appear as an emergent phenomena, by means of evolutionary computation applied to some biologically-inspired systems. This paper shows the application of co-evolution and other adaptation techniques of the evolutionary landscape to motion control problems, and their advantages against more classical evolutionary computation methods. First, some preliminaries are presented. Next, a predator-prey control problem is explained. Next, a set of classical evolutionary methods are applied to it. Next, a set of evolutionary landscape adaptation methods are applied, with incremental complexity. Finally, the advantages and disadvantages of each one are remarked and some implications are shown.

Categories and Subject Descriptors I.2.9 [*Biorobotics*]: Evolutionary Robotics

General Terms

Keywords Computational Intelligence, Motion Control, Evolutionary Algorithms, Co-evolution

1. Introduction

Artificial life aims to find those emergent phenomena that give complex attributes to the existent living beings. In this work, efforts are directed towards the study of emergence of motion control capabilities and dynamical planning behaviors of agents. Following the artificial life approach, those capabilities are expected to evolve from a very simple or non-functional initialization. Nonetheless, given a set of parameters for the evolution (fitness function, transformation operators, population size) and control architecture, evolution progress stalls after a given number of iterations in a local minimum. When the control problem is simple enough, the local minimum found might be the global minimum; but in complex instances, the solution found is suboptimal. This limitation is not exclusive of the application of evolution to control tasks, but is a general limitation of evolutionary methods.

On the other hand, life found its route among the multiple local minima, and has shown the capability to progressively evolve behaviors from simpler ones, or at least to not stall in one specific behavior, attaining the goal of open-ended evolution. Some notable differences between evolution schemas used in traditional evolutionary computation and those present in biological evolution are, first, that biological evolution does not pursue explicitly complex behaviors; second, that biological evolution has a non-static fitness function, i.e. best fit individuals in one evaluation may not be the fittest in the next evaluation; third, several population groups have coupled fitness functions, and even the environment is coupled to the evolution of those groups; fourth, individuals from one generation may be present in several iterations or evaluations of their fitness functions; five, population size varies with the performance of the group, the performance of coupled populations, and environment properties; and seven, there are presents phenomena of reproductive isolation, where several populations are not allowed to breed, giving rise to partial or total speciation, and sometimes to hibrid speciation. That way, biological evolution has means to explore broader regions of the evolutionary landscape than its homologous computational methods.

Next, some preliminaries on computational intelligence are presented, mainly in the areas of evolutionary computation, which is the main focus of the article, and in the area of neural controllers. In the remaining sections the control problem used as test bed is presented, an application of traditional evolutionary computation methods is shown, a co-evolution scheme is applied, other incremental evolution methods are used to obtain greater complexity, and finally, a discussion is made.

2. Preliminaries on Computational Intelligence

2.1 Evolutionary algorithms

Evolutionary algorithms are a set of population-based heuristic search and optimization techniques. They maintain a population, and apply a set of operators or transformations over its members. Those operators are typically inspired on biological evolution and usually include selection, reproduction and mutation, among others. The operators are dependent of the evaluation of a performance function called fitness function. Generally, fitness function evaluation may include, from a simple numerical evaluation, to a complex simulation, in order to get the performance criterion which its optimization is pursued.

The pseudo-code of a general evolutionary algorithm is as follows:

The most predominant form of an evolutionary algorithm is embodied by genetic algorithms. They most frequent genotypical representation is a bit sequence, although other representations can be used. Usually they are implemented with a generational replacement of population, but in some situations it is useful to conserve a small set of the better individuals across generations in a steady-steady replacement.

Algorithm 1 $EvolutionaryAlgorithm$

1: $P \leftarrow$ Generate initial population of size N
2: Evaluate fitness for each individual in P
3: **repeat**
4: $P' \leftarrow$ Apply operators to P
5: Evaluate fitness for each individual in P'
6: $P \leftarrow$ Select N individuals in P' according to a selection scheme
7: **until** Termination condition is met

2.2 Neural networks

Artificial neural networks are a connectionist computing scheme inspired by brain neural layout at cellular level. It is based on simple computing structures called neurons which have several inputs and a single output. The neuron performs a weighted sum over its inputs to determine its neuron potential v_i:

$$v_i = \sum_j w_{ji} x_j \tag{1}$$

where w_j is the connection weight from input j to this neuron (i) and x_j is the output signal of neuron j. Then, it applies a activation function to its potential to generate its output:

$$y_i = f(v_i) \tag{2}$$

The activation function can be the signum function (as in the MacCulloch and Pitts model [**?**]), unipolar or bipolar sigmoidal (as in Perceptrons [**?**]), or gaussian (as in Radial Basis Networks [**?**]), among others.

Three types of inputs can be considered. The most basic are the external inputs. Another type of input is the bias input (each neuron has only one) which serves as activation threshold and typically has an unitary associated weight. The last type of inputs is found in neuron connections, in which the output of a neuron serves as input of another. This way, neural networks can be arranged in layers with neurons connecting in its input only with outputs of neurons in previous layers, i. e. feedforward, or can form closed loops i. e. recurrent. Some particular specifications to the way in which neurons are connected raise several neural network topologies.

The neurons can compute continuously or can compute in discrete time. Also its potential can be specified not to be an algebraic equation of its inputs but a differential equation, in such a form that it has inner dynamics, as in neural oscillators.

Another major classification of neural networks is the way in which they are trained, that is, the connection weights necessary to perform the desired computation. If they are learned by a training set of inputs and desired outputs it is said to be supervised learning, and if they are learned without a set of desired outputs but by detecting statistical patterns in the input data it is said to be unsupervised. A third type of learning appear when the network does not have a training set but the environments gives a penalty or reward to each output, as in reinforcement learning (it also can be thought as unsupervised learning).

The particular topology of interest here are recurrent neural networks. The model, as is presented in [**?**] is:

$$\begin{aligned} x_{rnn}(n+1) &= \psi(W_a x_{rnn}(n) + W_b u_{rnn}(n)) \\ y_{rnn}(n) &= C x_{rnn}(n) \end{aligned} \tag{3}$$

Here, x_{rnn} is the 1-by-q system state vector at n, ψ is a diagonal function with domain and co-domain R^q corresponding to activation function, u_{rnn} is the 1-by-m input vector at n, the q-by-q matrix W_a represents the connection weights between neurons, and the q-by-m+1 matrix represents the connection weights between input nodes and neurons, including a bias term. Also, y_{rnn} is the neuranl output vector of the neural net, and C is a p-by-q matrix of linear combination from the neurons to the outputs.

3. The Predator-Prey Control Problem

The problem which will be used as test-bed for the different strategies of evolution is a predator-prey system of simulated mobile robots. Each robot has its own dynamic which has the form:

$$\frac{d}{dt} x = f(x) \tag{4}$$

That is, a dynamic state equation. On it, there are accounted the linear and rotational inertia of the robot mass, and two traction forces given by two simulated wheel. Also, the lateral friction of the wheels is considered.