# Evolution of Neurocontrollers in Changing Environments

Javier Apolloni, Carlos Kavka and Patricia Roggero

LIDIC
Fac. de Cs. Físico Matemáticas y Naturales
Universidad Nacional de San Luis
Ejército de los Andes 950 - D5700HHW
San Luis - Argentina
email: {javierma,proggero,ckavka}@unsl.edu.ar

**Abstract**

One of the most challenging aspects of the control theory is the design and implementation of controllers that can deal with changing environments, i. e., non stationary systems. Quite good progress has been made on this area by using different kind of models: neural networks, fuzzy systems, evolutionary algorithms, etc. Our approach consists in the use of a memory based evolutionary algorithm, specially designed in such a way that can be used to evolve neurocontrollers to be applied in changing environments. In this paper, we describe our architecture, and present an example of its application to a typical control problem.

**Keywords:** evolutionary algorithms, neural networks, control

## 1 Introduction

The well known problem of system identification consists in the determination of a model that can be used to map a set of input output patterns, defining the behavior of a physical system (target). Once determined, the model can be used as a ”black box” to control the system. There exists a lot of methods that can be used for system identification, but most of them are only applicable to linear systems, while most systems in natural life are non linear.

As an alternative to traditional system identification, a number of new techniques have been used quite successfully [2] [6] [5]: artificial neural networks (ANN), evolutionary algorithms (EA), fuzzy logic, hybrid systems, etc. However, in most cases, they are applied to static systems that do not change in time. This is, in some cases, an unrealistic assumption.

The EUNITE (EUropean Network on Intelligent TEchnologies for Smart Adaptive Systems) defines three levels of adaptation requirements for an adaptive controller [8]:

- adaptation when changes are due to drifting over time, space, etc.

- adaptation to similar settings without the controller being specifically ported to it.
- adaptation to new/unknown applications.

Our algorithm is concerned with the first type of adaptation. It is the case of a non stationary system, where the physical properties change in time. For example, a mechanical system that increases its frictions over time, a temperature based system that drifts over time, etc.

The next subsection presents a short description of some successful current methods applied to different kinds of changing environments. Section 2 presents a description of the usual approach used to evolve controllers. The details of our model are presented in section 3, and an example of its application to a typical control problem are presented in section 4.

## 1.1 State of the art

As it was mentioned before, there exists a number of methods to get controllers that can be used for changing environments [8] [13] [6] [3].

ANN can be trained with a set of input output patterns and in this way, they can be used to predict the correct control signal to be sent to the target system. If a training data set is not available, a teacher signal (or reinforcement) that is available ocasionally, can be used for learning [1] [7]. Usually after learning, the ANN is "frozen" and put to work. Even if the ANN has a generalization ability, it cannot adapt itself to system drifting. Retraining can be performed on the ANN in this case. However, the ANN suffers from *catastrophical interference*, a problem that happens when the ANN cannot adapt the parameters and/or the topology to reflect the changing environment [9]. This problem arises since the ANN needs both a short term memory for on-line learning, and a long term memory to recognize context drifting.

EA have been used to evolve a population of controllers. At the end of the evolutionary process, the best controller is selected, "frozen" and put to work. Again the controller is fixed, and cannot cope with changing environments. The usual approach to solve the problem is to restart the evolutionary process, in order to get an adapted controller. Restarting from a random population is an approach that can be useful in the third adapting situation (see section 1), but it is a wastefull for drifting systems. In order to allow the EA to restart from a better position, a memory must be added. Usually, the memory does not store a complete population, but just numerical information on the best and/or the worst decisions that have been done in the past. In some cases is the operator rates, the number of violated constraints in the individuals, individuals that represents populations (virtual liders [15]), etc. Experiments were also carried by storing complete populations in order to study the readaptation capabilities faced to important environmental changes. The conclusion is that the evolutionary process must be started from early populations, since in other case, the convergence produced on them makes impossible for the EA to get good solutions [13].

There exists a lot of other approaches to the development of controllers for changing environments. One example is the so called continuous learning [14], in which an evolutionary algorithm updates continuously a knowledge base that defines the behavior of the controller. It has been successfully applied to environments with important changes.

## 2 The evolutionary process

The EA (see figure 1) applies the selection, crossover and mutation operators to a population of individuals. Each individual represents a complete controller [1], which in our case, means that it contains all parameters that defines a neurocontroller. Each individual is evaluated by applying it to the target system. A fitness value is assigned to each neurocontroller as a measure of its performance in the control task. The first population is created as a set of random neurocontrollers. It is expected that with a controlled application of the three operatores, each population of individuals will exhibit better performance.

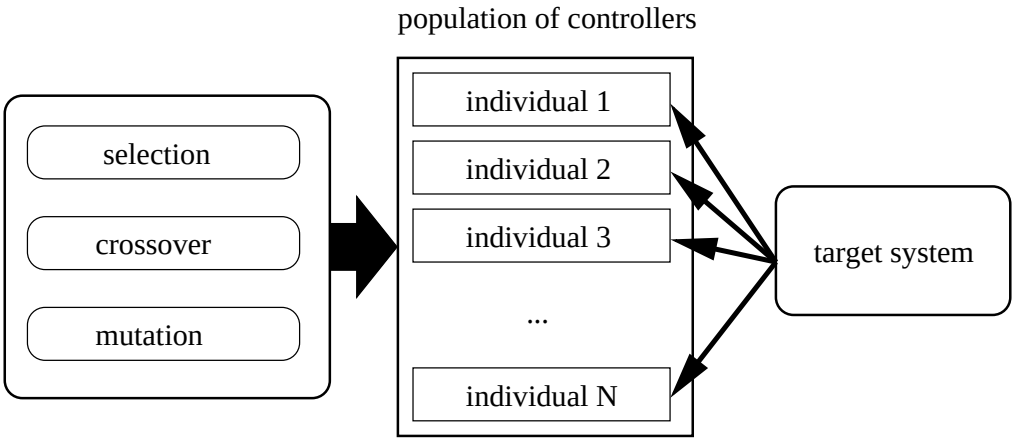


Figure 1: An evolutionary algorithm that evolves controllers

## 3 The auto re-startable evolutionary algorithm

In this section we introduce the auto re-startable evolutionary algorithm (AREA) model (see figure 2). Its primary motivation is to evolve neurocontrollers that are adequate to be applied in changing environments of the first EUNITE type.

The main algorithm is a standard EA that is applied to a random initial population of neurocontrollers. Once the evolutionary process is finished, the best neurocontroller is selected and applied to the target system. A distinctive feature of the AREA model is the fact that the EA stores selected intermediate populations that will be used in the future to restart the evolutionary process if adaptation is required. The AREA model includes a quality control component that is used to evaluate the performance of the controller in the target system while the system is running. The quality control component signals the restart evolution module when a drift is detected in the behavior of the target system, or in other words, when the controller cannot get the expected performance level. The restart

[1] It is also possible to allow each individual to represent part of the solution. This is usually the case with the so called symbiotic approach [12] [11].

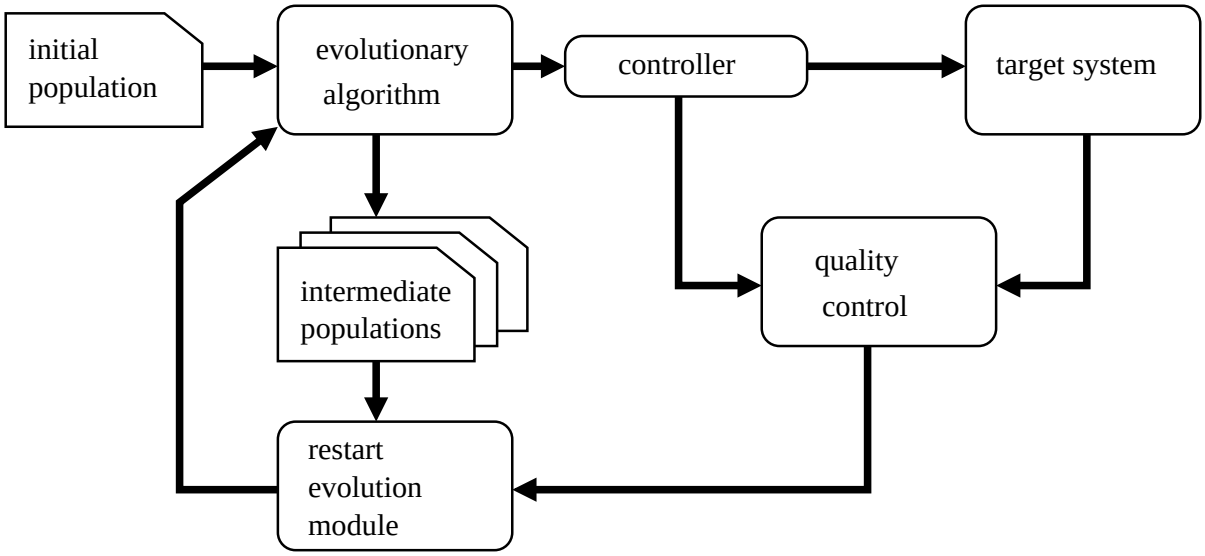


Figure 2: The auto re-startable evolutionary algorithm

evolution module then selects a particular intermediate population in order to start an evolutionary process to get a controller that is adequate to control the target system.

## 3.1 Intermediate populations

As it was quoted in the previous section, the EA selects intermediate populations during the evolutionary process and stores them. They are selected in the different stages of the evolution, considering the so called genetic diversity. Genetic diversity is a property of the populations that indicates the difference between the individuals [10]. It is usual that the genetic diversity reduces its value during the evolutionary process from a maximum (when all individuals are random) to zero when the population has converged to a single dominant individual [2]. The following sets of intermediate populations are selected:

- a small set from the very first generations, or in other words, populations with high genetic diversity.
- a bigger set when the fitness (or performance of the evolved neurocontrolers) increases, or in other words, populations in which the genetic diversity is being reduced.
- a small set when the population has converged, or in other words, populations with low genetic diversity.

The objective behind this selection process is to have a representative set of intermediate populations that corresponds respectively to controllers that are not capable of controlling the target (first

[2] Note that there exists a lot of techniques that avoids this behavior.

set), controllers that are developing good abilities to control the target (second set), and controllers that can successfully control the target (third set).

Controllers in the last set are very good, however, the populations have lost genetic diversity, and only similar controllers can be evolved from this population without introducing high levels of mutation [3].

## 3.2 Quality control

The objective of the quality control (QC) module (see figure 3) is to determine when the evolutionary process has to be restarted, based on the quality of the control that the current controller is performing. It has to provide also a measure of the drift of the target system.

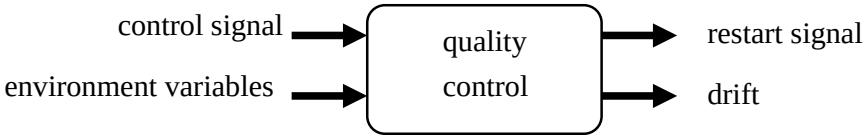


Figure 3: The quality control module

The implementation of the module depends on the target system and the complexity of the evaluation task. For example, the quality control module of a temperature control system that has to keep a constant temperature, can just be defined as threshold based function. The module generates the restart signal when the difference between the expected value and the obtained value is greater than an error $\epsilon$ for more that a $\delta$ length time period. The magnitude of the drift can be measured as the difference between the expected and the obtained temperature.

The QC module can also be implemented with fuzzy rules, a neural network, or any other convenient method. It can also be implemented as a system that compares the performance of the controller in the past with the current performance, discovering in this way the drift produced in the target system.

## 3.3 Restart evolution module

The restart evolution (RE) module (see figure 4) has to select an adequate intermediate population to restart the evolutionary process based on the restart signal produced by the QC module and the magnitude of the drift.

For very small drifts, one of the populations belonging to the last set has to be selected (see subsection 3.1). In this way, with low genetic diversity, the evolutionary process can get in very few generations a new controller adapted for the new control problem. For greater drifts, a population belonging to the intermediate set has to be selected. The number of generations needed to obtain a valid controller will be greater than in the previous case, but certainly smaller compared with a fresh start of the evolutionary process (with a random population).

[3] Mutation behaves just as a kind of random search

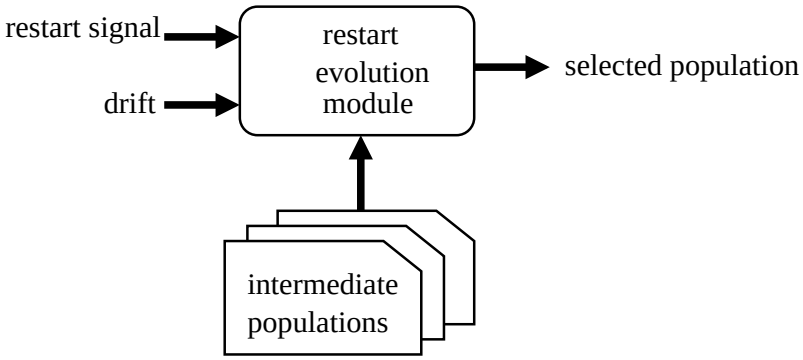


Figure 4: The restart evolution module

A typical implementation of the RE module is with a rule base, fuzzy rules or even neural networks.

# 4 A case-study: The cart-pole balancing system

This section describes an experiment that shows that the AREA model can be used to get a neuro-controller for a cart-pole balancing system (CPBS) which drifts over time. The CPBS, also called inverted pendulum system, is usually used as a benchmark in control. In our experiments, we introduce changes in the physical parameters of the CPBS in order to simulate a changing environment of type 1. The problem consists in determining the force to be applied, to push to the left or to the right, a cart with a pole on top of it. The pole must remain balanced and the cart between the track boundaries. The system is inherently unstable and the solution is an oscillatory application of forces first in one direction and later in the other. The system is simulated by using Euler's and Runge-Kutta numerical approximation for solving the second order differential equation that defines the physical system, with a time step integration of 0.005 seconds. This problem is well understood and the equations represent an extremely accurate simulation including accounting for friction between the cart and the pole. The figure 5 shows a simple representation of the CPBS system.

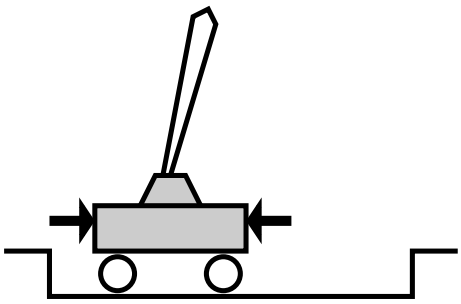

Figure 5: The cart-pole balancing system

The equations are defined in many books and papers, like for example in [4]. The physical parameters used during the simulations are:

| parameter | description | value |
|---|---|---|
| $m_c$ | cart mass | 1 Kg. |
| $m$ | pole mass | 0.1 Kg. |
| $l$ | pole length | 1 m. |
| $\mu_c$ | friction of cart on track | 0.0005 |
| $\mu_p$ | friction of pole on cart | 0.000002 |

## 4.1 The neurocontroller

The neurocontroller is implemented as a radial basis function (RBF) artificial neural network [9] (see figure 6). The box represents the neurocontroller with one input for each of the state variables (angle and angular velocity of the pole), and one output for the force.

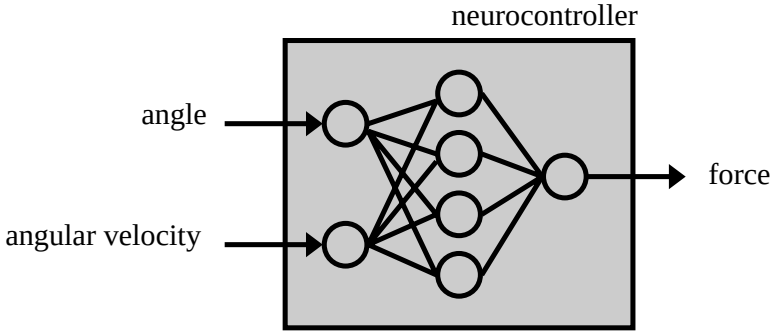


Figure 6: An example of a neurocontroller defined by an RBF artificial neural network with two inputs, four hidden units and one output

The model has three layers of units: one input layer, a layer of hidden units and one or more output units. The hidden units implement the so called RBF function, where $x$ is the input vector, $n$ the number of inputs, and $c$ and $\sigma$ are the parameters of the unit:

$$y = \sum_{i=1}^{n} e^{-\frac{(x_i - c_i)^2}{\sigma^2}} \tag{1}$$

The output units compute just a weighted sum of the outputs produced by the hidden neurons:

$$z = \sum_{i=1}^{m} w_i * y_i \tag{2}$$

This model has a set of interesting properties:

- it is an universal approximator since it can approximate arbitrarily well any continuous function.

- it has the best approximation property, since there exists always a selection of the coefficients that approximates better that any other possible selection.
- it provides an optimal solution since it minimizes a function that measures the current deviation from the real value.

The model is local in the sense that the hidden units produce output only in a specific region of the input space. This is an important property that is fully exploited in the AREA model, since when there are changes in the environment, it is quite possible that only few weights have to be modified, producing a faster learning.

## 4.2 Evolutionary algorithm

The EA maintains a population of individuals that represent each one a complete neurocontroller. Each individual is implemented as a string of variable length of triples $R_i$:

$$Ind = (R_1, R_2, \ldots, R_m) \tag{3}$$

where each $R_i$ defines a complete RBF unit and contains three values:

$$R_i = (c_i, \sigma_i, w_i)$$

In this way, an individual codifies a complete RBF network, including structure and weights. The EA evolves the population by using crossover and mutation operators that are standard for variable length individuals. The fitness is computed by evaluating the performance on the control task (see figure 7).

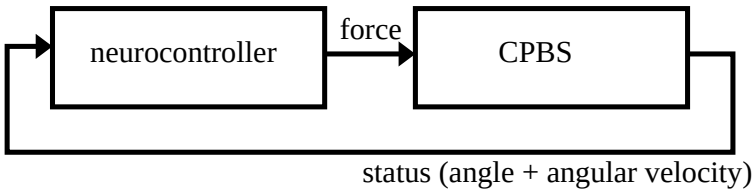


Figure 7: The computational model of the fitness

In our example, the CPBS receives the output of the neurocontroller (force). The neurocontroller then gets the updated status variables (angle and angular velocity), which are in turn used to compute the next control signal. This process is repeated for a predefined number of cycles (5000 in our experiments) or till the pole falls down. Each individual receives then a fitness value defined as follows:

$$\text{fitness(Ind)} = \sum_{k=1}^{r}(s + \sum_{t=1}^{s}((maxA)^2 - a_t^2))$$

This function gives higher values to the neurocontrollers that succeed in keeping the CPBS inside the expected ranges for a long time, and as close as possible to the center position. In the definition of the function, *s* denotes the number of steps the neurocontroller is able to keep the CPBS inside the expected ranges, *maxA* is the maximum allowed angle for the pole, *a* is the angle of the pole at time step *t*, and *r* the number of times each individual is evaluated.

The figure 8 displays the fitness of the best individual on a typical execution of the EA with a population of 100 individuals, a crossover rate of 0.9, mutation rate of 5% and a maximum number of evaluation cycles of 5000 for each individual. Note that after generation 150, the best individuals can keep the CPBS stable for the whole period.

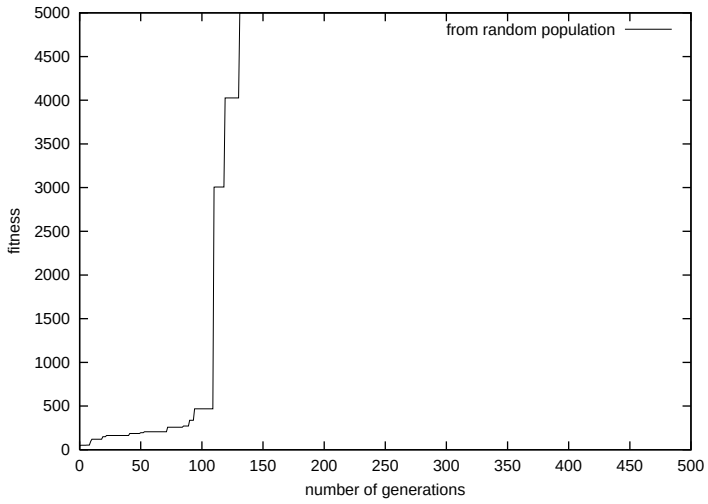


Figure 8: Best fitness at each generation when the EA is applied to the PCBS

The figure 9 corresponds to the window of the CPBS simulator in which the changes in the angle of the pole and its angular velocity are shown. The center horizontal line corresponds to the target position with both angle and angular velocity equal to zero. The best individual can succesfully control the system starting from a random position.

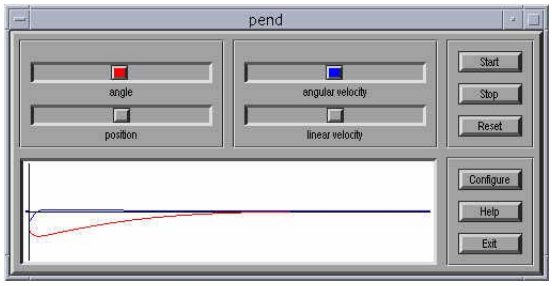


Figure 9: The simulator window of the PCBS

## 4.3 Readaptation

The weight of the cart in the CPBS is modified in a 10% in order to simulate a change of type 1 in the environment. The figure 10 shows that the best controller obtained by the evolutionary algorithm cannot succesfully control the CPBS now. The system is unstable and the pole will certainly fall down.

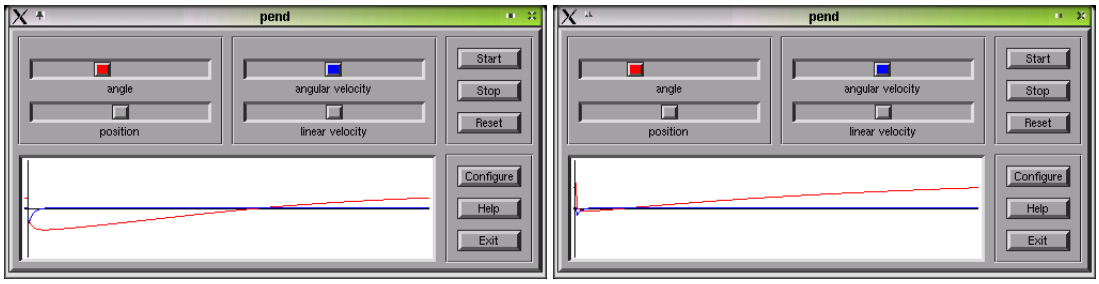


Figure 10: The simulator window of the PCBS when the best controller is applied to a modified environment

The figure 11 displays the fitness of the best individual in the modified PCBS by using an EA that starts from a random population and the AREA model starting from a selected population (in this example is from generation 80). It can be seen that the number of generations needed is clearly smaller when the evolution is started from an intermediate population. The figure also shows that the best fitness obtained in the first generation is higher when the evolution is started from an intermediate population.

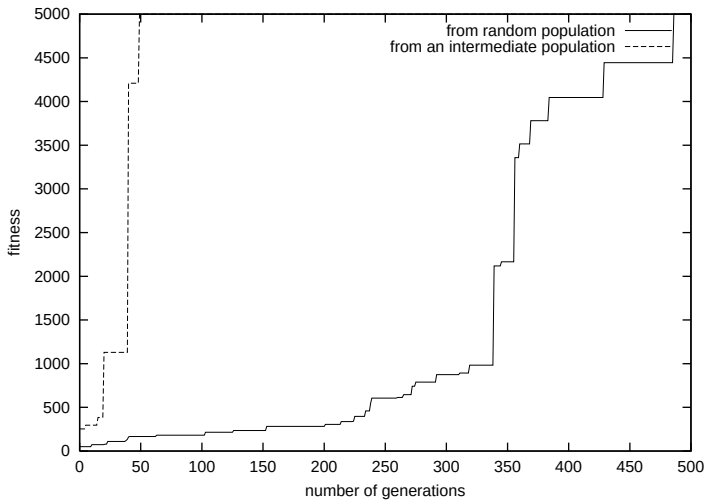


Figure 11: Best fitness at each generation when the EA is applied to the CPBS starting from a random population and from a selected intermediate population

# 5 Conclusions

Most of the current research in evolution of neurocontrollers is applied to static systems. This work shows that the AREA model is a promising approach for the evolution of neurocontrollers for changing environments. When the QC module detects a drift in the target system, the EA is restarted from an intermediate population selected by the RE module. The AREA model applied to the CPBS system shows that the time to get new controllers is significatively reduced. Current research is being performed now in the interaction of the modules and the aplication of the AREA model to other control problems.

# References


[1] R. S. Sutton A. G. Barto and C. W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 13, 1983.

[2] Azarmi N. Azvine B. and Tsui K. C. Soft computing - a tool for building intelligent systems. *BT Technical Journal*, 14(4), 1996.

[3] Jeffrey Bassett and Keneth DeJong. Evolving behaviors for cooperating agents. In Z. W. Ras and S. Oshuga, editors, *ISMIS 2000*. Springer Verlag, 2000.

[4] H. R. Berenji and P. Khedkar. Learning and tuning fuzzy logic controllers through reinforcements. *IEEE Transactions on Neural Networks*, 3(5), 1992.

[5] Wu Geng feng Carlos Kavka, María Liz Crespo and Fu Zhong quian. Fuzzy systems generation through symbiotic evolution. In *Symposium on Engineering of Intelligent Systems*, February 1998.

[6] Andrew Chipperfield and Peter Fleming. An overview of evolutionary algorithms for control systems engineering. Technical report, The University of Sheffield, May 2002.

[7] Alan Schultz David Moriarty and John Grefenstette. Evolutionary algorithms for reinforcement learning. *Journal of Artificial Intelligence research*, 11, 1999.

[8] EUNITE. Smart adaptive systems: State of the art and future directions of research. Technical report, EUropean Network on Intelligent TEchnologies for Smart Adaptive Systems, November 2001.

[9] Simon Haykin. *Neural Networks, A Comprehensive Foundation*. Macmillan College Publishing Company, 1994.

[10] Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, 3rd edition, 1999.

[11] Risto Miikulainen and David Moriarty. Efficient reinforcement learning through symbiotic evolution. *Machine Learning*, (22):11–33, 1996.

[12] Risto Miikulainen and David Moriarty. Forming neural networks through efficient and adaptive coevolution. *Evolutionary Computation*, (5), 1998.

[13] Stefano Nolfi and Dario Floreano. *Evolutionary Robotics : The Biology, Intelligence, and Technology of Self-Organizing Machines*. MIT Press, 1st edition, November 2000.

[14] Alan Schultz and John Grefenstette. Continuous and embedded learning in autonomous vehicles: Adapting to sensor failures. In Robert Gunderson Grant Gerhart and Chuck Shoemaker, editors, *Unmanned Ground Vehicles Technology*. SPIE, 2000.

[15] Michele Sebag and Marc Schoenauer. Toward civilized evolution: Developing inhibitions. *Proceedings of the ICGA*, 1997.