Chapter 8

M. O. Odetayo
D. Dasgupta
Department of Computer Science
D Montfort University
Leicester LE1 9BH
U.K.

odetayo@cs.unm.edu

Controlling a Dynamic Physical System Using Genetic-Based Learning Methods

Abstract

This chapter presents two different approaches of designing genetic-based controllers for an unstable physical system (a simulated pole-cart system). One approach induces *rule-base controller* using a simple genetic algorithm (GA) and the other evolves *neuro-controller* applying a recently developed Structured Genetic Algorithm (sGA) which appears to offer improvements over a simple GA approach. The control task here is a typical unstable, multi-output, dynamic system in which a pole is supposed on a controllable cart, and the controller must keep the pole upright (within a specified vertical angle) and the cart within the limits of the given track. In this chapter, we first describe a simple GA based learning method for inducing control rules, and then demonstrate the evolvability of neuro-controller using a Structured GA.

Introduction

When building a controller for a dynamic system, traditional control theory requires a mathematical model to predict the behaviour of the system. In many cases this cannot be done, either because the system is too complicated or because insufficient information about its environment is available. The pole balancing problem is one such inherently unstable classical control problem. The complexity of the task is significant enough to make the problem interesting while still being simple enough to make it computationally tractable.

We have chosen this task for several reasons, they include:

Many learning algorithms [19][1][24][5][23] have solved the problem in the form considered here and therefore we have a good test bed for evaluating the effectiveness of the genetic algorithm-based learning method.

The task is regarded as an example of the inherently unstable, mutiple-output, dynamic systems present in many balancing conditions such as the aiming of a rocket thruster [1][4][13]. The system is non-linear and has multi-output interacting parameters that are to be controlled. It is inherently unstable — the system can only be controlled by an appropriate thrust at the base of the cart.

• The way the task is set up creates a genuinely difficult credit-assignment problem and therefore poses a great challenge to a learning method.

• The complexity of the problem could be increased to a desired level by balancing 1, 2, ..., etc. poles each on top of the next [19].

We shall discuss how the time to learn and the amount of computation required by GAPOLE compares with those taken by learning methods that have previously solved the problem [23]. We show that it copes well with changing conditions and that it is an effective alternative technique.

Genetic Algorithms (GAs) are a class of adaptive general purpose methods, for machine learning and optimisation, based on the principles of population genetics and natural evolution. However, not much has been done to determine their suitability as a machine learning and adaptive control tool for other more general applications. Knowledge in this field is being advanced rapidly, however.

This chapter is divided into two main sections. The first section discusses our experiments using a simple genetic algorithm and compares the performance with other classical AI methods. We developed and implemented a simple GA-based program (GAPOLE) for inducing control rules for a dynamic physical system: a simulated pole-cart system. The second section describes the use of a structured genetic algorithm for automatic designing neuro-controller for the same task.

In the following, we briefly describe the principle of genetic algorithms and develop a Simple GA-based learning system, called GAPOLE, and assign it the task of inducing control rules for a dynamic system — a simulated pole-cart system. The dynamics of the pole-cart system are not made available to the algorithm. The only evaluative feedback indicating how well it is performing is a failure signal which occurs when the system is out of control. That is, either when the cart has gone beyond the track limit or the pole has fallen past a predefined vertical angle. This presents a big challenge to a learning method as the effect of a wrong action may not be known until several steps later. Thus training information may be delayed making it difficult to correctly credit individual actions.

The GAPOLE program was used to derive (or 'breed') a species of controllers that give a specified level of performance. Comparison of its performance (time to learn and the amount of computation) with the best available alternatives showed that it compares well, but it is noteworthy that it performs well in a "noisy" and changing control environments.

8.1 The Control Task

The task is to move a wheeled cart, with a rigid pole hinged on top of it, along a bounded straight track without the pole falling beyond a predefined vertical angle and without the cart going off the ends of the track limits. This is achieved by applying a force of fixed magnitude (a 'bang-bang' force) to the left or right of the cart (Figure 8.1).

The state of the pole-cart system at any time t is specified by four variables:
x = position of the cart on the track, where:
$\dot{x}$ = velocity of the cart.
θ = angle of the pole with the vertical.
$\dot{\theta}$ = angular velocity of the pole.

The pole-cart system was simulated using the following equations of motion derived by Anderson [1] with the given parameter values:

$$\ddot{\theta}_t = \frac{mg\sin\theta_t - \cos\theta_t[F_t + m_p L\dot{\theta}_t^2 \sin\theta_t]}{L[(4/3)m - m_p\cos^2\theta_t]}$$

$$\ddot{x}_t = \frac{F_t + m_p L[\dot{\theta}_t^2 \sin\theta_t - \ddot{\theta}_t\cos\theta_t]}{m}$$

where
m_c = 1.0 kg = mass of the cart.
m_p = 0.1 kg = mass of the pole.
$m = m_c + m_p$ = 1.1 kg = total mass of the system.
L = 0.5 m = distance of centre of mass of pole to the pivot.
g = 9.8 ms^2 = acceleration due to gravity.
F_t = force applied to cart (of specified magnitude).

A time step of r = 0.02 seconds and the following discrete-time state equations were also used in the simulation.

$$x_{t+1} = x_t + \tau\dot{x}_t \qquad \dot{x}_{t+1} = \dot{x}_t + \tau\ddot{x}_t$$
$$\theta_{t+1} = \theta_t + \tau\dot{\theta}_t \qquad \dot{\theta}_{t+1} = \dot{\theta}_t + \tau\ddot{\theta}_t$$

The state space can be regarded as a four dimensional space and a state variable defines each dimension. The dynamics of the physical system being unknown to the controlling system; the only information for evaluating performance is a failure signal indicating that the pole-cart system is out of control. The pole-balancing problem consists of: 1) how to divide the quantity space of each variable into a small set of intervals; 2) what action to select for each combination of intervals describing the state of the pole-cart system in a way such that

• the poles are balanced i.e do not fall.
• the cart does not leave a predetermined limited track.

It is a good test bed for evaluating the effectiveness of investigating the learning performance of the genetic-based systems because:

• there is randomness in the task;
• the dynamics of the system are not known to the learning program; and
• it is a difficult and challenging task.

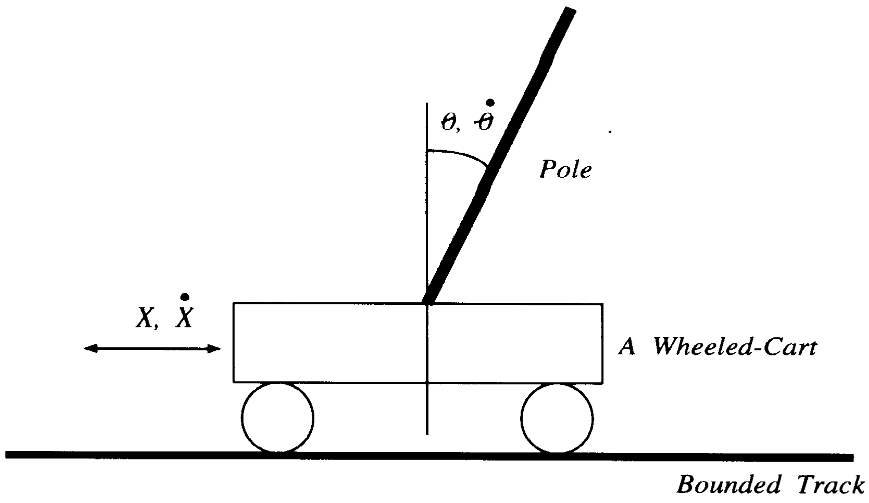


Figure 8.1: A pole-cart system on a bounded track.

The dynamics of the pole-cart system are unknown to the learning controller. The only information available to it at discrete time steps is either a vector indicating the current state of the system or a failure signal telling it that the system is out of control. In this experiment, the system is out of control when the cart has gone beyond ±2.4 meters from the centre or the pole has fallen beyond 12° from the vertical. These limits were employed for the three alternative methods for the same task [23].

The learning ability of the genetic-based system which can carry out a complex task was demonstrated by our work on the pole-balancing system[22]. Comparison of its performance with the best available alternatives showed that it compares well, but it is noteworthy that it is robust, and performs well in *noisy* and changing control environments.

In the following subsections, we briefly state the previous learning methods for the pole-cart problem and discuss the working principle behind GAs. We then give a detail description of the experiments for inducing simple GA-based control rules and the comparative results with other AI methods.

8.2 Previous Learning Algorithms for the Pole-Cart Problem

We review the three best learning algorithms — BOXES, AHC and CART — that have been applied to the problem of learning to control a simulated pole-cart system.

8.2.1 BOXES

Miche and Chambers [19] developed a program known as BOXES for learning to control the pole-cart system. They reduced the problem space into manageable proportion by partitioning it into disjoint regions called boxes. This was achieved by quantizing the four state variables. The quantization thresholds were predefined before the experiments started. They used a total of 225 partitions or boxes.

Each box is imagined as having a local demon that decides where the pole-cart system should move next (left or right) whenever it enters its box. In order to do this, a demon gathers data about its box through the following sets of variables [19]:

LL, the 'left life' of its box, which is a weighted sum of the 'lives' of left decisions taken on entry to its box during previous runs. (The 'life' of a decision is the number of further decisions taken before the run fails.)

RL, the 'right life' of its box.

LU, the 'left usage' of its box, which is a weighted sum of the number of left decisions taken on entry to its box during previous runs.

RU, the 'right usage' of its box.

TARGET is a figure supplied to every box by the supervising demon, to indicate a desired level of attainment, for example, a constant multiple of the current mean life of the system.

T_1, T_2, ..., T_n times at which its box has been entered during the current run. Time is measured by the number of decisions taken in the interval being measured, in this case between the start of the run and a given entry to the box.

It uses a function of these variables to rate how good a decision to go right or left was. If the right value is greater than the left value then the demon would move the system to the right and vice versa.

Weighted averages of the lifetimes of the pole-cart after a decision to go left or right are used because estimates of the worth of a decision are inaccurate in the early stages of a trial (a trial is the period the system is kept under control from a starting position before failure occurs). For example, the direction in which the system should go whenever it enters a box may be correctly set, say to go right, but if the directions for the boxes around it are not properly set, then its decision to go right may appear bad [23].

Another interesting feature of the algorithm is its solution to the problem of getting stuck at a local peak. In order to prevent a box from settling down to a seemingly good direction (left or right), it compares an optimistic projection of how well each decision may perform [19][23]. For example the value of going left is determined as follows:

$value_L = LL + K \bullet TARGET^{LU+K}$, where K was set to 20.

8.2.2 AHC

AHC (Adaptive Heuristic Control Algorithm) partitioned the problem state space into predefined regions like the BOXES. However, unlike BOXES, learning takes place during a trial as well as at the end of it.

It uses four parameters to evaluate the performance of a box and to decide which action to take whenever the system enters the box. The parameters for a box are [23]:

• ACTION, a real number that is used to determine whether to go left or right.

• MERIT, a measure of its ability to predict the correct action.

• ELIGIBILITY determines if it is eligible to change from going one way to the other.

• FREQUENCY, a weighted count of the number of entries into the box.

A negative value of ACTION represents a tendency to go left, while a positive value represents a tendency to go right. The bigger the magnitude the greater the tendency. The actual direction is determined by mapping the action into a range from zero to one, then generating a random number between zero and one, and comparing the two. If the random number is less than the mapped number then go left, if not, go right.

8.2.3 CART

CART [5] does not partition the state space into regions. It has as its main goal the ability to accurately estimate the desirability of a state and therefore tries to avoid bad ones. Thus its main thrust is the search for a more desirable state.

CART chooses an action that is estimated to lead to a desirable state. At every step, it decides whether the same action as the last should be repeated or the action should be changed. If it is estimated that the pole-cart system will move to a more desirable state by continuing with an action, then the action is chosen; if not the other action is selected.

As learning progresses it gathers information that enables it to improve its ability to estimate the degree of desirability of the pole-cart states. It does this by labelling certain states in the trial as desirable or undesirable. This is achieved in three ways:

(1) It starts the learning process from the state when the pole is upright, the cart is centred, and the velocites (angular and cart) are zero. This initial state is labelled as a desirable state.

(2) An undersirable state is reached when the pole-cart is out of control (when the pole falls or cart has gone past the defined limits). The state immediately preceding the failure is labelled as undesirable, unless its degree of desirability is already less than -0.98.

(3) At the end of a trial that lasted more than 100 time steps, it backs up 50 states from the failure point; and from then on, searches for a state in which at least three of the state variables are approaching zero in magnitude, i.e., are approaching the starting state. This point is labelled as desirable.

The algorithm uses a function to interpolate from a chosen set of states (known as the training set) that have been labelled (as desirable or undesirable) in order to improve its accuracy in estimating the desirability of a pole-cart state. In order to do this, it views the desirability of a state as the height of a surface in five dimensional space. The first four dimensions represent the state, and the fifth represents its degree of desirability. The surface is changed after each trial as new points are used to evaluate the interpolating function.

8.3 Genetic Algorithms (GA)

Genetic Algorithms (GAs) are iterative adaptive general-purpose search strategies, based on the principles of population genetics and natural selection [14][12]. They simulate the mechanics of population genetics by maintaining a population of knowledge structures, analogous to the gene pool of a species, which is made to evolve.

An outline of the generic Genetic Algorithm is given below:

```
Initialise P(t=O); /, P(O) = initial population ,/
  Evaluate members of P(t);
   While (not termination condition)
    {
      Generate P(t+1) from P(t) as follows:
       {select individuals from P(t) on basis of fitness;
         recombine those selected;
       }
        t = t+1;
       evaluate members of P(t);
    }
```

A GA therefore learns by evaluating its knowledge structures using the fitness function, and forming new ones to replace the previous generation by breeding from more successful individuals in the population using the crossover and the mutation operators.

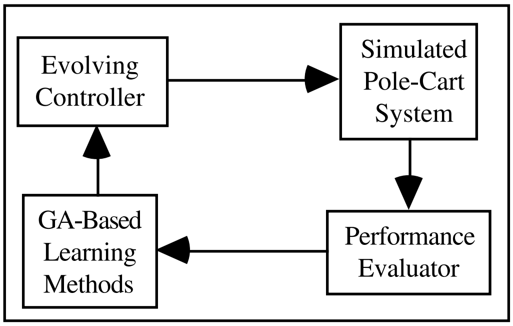


Figure 8.2: A genetic-based control system (GAPOLE).

8.4 Generating Control Rules Using a Simple GA

A simple GA-based learning program (GAPOLE) we developed for the pole-cart balancing problem described above. It consists of 4 components: A fixed population size of learning (rule-base) controllers; the learning algorithm; the performance evaluator; and the simulated pole-cart system. Their interaction is shown in Figure 8.2.

The following pseudo code shows how the chromosomal information (a set of directions) is used to control the pole-cart system:

```
while (state_of_pole != FALLEN and time_pole_held< MAX_HOLD)
 {Increment time_pole_held; move_system;}
```

8.4.1 Population of Learning Controllers

A learning controller is a set of production rules for controlling the pole-cart system. A production rule has a format as follows:

condition then **action**

The specified action will be performed when the condition is satisfied. A controller is regarded as a chromosome by the learning algorithm. We use the two names interchangeably without any loss of meaning.

8.4.2 Representation

As we stated in defining the control task, the state of the pole-cart system is specified by four real-valued variables. The state space can therefore be regarded as a four dimensional space. A state variable defines each dimension. At each point in the state space, the learning controller is required to decide whether the pole-cart system should go left or right so as to keep it under control. This implies that it has an infinite number of points and so the state space is reduced to manageable proportions by partitioning it into predefined regions as in [19] so that points within a region are mapped into the same decision.

We experimented with a number of partitions, taking full advantage of those used in [19]. The set of partitions we found that gave the best results and which we employed in our experiments are:

x (cart position): - 2.4 to 2.4 metres [1 region]
$\dot{x}$ (cart velocity): ∞ ms^{-1} to - 0.5, - 0.5 to 0.5, 0.5 to ∞ ms^{-1} [3 partitions]
θ (pole angle): 12^{o} to - 6, - 6 to - 1, - 1 to 0, 0 to 1, 1 to 6, 6 to 12° [6 partitions]
$\dot{\theta}$ (angular velocity): $-\infty^{\circ}s^{-1}$ to - 50, - 50 to 50, 50 to $\infty^{\circ}s^{-1}$ [3 partitions]

This creates a total of 54 regions (1*3*6*3). Conceptually a region can be regarded as a production rule with its condition part specified by the values of the state variables it covers and its action part specified by the direction of the pole-cart system whenever it is in that region. Thus the learning algorithm is required to evolve a set of 54 rules that will be able to keep the system under control.

We represent a sequence of these 54 regions (rules) — a controller — as a *chromosome* with a region regarded as a gene. A gene takes on a '1' indicating a left move or a '0' indicating a right move. At a time step, t, the pole-cart system will be at a gene (region) and the direction it moves next depends on whether the gene has a '1' (left move) or a '0' (right move). An individual chromosome, therefore, is made up of a string of '1's and '0's.

8.4.3 Performance Evaluator

The performance evaluator rates a chromosome (controller) by assigning it a fitness value. The value indicates how good the chromosome is in balancing the pole-cart system. The evaluator uses the length of the time (number of discrete time steps) that a chromosome holds the pole-cart system (from an initial position or state) without a failure as its fitness value.

8.5 Implementation Details

The population size affects the performance and efficiency of Simple Genetic Algorithm-based systems. A small size provides an insufficient sample, which makes them perform poorly. A large population size will undoubtedly raise the probability of the algorithm performing an informed and directed search.

However, a large population requires more evaluations per iteration or generation, possibly resulting in the method developing redundant controllers and becoming very slow especially when implemented serially. We experimented with different population sizes, they include 100, 150, 300 and 400. We found a population of 300 to be optimal for our task, but in general we assert that the optimum population size depends on the complexity of the domain, and in particular, on the shape of the fitness function.

We implemented a modified overlapping population in our simulations. First we do not replace a fixed percentage of the population from generation to generation, instead the percentage is allowed to vary dynamically within a fixed interval. Our aim is to strike a good balance between exploration and exploitation.

Secondly, we do not select those to be replaced randomly; we use the fitness value of an individual chromosome and the average fitness value of the population to determine whether or not a particular individual is to be replaced.

Since we do not generate a completely new population at each generation, some population members will continue unchanged into the next generation. This is at variance with the natural evolving process in which no member passes unchanged to the next generation. However, advantage can be taken of this 'immortality' when implementing a GA method on a computer system so as to minimise computation of evaluations of new entrants to the population. Also, in our application, we are interested in maintaining high performance levels as the algorithm learns to control the system; we therefore need to preserve the best rules so far discovered while we continue to search for better ones. This will ensure that the best information gained about the environment is not lost between generations. Our earlier simulations showed that it is difficult to preserve the best information with nonoverlapping populations.

Since we use a population of fixed size, some members have to be removed to make room for the newly generated ones. We decide on population members to be retained, if the termination condition has not been reached is as follows:

At the end of a generation, the average fitness of the current population is calculated. Individuals whose fitness values fall below the population average are replaced except when

(a) less than 20% of the population will survive (by 'survive' we mean continue into next generation unchanged) to the next generation; the best 20% of the population are retained;

(b) more than 60% of the population will survive; the best 60% are retained provided this has not been the case for more than 3 consecutive generations.

These measures are designed to discourage very few individuals from dominating the population, to ensure that adequate points are sampled in a generation and to increase diversity as soon as the algorithm detects that it has become low. The values of the parameters presented above were arrived at through experimentation.

Each offspring produced by crossover has a small probability (0.01) of being mutated. The number of offspring a survivor is allowed to produce by crossover is proportional to its fitness value. An individual is regarded as reproducing through crossover if it is the first of a couple to be chosen.

A mate is chosen randomly for a reproducing chromosome among the remaining survivors until all its children have been produced. When the number of children produced this way is less than the total needed, the remaining ones are produced by randomly choosing pairs from survivors for the crossover operation.

One of the main problems with implementing simple GA-based applications using finite population sizes is the possibility of premature convergence; that is, the possibility of the system converging onto suboptimal peak. Premature convergence takes place when population members are identical in their gene composition before the true optimum solution has been found [17]. That is, it occurs when there is a loss of diversity in the gene pool.

In order to minimise the loss of diversity, we introduced some innovative measures in our implementation that enable the program to dynamically alternate between exploiting the accumulated knowledge and exploring the solution space as the need arises. In that section, we specified that the percentage of population members retained to go unchanged ('survive') into the next generation varied between a minimum value (20%) and a maximum value (60%). In addition to the above measures, we introduced a new individual into a population only if it is different from every other member of the population by at least one bit. An offspring that is identical to a member present in a population is regarded as stillborn.

The GAPOLE is required to learn to control the simulated pole-cart system for 10,000 time steps continuously without a failure signal. A learning session is completed when at least one population member achieves this level of performance or when the total number of points sampled exceeds 100,000 points. No population sampled points close to this limit before at least one of its members achieved the required level of performance.

8.6 Experimental Results

The simulation program was written in C and to two sets of experiments were carried out on a Sequent Balance B8000 computer. Each set consists of running our GA program 50 times; each time initialising the Unix's random number generator with a new seed. Directions were randomly fixed for the chromosomes at the start of a run.

In the first set of experiments with a simple GA, a force of 10 Newtons was applied to the right or left (-10 for a left direction) of the base of the cart while a force of 5 Newtons was applied to right and 10 Newtons (-10) to left of the base of the cart in the second set of experiments.

Pop.	Generations			Failures			Time taken(hr:min:sec)		
Size	Min	Max	Mean	Min	Max	Mean	Min	Max	Mean
100	6	544	28	472	28853	1643	2:46	6:37:21	23:24
150	2	42	14	259	3571	1301	0:41	1:56:39	16:13
300	3	21	9	729	3808	1842	1:48	2:14:31	17:04
400	4	21	9	122 8	5297	2394	2:58	2:55:25	23:06

Table 8.1: Performance summary of GAPOLE for population sizes of 100, 150, 300 and 400 when pushing left and right with a force of 10 Newtons.

Pop.	Generations			Failures			Time taken(hr:min:sec)		
Size	Min	Max	Mean	Min	Max	Mean	Min	Max	Mean
100	4	321	43	290	17260	2382	0:48	13:50:21	41:12
150	6	97	30	702	7612	2477	3:16	2:09:10	23:50
300	4	51	16	876	8324	2876	1:41	1:53:02	19:29

Table 8.2: Performance summary of GAPOLE for population sizes of 100, 150 and 300 when using a force of 5 Newtons to push right and a force of 10 Newtons to push left.

The results of the experiments on simulated pole-cart system are shown in Table 8.1 and Table 8.2.

Table 8.1 and Table 8.2 show that population size of 300 produced the best average computational times (17:04 and 19:29 minutes, respectively) for two experiment sets.

The simple GA-based learning program was regarded to have succeeded in balancing the pole-cart as soon as it was able to hold the system continuously without a failure signal for 10,000 discrete time steps. This is in line with the performance level set by Sammut [23].

Sammut [23] evaluated three best alternative learning algorithms (reviewed in the previous subsection) — BOXES [19], AHC [24] and CART [5] — that solved the pole-cart balancing task in the form considered in this chapter. He used 162 regions for BOXES and AHC in his experiments while we employed 54 for ours. CART does not divide the solution space into regions. These three algorithms are point-based (i.e., they generate, test and modify single solutions) while GAPOLE is population-based. At a generation or an iteration, therefore, they sample only one point in the solution space while the number of points sampled by our simple genetic algorithm-based program is equal to the number of new individuals introduced into the population at that generation. For our comparisons, a generation (or an iteration) when used for the point-based algorithms is equivalent to a trial or a sampled point.

The average number of generations and the average points sampled by the genetic-based method (GAPOLE), BOXES, AHC and CART to learn to balance the pole-cart system for 10,000 time steps using a force of 10 Newtons are in Table 8.3.

	GAPOLE	BOXES	AHC	CART
Iterations	9	225	90	13
Points	1842	225	90	13

Table 8.3: Average iterations and points sampled using a force of 10 Newtons. Averages for BOXES, AHC and CART were over 5 runs (Sammut [23]).

When the force applied to the pole-cart system was changed, i.e., a force of 5 Newtons was applied when going right and a force of 10 Newtons applied when going left, the average number of generations and the average points sampled by the four algorithms are given in Table 8.4.

	GAPOLE	*BOXES*	*AHC**	*CART+*
Iterations	16	837	2562	terminated
Points	2876	837	2562	terminated

Table 8.4: Average iterations and points sampled using 5 Newtons to push right, and 10 Newtons to push left. * The averages for AHC were taken over 4 runs. The fifth was stopped after it failed to achieve the performance level within 50,000 iterations (Sammut [23]) - the maximum number allowed. + CART terminated with a floating point exception (Sammut [23]).

The percentage increases in the number of generations and the number of points sampled by these algorithms when the force applied changed from being even to uneven is given in Table 8.5.

	GAPOLE	*BOXES*	*AHC*	*CART*
Generations	77.78%	272.00%	2746.67%	-
Points	56.13%	272.00%	2746.67%	-

Table 8.5: Percentage (%) increases in generations k points sampled when the force changed from even to uneven.

BOXES and AHC collect some statistical data for each box or partition in order to determine what their actions should be whenever the system is in that region. CART assumes that the control surface is described by a smooth function and thus cannot be used for surfaces that are discontinuous. Our technique neither keeps statistical information for each box nor assumes a particular type of surface. Also, it acts on the knowledge structures (controllers) syntactically. That is, it manipulates them without taking into consideration any interpretations given to these structures. Any knowledge structures can therefore be substituted for the population of controllers.

Our program uses a table you look up to make a control decision (the same holds true for BOXES) and thus makes the decision quickly. AHC takes a longer time to choose a control action since it revises the settings of its boxes after each step. CART computes two vectors and their inner product before it decides on a control action. As these calculations take quite sometime to perform, CART takes a considerably longer time to choose a control action than our technique.

AHC and CART are, however, able to learn during and after trials compared to GAPOLE that only learns after trials.

8.7 Difficulties with GAPOLE Approach

The GAPOLE simulation also showed that the evolution of viable rule-set (candidate solution) requires that parameters are restricted to a particular range, which alone is searched. Moreover, altering the direction of several boxes (regions) simultaneously in a generation could arrive at the solution point faster. But increasing mutation rates could be harmful to simple genetic algorithm-based systems as the search could degenerate to a random search with many non-viable offspring generated.

Another difficulty with the present (GAPOLE) approach is the partitioning of the search space (as in BOXES), the number of partitions to be used needs to be decided. If the number of partitions are too small than the control rules will be coarse and inaccurate, however, use of too many partitions may results in fine control action, but will be very difficult to induce control rules, using the GAPOLE approach, within a reasonable time and with reasonable effort. Moreover, static partitioning of state space appears to be inefficient for precise control in a dynamic system. An alternative genetic approach for generating a more robust viable controller will be investigated next for solving the problem.

8.8 A Different Genetic Approach for the Problem

In previous sections, we have seen that the pole balancing problem has often been used as an exercise in the control of dynamic systems and has been studied extensively by researchers in different fields. Other than classical control theory approaches, it has been solved mostly using different techniques of Artificial Intelligence (AI). They include machine learning [19], fuzzy logic [3, 20], qualitative modelling [16], neural networks [1, 2, 13], genetic algorithms [18, 25], etc.

In genetic approaches [21, 25], the problem state-space was divided (discretised) into a number of predefined partitions (as in the BOXES method). The genetic encoding was a binary string where each gene represented each partition and its value determined the appropriate action (push left or right). However, with these approaches the performance of the controller (or control rule) depends on thc number of partitions used and has difficulty in generalising the control rules [26].

For this pole-balancing problem, neural-based methods are widely used. The advantage of using neural networks is twofold: versatile mapping capabilities from input to output and its learning ability without explicit knowledge of mathematical basis of the system. It is widely recognised that the architecture of a neural network can have a significant impact on the network's function and processing capability. In most cases, predefined architectures are used for performing tasks with neural nets. Though genetic algorithms can replace the effort of human designers in determining network structures and also can be used for training predefined neural nets, but until recently, GAs were used for one or the other purposes (designing network structure or optimising neural net weights).

A combination of neural networks and genetic algorithms has also been used where a fixed network has been trained with genetic reinforcement learning [26]. The Genetic Cascade Learning algorithm was employed [15] to sequentially build the net to perform the task.

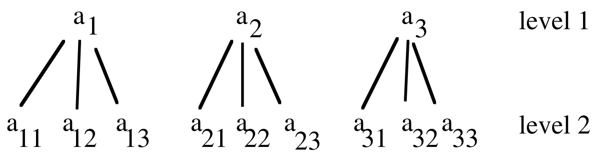
The method described below is an alternative neurogenetic approach, in which both the network architecture and its weights evolve together in an implicitly parallel fashion. In the remainder of this subsection we will give a brief description of Structured Genetic Algorithms (sGA) and then describe the application of sGA for the automatic design of neurocontrollers using genetic reinforcement learning.

8.9 The Structured Genetic Algorithm

The Structured Genetic Algorithm (sGA) [8][11] uses genetic redundancy and hierarchical genomic structures in its chromosome. Genes at different levels may be active *(on)* or passive (*off*) phenotypically. The primary mechanism for eliminating the conflict of redundancy is through higher level genes which act as switching operators for expressing genes at lower levels. The model also uses conventional genetic operators and the *survival of the fittest* criterion to evolve increasingly fit individuals. These characteristics allow the model to solve complex multi-stage problems.

In an sGA a chromosome is usually represented as a set of substrings. It also uses conventional genetic operators and the 'survival of the fittest' principle. However, it differs considerably from the Simple Genetic Algorithms in encoding genetic information in the chromosome and in its phenotypic interpretation. The fundamental differences are as follows:

• Structured Genetic Algorithms utilise chromosomes with a multi-level genetic structure (a directed graph). As an example, sGA's having a two-level structure of genes are shown in Figure 8.3a, and chromosomal representations of these structures are shown in Figure 8.3b.
• Genes at any level can be either active or passive.
• 'High level' genes activate or deactivate the lower level genes.


a_1
a_2
a_3
level 1
a_{11} a_{12} a_{13}
a_{21} a_{22} a_{23}
a_{31} a_{32} a_{33}
level 2

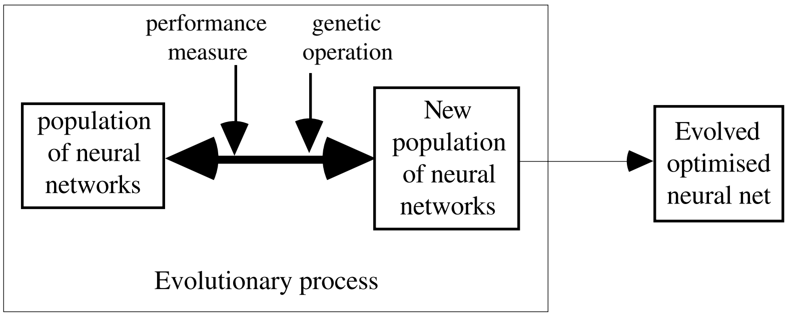
(a) A 2-level structure of sGA

(a1 a2 a3 a11 a12 a13 a21 a22 a23 a31 a32 a33) - a chromosome
and
(0 **1** 0 **1** 0 **1** 0 **1** 0 **1** 0 0) - a binary coding

(b) An encoding process of sGA

Figure 8.3: A simple representation of a two-level sGA.

While applying to the field of neural networks, the model can define the network structure and its connection weights in its chromosome, and these parameter sets can be optimized, in parallel, as a single unified process. In each generation, while some members of the population are engaged in searching for the feasible topology; others, which already have feasible structures, are searching for a set of optimal weights, and the process continues until a fully trained network evolves which can solve the task. The details of the model and its application for full designing of neural nets were explained in our previous work [6][7].



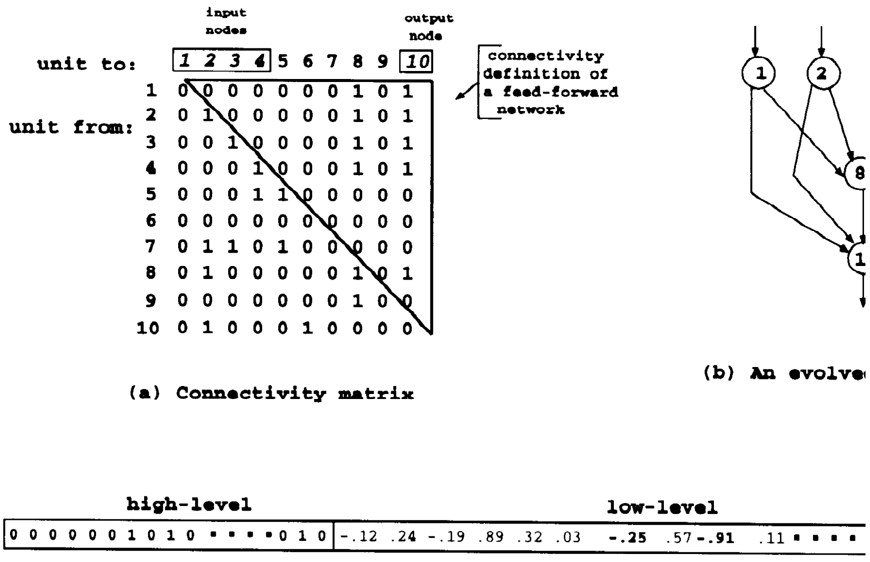
performance measure
genetic operation
population of neural networks
New population of neural networks
Evolved optimised neural net
Evolutionary process

Figure 8.4: Genetic process of evolving neural networks.

Figure 8.4 shows the working principle of the sGA for designing an application specific neural network architectures.

8.10 Evolving Neuro-Controllers Using sGA

For this empirical study we adopted a two-level sGA for encoding the complete neural network. Each individual (chromosome) has a two-level genomic structure in Figure 8.5c. The higher level defines the network configuration, while the lower level encodes the connection weights and biases. As mentioned before the high-level of the sGA searches the connectivity space of N units (to evolve an efficient network structure), while the low-level searches for an optimal set of weights to control the system. The fitness of each individual is determined by the combined performance of these two components [6]. A set of individuals (population) is generated randomly to initialise the evolution-learning process.



unit from: \ unit to:	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	0	1	0	1
2	0	1	0	0	0	0	0	1	0	1
3	0	0	1	0	0	0	0	1	0	1
4	0	0	0	1	0	0	0	1	0	1
5	0	0	0	1	1	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0
7	0	1	1	0	1	0	0	0	0	0
8	0	1	0	0	0	0	0	1	0	1
9	0	0	0	0	0	0	0	1	0	0
10	0	1	0	0	0	1	0	0	0	0

(a) Connectivity matrix

Figure 8.5: A two-level sGA representing neuro-controller.

Here we considered two vertical angles (12° *or* 35°) for balancing. As in previous sections, the initial starting position of the cart is randomly set between ±0.1 meters, the starting pole angle between $\pm 6^{\circ}$; the cart velocity and pole's angular velocity are set to 0.0 at the start of each training phase [18]. These values are considered as the initial inputs for each individual neural net at every generation. The input state vector is normalised so that the values lie in the range 0 and 1. The algorithm terminates if at least one evolved net holds the pole for 120,000 time steps (i.e., 40 minutes of simulated time) or the allowed number of iterations are used up (2000 generations).

8.11 Fitness Measure and Reward scheme

In every generation, each chromosome is decoded into its phenotype (a network structure with its weights), and its fitness is evaluated by taking into account the feasibility of the structure and its ability to learn the control task. More specifically, since a sGA is used to find both an architecture and the synapse weights, the evaluation function must include not only a measure the learnability of a net, but also a feasibility measure of network structure and its complexity (i.e., number of nodes and their connectivities).

In a randomly-generated initial population there are likely to be a large number of individuals which show poor performance due to two reasons:

1. They have a infeasible network structure i.e., improper connectivity pattern;

2. Arbitrary values of weight-bias parameters which may be far from optimum (even though the structure is feasible).

A network structure is infeasible

• If there exists no path from input nodes, and/or to output nodes,
• If there is fan-in to a hidden node but no fan-out or vice versa,
• If there is any unreachable substructure, etc.

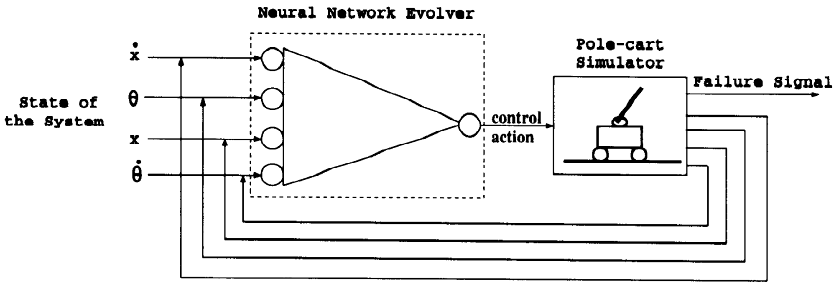


Figure 8.6: Reinforcement learning of neuro-controllers using sGA.

The infeasibility measures quantify the amount by which an individual structure exhibits 'congenital defects' (deformation).

It is necessary to avoid destructive interference between the two searches in their different spaces. For this reason, if an individual decodes to a feasible structure, it is rewarded by keeping its high-level portion stable (i.e., no changes are subsequently allowed to occur), and only the weight-bias space is then explored. However, while training a feasible net, if no improvement is noticed in balancing the pole for 50 successive generations, the individual then loses its structural stability and is downgraded or eliminated. Feasible individuals which have fewer nodes and links also get a selection advantage for reproduction relative to the competing feasible individuals with more complex structures. Since we are rewarding only the feasible structures, there is no chance of an individual structure getting reward by pruning all its connections and nodes so as to become infeasible.

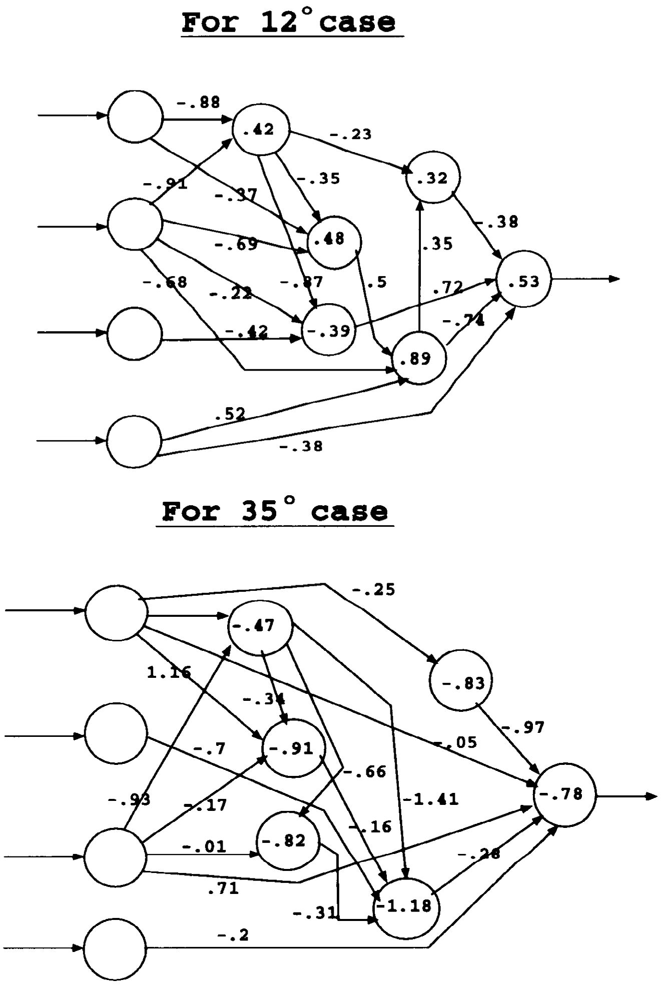


Figure 8.7: The evolved neural net controllers and their weights.

Thus each feasible net (individual) is trained through genetic reinforcement learning [26], where the learning process is also an object of evolution. Figure 8.6 shows the functional blocks involved during learning phase of feasible nets (controller's). The approach considers two 'black boxes' communicating with each other, neither knowing the internal dynamics of the other. The first 'black box' designs the controller to adapt the environment of the other through an evolutionary process. The second responds to the control action of the first, and feedback the system response at each time step. The only information for evaluating performance is a failure signal which indicates that the pole-cart system is/is not out of control.

The learning process of a feasible net starts by providing the initial state of the system to the net and the net's output response is applied to the simulated system. The output of the net is either 0.0 (push left) or 1.0 (push right) representing the direction of a bang-bang control force. The output of the system is a new state vector which is then reintroduced as new inputs to the net. This continues until failure occurs or successful control is performed for the prescribed maximal period of time. The balancing time is the measure of what has been learned by each feasible net.

The individuals decoding to infeasible structures are penalised according to their deformation and undergo a higher rate of mutation in their high level, structural portion of their genome when being selected for reproduction. They thus have the chance to reproduce by changing their connectivity pattern (which may result in feasible offspring) and thus becoming stable members of the population. Exploration of new feasible structures and evolution of weights of the existing stable networks continues until a near optimised network architecture evolves or the whole population converges to a feasible network architecture.

8.12 Simulation Results

Following our general methodology for neural network design and training (Section 5.3), in this experiment we used a mixed encoding technique, where the high-level portion of the chromosome is binary-coded representing the topology of a neural net. The 1ow-level is real-valued encoding the weight-bias space in the range of (-1.0, +1.0) and crossover points are allowed to occur only between the weights. A simple bit mutation is applied on the high level and a floating point mutation is used on the low level such that a random value within ±0.1 is added to the existing active weight space rather than replacing it. The mutation rate is varied between 5 and 10% adaptively in two levels of sGA. Different GA parameters were tested in a number of trial runs of the experiment. The reported results used a population size of 80 and a two-point crossover operator with a probability of 75% along with a ranking selection scheme for reproduction. The size of connectivity matrix used is 10 by 10 along with a 1ogistic transfer function for all the nodes.

In most trial runs, the algorithm takes less than 1000 generations to evolve a net that could successfully perform the balancing task. It is found that during different runs many feasible network structures evolved, but ones which could learn quickly proliferate in the population in the later stages. When no restriction is imposed on evolving nets, most of the rapidly-learned structures are highly irregular and have direct links between inputs and the output (these are, however, fully effective controllers) [10]. Our preliminary results [9] also showed that regular structures may be evolved by modifying the evaluation function. Figure 8.7 shows two network structures which evolved in two different runs for balancing the pole at different cut-off angles. Figure 8.8 shows the displacement of the pole and the cart. The performance of the best evolved net in one typical run shown in Figure 8.9.

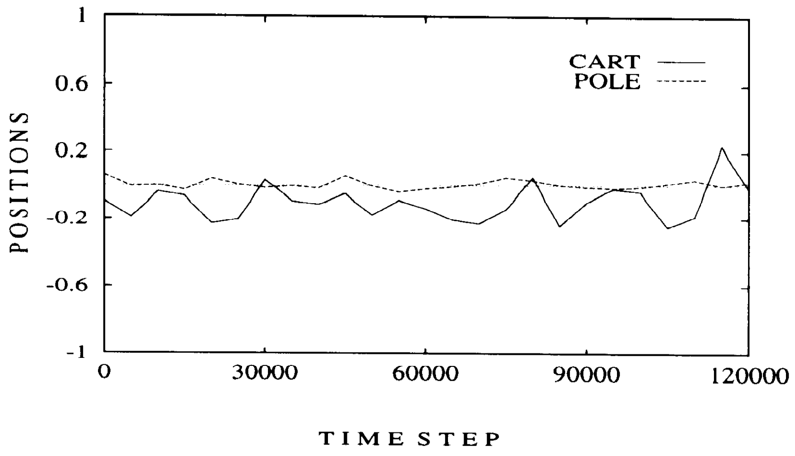


Figure 8.8: Graphs shown the position of the pole and cart with the evolved neuro-controller at different time step (in 12° case).

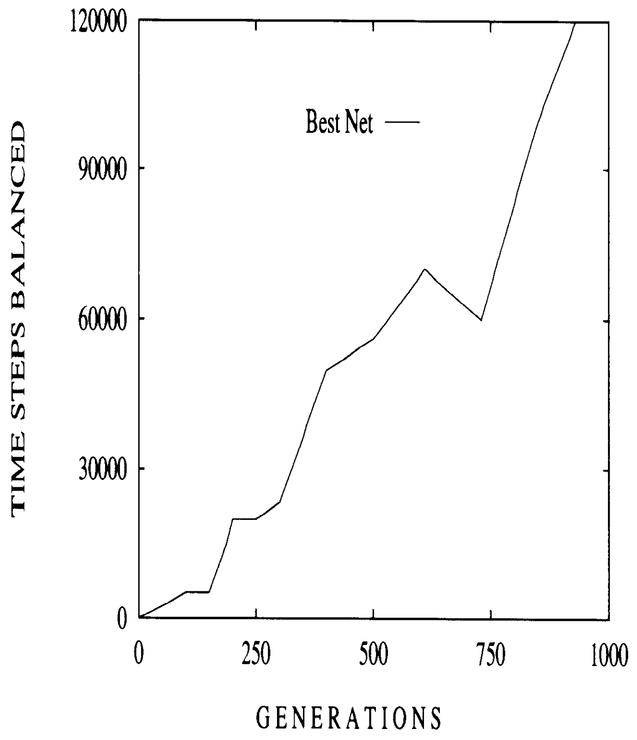


Figure 8.9: The best individual net's performance for balancing task (in 12° case).

8.13 Discussion

We applied sGA for evolving neuro-controllers which could learn a mapping between a dynamic system's state space and the space of possible actions. The significance of the sGA result is considerable. It makes possible to automatically design neuro-controller for a complex dynamic control task, by the expenditure of a relatively small amount of computational resource. The structured GA approach offers the following advantages:

1. It can evolve network structures and their weights in a single evolutionary process;
2. Each individual net can be trained using genetic reinforccment learning;
3.The method does not require partitioning of the state space of the problem;
4. No supervisory training data is required for performing the balancing task;
5.It uses global search rather than local search;
6. It can be implemented in parallel to improve the speed of convergence.

Since the results are encouraging, further work should examine generalising this evolutionary neuro-controller to enable it to operate over all possible initial input states of the system, in a similar way to that reported by [26].

References

[1] C. W. Anderson. Strategy learning with multilayer connectionist representations. In *Proceedings of the Fourth International Workshop on Machine Learning,* pages 103-114. Morgan Kaufmann, Los Altos, 1987.

[2] Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. Neuron-like adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man and Cybernetics,* Smc-13(5):834-846, Sept/Oct 1983.

[3] Hamid R. Berenji and Pratap Khedkar. Learning and tuning fuzzy logic controllers through reinforcements. *IEEE Transaction on Neural Networks,* 3(5):724-740, September 1992.

[4] Ka C. Cheok and K. Loh. A ball-balancing demonstration of optimal and disturbance-accommodating control. *IEEE Control Systems Magazine,* pages 54-57, 1987.

[5] Margaret E. Connell and Paul E. Utgoff. Learning to control a dynamic physical system. In *Proceedings AAAI-87 Sixth National Conference on Artificial Intelligence,* pages 456-460, 1987.

[6] Dipankar Dasgupta and D. R. McGregor. Designing Application-Specific Neural Networks using the Structured Genetic Algorithm. In *Proceedings of the International workshop on Combination of Genetic Algorithms and Neural Networks (COGANN-92),* pages 87-96. IEEE Computer Society Press, June 6, U.S.A 1992.

[7] Dipankar Dasgupta and D. R. McGregor. Designing Neural Networks using the Structured Genetic Algorithm. In *Proceedings of the International Conference on Artifical Neural Networks (ICANN),* pages 263-268, Brighton, U.K., 4-7 September 1992.

[8] Dipankar Dasgupta and D. R. McGregor. Nonstationary function optimization using the Structured Genetic Algorithm. In *Proceedings of Parallel Problem Solving From Nature (PPSN-2),* Brussels, 28-30 September, pages 145-154, 1992.

[9] Dipankar Dasgupta and D. R. McGregor. Evolving Neurocontrollers for Pole Balancing. In *Proceedings of the International Conference on Artificial Neural Networks (ICANN)*, pages 834-837, Amesterdam, The Netherlands, 13-16 September 1993.

[10] Dipankar Dasgupta and D. R. McGregor. Genetically Designing Neuro-controllers for a Dynamic System. In *Proceedings of the lnternational Joint Conference on Neural Networks (IJCNN)*, pages 2951-2955, Nagoya, Japan, 25-29 October 1993.

[11] Dipankar Dasgupta and Douglas R. McGregor. A More Biologically Motivated Genetic Algorithm: The Model and some Results. *To appear in Cybernatics and Systems: An International Journal,* 25(3), May 1994.

[12] David E. Goldberg. *Genetic Algorithms in Search, Optimisation and Machine Learning.* Addison-Wesley, first edition, 1989.

[13] E. Grant and Bing Zhang. A neural-net approach to supervised learning of pole balancing. In *Proceedings of IEEE International Symposium on Intelligent Control*, pages 123-129, Albany, New York, 25-26 September 1989.

[14] John H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, 1975.

[15] N. Karunanithi, D. Whitley and R. Das. Genetic Cascade Learning for Neural Networks. In *Proceedings of International Workshop on Combinations of Genetic Agorithms and Neural Networks,* pages 134-145. IEEE Computer Society Press, 1992.

[16] A. Makarovic. *A qualitative way of solving the pole balancing problem,* volume 12, chapter 16, pages 241-258. Oxford University Press, 1988.

[17] M.L. Mauldin. Maintaining diversity in genetic search. In *Proceedings of the National Conference on Artificial Intelligence,* pages 247-250, 1984.

[18] D. R. McGregor, M.O. Odeytayo, and D. Dasgupta Adaptive control of a dynamic system using genetic-based methods. In *IEEE International Symposium on Intelligent Control*, August 11-13 1992. Glasgow, U.K.

[19] D. Miche and R.A. Chambers. Boxes: An experiment in adaptive control. *Machine Intelligence,* 2:137-152, 1968.

[20] N.J. Hallman, N. Woodcock, and P. D. Picton. Fuzzy boxes as an alternative to neural networks for difficult problems. In G. Rzevski and R. A. Adey, editors, *Application of Artificial Intelligence in Engineering VI (AIENG/91)*, pages 903-919, 1991.

[21] M.O. Odetayo and D. R. McGregor. Genetic algorithm for control rules for a dynamic system. In *Proceedings of ICGA-89*, pages 177-181, 1989.

[22] Michael Omoniyi Odetayo. *On Genetic Algorithms in Machine Learning and Optimisation.* PhD thesis, Department of Computer Science, University of Strathclyde, Glasgow, U. K., December 1990.

[23] Claude Sammut. Experimental results from an evaluation of algorithms that learn to control dynamic systems. *Proceedings of the Fifth International Conference on Machine Learning*, 1988.

[24] Oliver G. Selfridge and Richard S. Sutton. Training and tracking in robotics. In *Proceedings of the Ninth International Conference on Artificial Intelligence (IJCAI)*. Morgan Kaufmann, Los Altos, 1985.

[25] Dirk Thierens and Leo Vercauteren. A topology exploiting genetic algorithm to control dynamic systems. In G. Goos and Hartmanis, editors, *Lecture Notes in Computer Science,* pages 104-108. Springer-Verlag, 1991. Proceedings of PPSN-I, 1990.

[26] D. Whitley, Stephen Dominic, and R. Das. Genetic reinforcement learning with multilayer neural networks. In *4th International Conference on Genetic Algorithms*, pages 562-569, 1991.