In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI99, Stockholm, Sweden).* Denver:Morgan Kaufmann, 1999.

Solving Non-Markovian Control Tasks with Neuroevolution

Faustino J. Gomez and **Risto Miikkulainen**
Department of Computer Sciences
The University of Texas
Austin, TX 78712, U.S.A
(inaki,risto@cs.utexas.edu)

Abstract

The success of evolutionary methods on standard control learning tasks has created a need for new benchmarks. The classic pole balancing problem is no longer difficult enough to serve as a viable yardstick for measuring the learning efficiency of these systems. The double pole case, where two poles connected to the cart must be balanced simultaneously is much more difficult, especially when velocity information is not available. In this article, we demonstrate a neuroevolution system, Enforced Sub-populations (ESP), that is used to evolve a controller for the standard double pole task and a much harder, non-Markovian version. In both cases, our results show that ESP is faster than other neuroevolution methods. In addition, we introduce an incremental method that evolves on a sequence of tasks, and utilizes a local search technique (Delta-Coding) to sustain diversity. This method enables the system to solve even more difficult versions of the task where direct evolution cannot.

1 Introduction

The pole-balancing or inverted pendulum problem has been established as a standard benchmark for artificial learning systems. For over 30 years researchers in fields ranging from control engineering to reinforcement learning have tested their systems on this task [Schaffer and Cannon, 1966; Michie and Chambers, 1968; Anderson, 1989]. There are two primary reasons for this longevity: (1) Pole balancing has intuitive appeal. It is a real-world task that is easy to understand and visualize. It can be performed manually by humans and implemented on a physical robot. (2) It embodies many essential aspects of a whole class of learning tasks that involve *temporal credit assignment* [Sutton, 1984]. In short, it is an elegant environment that is a good surrogate for more general problems.

Despite this long history, the relatively recent success of modern reinforcement learning methods on control learning tasks has rendered the basic pole balancing problem obsolete. It can now be solved so easily that it provides little or no insight about a system's ability. Neuroevolution (NE) systems (i.e. systems that evolve neural networks using genetic algorithms), for example, often find solutions in the initial random population [Moriarty and Miikkulainen, 1996; Gomez and Miikkulainen, 1997]. In response to this need for a new benchmark, the basic pole-balancing task has been extended in a variety of ways.

Wieland[1991] presented several variations to the standard single pole task that can be grouped into two categories: (1) modifications to the mechanical system itself such as adding a second pole either next to or on top of the other. (2) restricting the amount of state information that is given to the controller; for example, only providing the cart position and the pole angle. The most challenging of these is a double pole configuration where two poles of unequal length must be balanced simultaneously. Even with complete state information this problem is very difficult requiring extremely precise control to solve. In this paper, we demonstrate a neuroevolution method, Enforced Sub-populations (ESP;Gomez and Miikkulainen 1997), on an even harder version of this task in which the two poles must be balanced without velocity information. This task represents a significant leap in terms of difficulty. We show that ESP can solve this task, and can do so more efficiently than other methods have been able to solve it even *with* velocity information.

An interesting aspect of the double pole system is that it is more difficult to control as the poles assume similar lengths. When the poles are very close in length, solutions to this system cannot be evolved directly by current methods. In order to control the system under these conditions, shaping (or incremental learning) techniques can be employed that increase the length of the shorter pole very gradually [Wieland, 1991; Saravanan and Fogel, 1995]. This kind of approach is effective but can be extremely slow due to the limitations of the underlying evolutionary search method—many generations are required to recover from minute changes to the environment. Using an incremental approach in conjunction with a local search technique (Delta-Coding; Whitley et al. 1991) to sustain diversity, we demonstrate that ESP can cope with more significant changes to the environment. Instead of evolving on the goal task directly, ESP evolves on a sequence of increasingly difficult tasks.

The paper is organized as follows. Section 2 describes the ESP and Delta-Coding algorithms. In section 3 we describe incremental evolution in detail. In Section 4 we show the

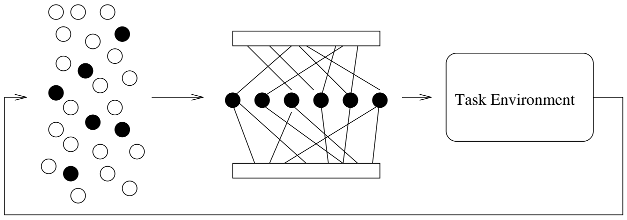

Figure 1: **Symbiotic, Adaptive Neuro-Evolution (SANE).** The population consists of hidden neurons, each with its own input and output connections. The networks are formed by randomly choosing u neurons for the hidden layer. Networks are evaluated in the task, and the fitness is distributed among all the neurons that participated in the network. After all neurons are evaluated this way, recombination is performed in the neuron population.

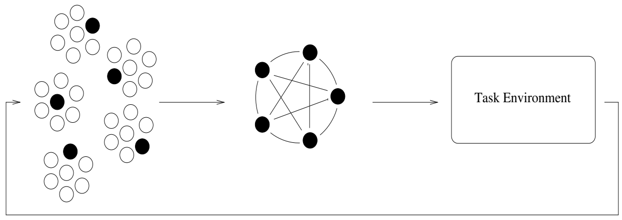

Figure 2: **The Enforced Sub-Populations Method (ESP)**. The population of neurons is segregated into sub-populations shown here as clusters of circles. The network is formed by randomly selecting one neuron from each sub-population.

results for three different tasks: (1) the double pole with velocities, (2) double pole without velocities, and (3) double pole without velocities demonstrating incremental evolution to almost equal pole lengths. The last two sections contain a discussion of the results and the conclusion.

2 Neuro-Evolution Method: *Enforced Sub-Populations + Delta-Coding.*

The Neuroevolution method used is based on Symbiotic, Adaptive Neuro-Evolution (SANE; Moriarty, 1997; Moriarty and Miikkulainen, 1996). SANE has been shown to be a powerful reinforcement learning method for tasks with sparse reinforcement.

2.1 SANE

SANE differs from other NE systems in that it evolves a population of neurons instead of complete networks (figure 1). These neurons are combined to form hidden layers of feedforward networks that are then evaluated on a given problem.

Evolution in SANE proceeds as follows:

1. Initialization. The number of hidden units u in the networks that will be formed is specified and a population of neuron chromosomes is created. Each chromosome encodes the input and output connection weights of a neuron with a random string of binary numbers.
2. Evaluation. A set of u neurons is selected randomly from the population to form a hidden layer of a feedforward network. The network is submitted to a *trial* in which it is evaluated on the task and awarded a fitness score. The score is added to the *cumulative fitness* of each neuron that participated in the network. This process is repeated until each neuron has participated in an average of e.g. 10 trials.
3. Recombination. The average fitness of each neuron is calculated by dividing its cumulative fitness by the number of trials in which it participated. Neurons are then ranked by average fitness. Each neuron in the top quartile is recombined with a higher-ranking neuron using 1-point crossover and mutation at low levels to create the offspring to replace the lowest-ranking half of the population.
4. The Evaluation–Recombination cycle is repeated until a network that performs sufficiently well in the task is found.

In SANE, neurons compete on the basis of how well, on average, the networks in which they participate perform. A high average fitness means that the neuron contributes to forming successful networks and, consequently, suggests a good ability to cooperate with other neurons. Over time, neurons will evolve that result in good networks.

The SANE approach has proven faster and more efficient than other reinforcement learning methods such as Adaptive Heuristic Critic, Q-Learning, and standard neuroevolution, in, for example, the basic pole balancing task and in the robot arm control task [Moriarty, 1997; Moriarty and Miikkulainen, 1996].

2.2 Enforced Sub-Populations (ESP)

In Enforced Sub-Populations, as in SANE, the population consists of individual neurons instead of full networks, and a subset of neurons are put together to form a complete network. However, ESP allocates a separate population for each of the u units in the network, and a neuron can only be recombined with members of its own sub-population (figure 2).

ESP speeds up SANE evolution for two reasons: The subpopulations that gradually form in SANE are already circumscribed by design in ESP. The "species" do not have to organize themselves out of a single large population, and their progressive specialization is not hindered by recombination across specializations that usually fulfill relatively orthogonal roles in the network. Second, because the networks formed by ESP always consist of a representative from each evolving specialization, a neuron is always evaluated on how well it performs its role in the context of all the other players. In SANE, networks can contain multiple members of some specializations and omit members of others, and its evaluations are therefore less consistent.

The main contribution of ESP, however, is that it allows evolution of recurrent networks. Since SANE forms networks by randomly selecting neurons from a single population, a neuron cannot rely on being combined with similar neurons in any two trials. A neuron that behaves one way in one trial may behave very differently in another, resulting in evaluations of neuron fitness that are very noisy. The sup-population architecture of ESP makes the evaluation of the neurons more consistent. A neuron's recurrent connection weight r_i will always be associated with neurons from sub-population S_i. As the sub-populations specialize, neurons evolve to expect, with increasing certainty, the kinds of neurons to which they will be connected. Therefore, the recurrent connections to those neurons can be adapted reliably.

As evolution progresses, each sub-population will decline in diversity. This is a problem, especially in incremental evolution, because a converged population cannot easily adapt to a new task. To accomplish task transfer despite convergence, ESP is combined with an iterative search technique known as Delta-Coding.

2.3 Delta-Coding

The idea of Delta-Coding [Whitley *et al.*, 1991] is to search for optimal modifications of the current best solution. In a conventional single-population GA, when the population of candidate solutions has converged, Delta-Coding is invoked by first saving the best solution and then initializing a population of new individuals called Δ-*chromosomes*. The Δ-chromosomes have the same length (number of genes) as the best solution and they consist of values (Δ-*values*) that represent differences from the best solution. The new population is evolved by selecting Δ-chromosomes, adding their Δ-values to the best solution, and evaluating the result. Those Δ-chromosomes that improve the solution are selected for reproduction. Therefore, Delta-Coding explores the hyperspace in a "neighborhood" around the best previous solution. Delta-Coding can be applied multiple times, with successive Δ-populations representing differences to the previous best solution.

In the experiments presented in this paper, Delta-Coding is implemented with the ESP sub-population architecture. Once the neuron sub-populations have reached minimal diversity, the best solution (i.e. the best network specification) is saved. New sub-populations are then initialized with Δ-chromosomes so that each neuron in the best solution has a dedicated sub-population of Δ-chromosomes that will be evolved to improve it specifically. ESP selects a Δ-chromosome from each sub-population and adds the Δ-values to the connection weights of the neurons in the best-solution. When these sub-populations converge the best Δ-chromosomes are added to the best solution to form the new best solution for the next iteration of the Delta phase.

Delta-Coding was originally developed to enhance the fine local tuning capability of Genetic Algorithms for numerical optimization by Whitley *et al.* [1991]. Gomez and Miikkulainen [1997], showed how Delta-Coding can be used to facilitate incremental evolution. When a task was completed the best solution was saved and Δ-populations initialized before evolution was begun on the next task. A more general approach has been taken in the experiments described in this paper. Delta-Coding is activated whenever the system's performance ceases to improve over a predefined number of generations. This strategy limits the disruption of genetic building blocks when the population is still adjusting well to task changes by only introducing additional variation when necessary.

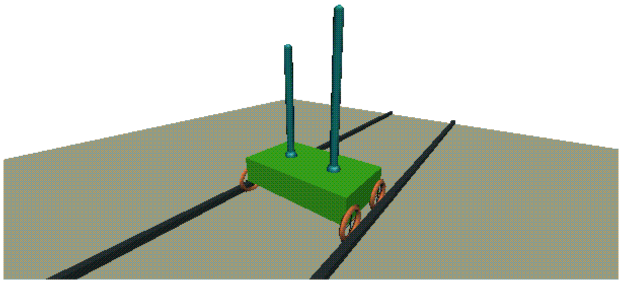

Figure 3: **The double pole system**. Snapshot of 3D real-time display available at `http://www.cs.utexas.edu/users/inaki/esp/two-pole-demo`.

3 Incremental Evolution

Evolutionary search methods can be ineffective if the task is too demanding to exert significant selective pressure on the population during the early stages of evolution. In such a case, all individuals perform poorly and the GA gets trapped in an unfruitful region of the solution space. One remedy is to enlarge the population size so that a more diverse set of phenotypes is sampled. However, prohibitively large populations may be required to discover individuals with sufficient competence to direct the search. Another approach is to view the goal task as one of many possible instances of a more general parameterized task. The system then learns by evolving on a sequence of increasingly difficult *evaluation* tasks culminating in the intended *goal* task.

A number of researchers have applied task decomposition, or shaping, to make learning complex tasks tractable [Colombetti and Dorigo, 1992; Perkins and Hayes, 1996; Singh, 1992]. Typically, in these approaches the complex task is broken into simpler components or *subtasks* that are each learned by separate systems (e.g. GAs or rule-bases) and then combined to achieve the goal task. In contrast, in incremental evolution as proposed in this paper (and also used by Wieland [1991] and Saravanan and Fogel [1995]), a single system learns a succession of tasks. Such an adaptation process is similar to continual (or lifelong) learning [Elman, 1991; Ring, 1994], and motivated by staged learning in real life.

4 Pole Balancing Experiments

The starting point for our experiments is the more challenging double pole problem in which a second pole is placed next to the first (figure 3). The objective is to apply force to the cart at regular time intervals such that the poles are balanced indefinitely and the cart stays within the track boundaries. The state of this system is defined by six state variables: the angle of each pole from vertical θ_i, the angular velocity of each pole $\dot{\theta}_i$, the position of the cart on the track x, and the velocity of the cart $\dot{x}$, where $i \in (1, 2)$ (see Wieland[1991] for the equations of motion and parameters used in this task). We adopt the notation e_ℓ to denote the evaluation-task where ℓ is the length of the short pole in meters. The long pole is always set to 1 meter. Three different experiments were conducted using this configuration with the following three goal tasks:

1. $e_{0.1}$ with velocity information.
2. $e_{0.1}$ *without* velocity information.
3. $e_{0.8}$ *without* velocity information.

All of the pole balancing experiments were implemented using the Runge-Kutta fourth-order method with a step size of 0.01s. The state variables were scaled to [-1.0, 1.0] before being input to the network. During simulation the networks output a force value every 0.02 seconds (i.e. time step) in the range [-10, 10]N. For tasks 1 and 3, the initial angle for the long pole was set to 1° (so that the networks could not control the system by simply outputting values close to zero), and fitness was determined by the number of time steps a network could keep both poles within ± 36 degrees from vertical and keep the cart between ends of a 4.8 meter track. A task was considered solved if a network could balance the poles for 100,000 time steps, which is equal to over 30 minutes in simulated time. For task 2, the fitness function and starting state are described in section 4.3. Neuron chromosomes were encoded as strings of floating point numbers. Arithmetic crossover was used to generate new neurons. Each chromosome was mutated with probability 0.2, replacing a randomly chosen weight value with a random value within the range [-6.0, 6.0]. The techniques and parameters were found effective experimentally; small deviations from them produce roughly equivalent results.

4.1 Related Work

We compared the performance of ESP with SANE and the published results of three other evolutionary methods. The first two [Wieland, 1991; Saravanan and Fogel, 1995] have been applied to the double pole problem with velocities. Wieland used an NE approach which we have termed Conventional NE. This is a single population method for evolving neural networks in which each individual represents a complete network. Fogel and Saravanan use Evolutionary Programming, a general mutation-based approach that generates offspring by perturbing the best individuals with Gaussian noise. For the case without velocities, Gruau *et al.* [1996] is the only study we know that has addressed this problem with some success. Therefore, for this task, we compare ESP only to the Cellular Encoding (CE) method. CE uses a graph transformation language to evolve the network topology as well as its weights. We did not compare ESP with conventional Reinforcement Learning methods (e.g. Q-learning, TD(λ)) in this study, because NE methods have already been shown more efficient on easier versions of these tasks [Moriarty and Miikkulainen, 1996].

4.2 2 Poles with Velocities($e_{0.1}$)

Table 1 shows the results for the 2 pole configuration with velocities. As in Wieland [1991] and Saravanan and Fogel [1995], the networks were composed of 10 hidden units. Fogel and Saravanan used feed-forward networks, and Wieland used a fully recurrent architecture. It is clear that NE methods based on partial solutions are superior to other neuroevolution methods in terms of learning speed. Although CPU time was unavailable for the other methods, it can be estimated from the number of evaluations required (i.e. Generations$\times$#Nets) that they are considerably slower. Also, ESP is faster than SANE by a factor of 2.

Method	CPU	Generations	No. Nets
Conventional NE	—	≈800	100
Ev. Programming	—	150	2048
SANE	37	63	200
ESP	22	19	200

Table 1: Comparison of results for the double pole simulations with velocities (Long pole = 1m; Short pole = 0.1m). Evolutionary Programming data is taken from Saravanan and Fogel [1995], and Conventional NE from Wieland [1991]. SANE and ESP data are average of 50 simulations.

Method	Noise	CPU	Generations	Failures
SANE	±5	38	96	45
ESP	±5	22	20	0
SANE	±10	—	—	50
ESP	±10	25	36	0

Table 2: Comparison between SANE and ESP on the double pole problem with evaluation noise. The starting angle of the long pole was chosen randomly from a uniform distribution within the specified range of degrees from vertical (Long pole = 1m; Short pole = 0.1m). Each entry is the average of 50 simulations.

To verify the robustness of SANE and ESP we also performed experiments in which the long pole was started with an angle chosen randomly from a fixed range. Table 2 shows the results for two ranges, one of ± 5 degrees and the other ± 10 degrees. This has the effect of varying the difficulty of the task from trial to trial thereby introducing noise into the fitness evaluation. Evaluation noise is a real problem in non-deterministic domains because it limits a GAs ability to determine the underlying fitness of a population. Even though SANE has been shown robust against noisy evaluations in general [Moriarty, 1997], it could not handle it in this more difficult task, most of the time failing to find a solution at all. In contrast, ESP was largely unaffected by such variation solving this task every time even when the range was extended ± 10 degrees. These results demonstrate that ESP is highly resistant to evaluation noise.

4.3 2 Poles without Velocities ($e_{0.1}$)

This task is identical to the one in the previous section except that the networks do not receive any velocity information. Therefore, the networks need to be recurrent so that the velocities can be computed internally using feedback connections. This makes the task significantly harder in two ways: (1) It is simply more difficult to control such a delicate system when the concomitant problem of velocity calculation must also be solved. (2) The number of connections in the networks is necessarily larger, thereby expanding the size of the search space.

For these simulations we compare ESP to CE using the same fitness function as Gruau *et al.*[1996]. The function $\mathcal{F}$ is the weighted sum of two separate fitness measurements

$(0.1f_1 + 0.9f_2)$ taken over a simulation of 1000 time steps:

$$f_1 = t/1000, \quad (1)$$

$$f_2 = \begin{cases} 0 & \text{if } t < 100 \\ \left(\frac{K}{\sum_{i=t-100}^{t} (|x^i| + |\dot{x}^i| + |\theta_1^i| + |\dot{\theta}_1^i|)} \right) & \text{otherwise,} \end{cases} \quad (2)$$

where t is the number of time steps the pole was balanced, K is a constant (set to 0.75), and the denominator in (2) is the sum of the absolute values of the cart and the large pole state variables, summed over the last 100 time steps of the run. This complex fitness is intended to force the network to compute the pole velocities by penalizing swinging, and thereby making the GA favor controllers with the ability to return the poles to the upright position and damp oscillations. This kind of fitness measure is necessary because otherwise networks can balance the poles by merely swinging them back and forth (i.e. without calculating the velocities) [Gruau *et al.*, 1996].

Gruau *et al.* claimed that the decidedly large number of evaluations required by CE to solve this task, compared to direct encoding (table 3), is offset by the number of evaluations saved by not having to search for an effective network architecture to solve the problem. Also, they were unable to solve this problem using direct encoding. CE does not assume an a priori network topology, and is therefore able to optimize it to suit a particular problem. While such methods are an important area of research, we found that they are not necessary nor advantageous for this problem. To demonstrate this we designed an experiment that minimizes the amount of human intervention in the determination of network topology that ESP evolves. In this experiment, the number of hidden units H is still fixed for each evolution, but instead of being prescribed by the user it is chosen at random by the system in a range from [1-10]. For each simulation, the system begins evolving with the randomly selected H. If it does not solve the task (for whatever reason), it will restart with a new H. This occurs repeatedly until the task is solved. The total number of evaluations over all of the restarts is then counted.

Table 3 compares the performance of ESP and CE in this task. The ESP experiments are the aforementioned where H is selected randomly. The initial angle for the long pole was set to $4.5°$ for all simulations. To determine if the task had been solved, we tested the most fit individual from each generation to see if it could balance the pole for 100,000 time steps and score at least 200 on the generalization test describe below. The latter was necessary because we found that fitness $\mathcal{F}$ did not correlate well with the ability to generalize to novel initial states.

In addition to learning speed, the robustness of the solutions was also tested. The column labeled "Generalization" refers to each method's average score on a test where a successful controller is awarded a point for each of 625 different initial states from which it is able to control the system for 1000 time steps. The test cases were generated by allowing the state variables $x, \dot{x}, \theta_1$, and $\dot{\theta}_1$ to take on the values: -0.9, -0.5, 0.0, 0.5, 0.9, ($5^4 = 625$) . This test, first introduced in [Dominic *et al.*, 1991], has become a standard for evaluating

Method	Evaluations	Generalization	No. Nets
CE	840,000	300	16,384
ESP	169,466	289	1,000

Table 3: Results for double pole without velocities. Long pole = 1m; Short pole = 0.1m. Average of 20 simulations. Results for CE taken from Gruau *et al.* [1996].

the generality of solutions in pole balancing. A high score indicates that a solution has competence in a wide area of the state space.

The main result is that ESP is roughly 5 times faster than CE without significantly compromising generalization, showing that the search for an appropriate architecture for this task can be automated by a simple stochastic mechanism. The ESP simulations had a restart rate of 4.06. That means that, on average, the system had to start over with a new random H about 4 times per run.

4.4 Incremental Evolution of 2 Poles without Velocities ($e_{0.8}$)

This section compares the results of incremental versus direct evolution. For the incremental experiments, the following method was used the determine the sequence of tasks: The evolution begins with the pole balancing system described in the previous section $e_{0.1}$ as the initial evaluation task t_1. When this task is solved, the shorter pole is lengthened by a predefined increment (P). P can change from task to task according to a simple rule. If ESP is unable to solve the next task after two Delta phases, P is halved and ESP then tries to solve the problem where the shorter pole has a length halfway between t_1 and the new, "unachieved" task t_2. Once this intermediate task t_1' is solved, ESP will move on to the next task t_2. So instead going from t_1 to t_2 in a single step, the system does it in two steps $t_1 \rightarrow t_1' \rightarrow t_2$. If after completing t_1', task t_2 can still not be achieved, P is halved again and added to t_1' yielding t_1'' which is halfway between t_1' and t_2. Tasks are therefore repeatedly simplified, with P decreasing monotonically, until either a transition occurs or a lower bound on P is reached.

The incremental evolution can be illustrated best with an example. Normally with this method the task differences are quite large at first. As the networks move on to harder tasks, P tends to shrink, and more task transitions are required for a given increase in the length of the short pole. For the initial value of P used in these experiments (0.1), a typical evolution schedule might look like:

$$e_{0.1} \rightarrow e_{0.2} \rightarrow e_{0.25} \rightarrow e_{0.3} \rightarrow e_{0.325} \rightarrow e_{0.35} \rightarrow \ldots \rightarrow e_{0.8}$$

In each of the direct evolution simulations, the evaluation task was fixed. Four different tasks were chosen to test this approach: $e_{0.3}, e_{0.5}, e_{0.7}, e_{0.8}$. Each task was attempted 50 times evaluating 1000 networks per generation. Note that the direct simulations were given a population size over two times larger than that of the incremental (1000 vs.400). This was done to see if the harder tasks could be solved by simply increasing the number of search points.

Table 4 compares the two methods on different tasks. P was started with a value of 0.1. This means that after solv-

Pole Length	Direct	Incremental
0.3	17	100
0.5	0	100
0.7	0	98
0.8	0	80

Table 4: Incremental vs. Direct evolution on the double pole problem without velocities. The table shows the percentage of simulations that were able to achieve each task for the two approaches. The tasks are denoted by the length of the shorter pole (first column).

ing the initial task $e_{0.1}$ the short pole will be increased by 100% to $e_{0.2}$. This is a significant change to the environment. Other approaches that have applied shaping to the easier double pole task (with velocities) have incremented the short pole by only 1% [Wieland, 1991; Saravanan and Fogel, 1995]. ESP was almost always able to complete the first three tasks ($e_{0.1}, e_{0.2}, e_{0.3}$) without having to decrement P, and the first two tasks were always achieved in less than 70 generations.

Even with a larger population, Direct evolution was incapable of solving $e_{0.5}$. When the task is this hard ESP cannot discover a good region of the search space before converging. No individual does well enough to guide the search. ESP selects genotypes that are slightly better than others in terms of the fitness scalar but are not necessarily any closer to the goal. The incremental approach, on the other hand, was able to solve the hardest task ($e_{0.8}$) within 3000 generations 80% of the time, making an average of 30 task transitions per run.

5 Conclusions

The results show that ESP with Delta-Coding can be an efficient method for controlling unstable systems. It was able to solve a Markovian version of the double pole balancing problem much faster than other methods, and also a much more difficult non-Markovian version. Incremental evolution was found to be an effective way to scale up the approach to even more difficult tasks. The non-Markovian control task is an important benchmark not only because is it presents nonlinear dynamical environment, but also because it requires memory. Many tasks in varied domains from game-playing to robotics require memory to overcome perceptual aliasing.

In the future, we plan to apply ESP to real-world robot navigation tasks. Tasks of this kind are often naturally decomposable into a hierarchy of subtasks amenable to the incremental paradigm. They also pose the challenge of changing environments, which we believe can also be solved effectively by the same approach.

Acknowledgments

Special thanks to Oliver Gomez for help in preparing the illustrations. This research was supported in part by National Science Foundation under grant #IRI-9504317.

References

[Anderson, 1989] Charles W. Anderson. Learning to control an inverted pendulum using neural networks. *IEEE Control Systems Magazine*, 9:31–37, April 1989.

[Colombetti and Dorigo, 1992] Marco Colombetti and Marco Dorigo. Robot shaping: Developing situated agents through learning. Technical Report TR-92-040, International Computer Science Institute, Berkeley, CA, 1992.

[Dominic *et al.*, 1991] S. Dominic, R. Das, D. Whitley, and C. Anderson. Genetic reinforcement learning for neural networks. In *Proceedings of the International Joint Conference on Neural Networks* (Seattle, WA), volume II, pages 71–76, Piscataway, NJ, 1991. IEEE.

[Elman, 1991] Jeffrey L. Elman. Incremental learning, or The importance of starting small. In *Proceedings of the 13th Annual Conference of the Cognitive Science Society*, pages 443–448, Hillsdale, NJ, 1991. Erlbaum.

[Gomez and Miikkulainen, 1997] Faustino Gomez and Risto Miikkulainen. Incremental evolution of complex general behavior. *Adaptive Behavior*, 5:317–342, 1997.

[Gruau *et al.*, 1996] Frederic Gruau, Darrell Whitley, and Larry Pyeatt. A comparison between cellular encoding and direct encoding for genetic neural networks. Technical Report NC-TR-96-048, NeuroCOLT, 1996.

[Michie and Chambers, 1968] Donald Michie and R. A. Chambers. BOXES: An experiment in adaptive control. In E. Dale and D. Michie, editors, *Machine Intelligence*. Oliver and Boyd, Edinburgh, UK, 1968.

[Moriarty and Miikkulainen, 1996] David E. Moriarty and Risto Miikkulainen. Efficient reinforcement learning through symbiotic evolution. *Machine Learning*, 22:11–32, 1996.

[Moriarty, 1997] David E. Moriarty. *Symbiotic Evolution of Neural Networks in Sequential Decision Tasks*. PhD thesis, Department of Computer Sciences, The University of Texas at Austin, 1997. Technical Report UT-AI97-257.

[Perkins and Hayes, 1996] Simon Perkins and Gillian Hayes. Robot shaping–principles, methods, and architectures. Technical Report 795, Department of Artifical Intelligence, University of Edinburgh, 1996.

[Ring, 1994] Mark B. Ring. *Continual Learning in Reinforcement Environments*. PhD thesis, Department of Computer Sciences, The University of Texas at Austin, Austin, Texas 78712, August 1994.

[Saravanan and Fogel, 1995] N. Saravanan and David B. Fogel. Evolving neural control systems. *IEEE Expert*, pages 23–27, June 1995.

[Schaffer and Cannon, 1966] J Schaffer and R Cannon. On the control of unstable mechanincal systems. In *Automatic and Remote Control III: Proceedings of the Third Congress of the International Federation of Automatic Control*, 1966.

[Singh, 1992] S. Singh. Transfer of learning by composing solutions of elemental sequential tasks. *Machine Learning*, 8:323–339, 1992.

[Sutton, 1984] Richard Sutton. *Temporal Credit Assignment in Reinforcement Learning*. PhD thesis, University of Massachusetts, Amherst, MA, 1984.

[Whitley *et al.*, 1991] D. Whitley, K. Mathias, and P. Fitzhorn. Delta-coding: An iterative search strategy for genetic algortihms. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, Los Altos, CA, 1991. Morgan Kaufmann.

[Wieland, 1991] Alexis Wieland. Evolving neural network controllers for unstable systems. In *Proceedings of the International Joint Conference on Neural Networks* (Seattle, WA), volume II, pages 667–673, Piscataway, NJ, 1991. IEEE.