To appear in *Adaptive Behavior*, 5:317-342, 1997

Incremental Evolution of Complex General Behavior

Faustino Gomez
Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712
inaki@cs.utexas.edu

Risto Miikkulainen
Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712
risto@cs.utexas.edu

June 1, 1996

Abstract

Several researchers have demonstrated how complex action sequences can be learned through neuro-evolution (i.e. evolving neural networks with genetic algorithms). However, complex general behavior such as evading predators or avoiding obstacles, which is not tied to specific environments, turns out to be very difficult to evolve. Often the system discovers mechanical strategies (such as moving back and forth) that help the agent cope, but are not very effective, do not appear believable and would not generalize to new environments. The problem is that a general strategy is too difficult for the evolution system to discover directly. This paper proposes an approach where such complex general behavior is learned incrementally, by starting with simpler behavior and gradually making the task more challenging and general. The task transitions are implemented through successive stages of delta-coding (i.e. evolving modifications), which allows even converged populations to adapt to the new task. The method is tested in the stochastic, dynamic task of prey capture, and compared with direct evolution. The incremental approach evolves more effective and more general behavior, and should also scale up to harder tasks.

1 Introduction

Neuro-Evolution (NE), the training of Artificial Neural Networks with Genetic Algorithms, has proven an attractive alternative to standard learning methods in reinforcement environments (Nolfi et al. 1990; Colombetti and Dorigo 1992a; Yamauchi and Beer 1994; Nolfi and Parisi 1995; ?). In NE, strings of values (chromosomes) representing neural network parameters (e.g. connection weights, thresholds, and connectivity) are recombined based on the principle of natural selection in order to find an optimal network for a given problem. Many NE systems consist of a mobile agent in a static environment where the agent must learn a policy that minimizes the travel distance between certain desirable states (Cliff et al. 1992; Law and Miikkulainen 1994; Nolfi and Parisi 1994). These "goal states" are defined spatially by unique coordinates so that it is easy to assess how well the individual is doing by comparing the known optimal path to the actual path. As tasks require increasingly complex behavior, this type of metric becomes difficult to define. Wall-following, obstacle-avoidance, and predator-prey scenarios are examples of such tasks because they describe a more general, abstract behavior not tied to specific spatial locations.

A critical problem in trying to evolve complex behaviors is that the strategies that emerge are often "mechanical." These strategies yield reasonable performance in terms of maximizing a fitness measure, but do not exhibit the kind of intelligent responsiveness to the environment that is required to ultimately

accomplish the objective. Seeming competence in the task is often only a side-effect of the "mechanical" strategy; the agent does not learn the general task, but instead learns a relatively primitive environment-specific behavior. For example, a predator being evolved to capture a prey may exhibit a simple strategy of moving back and forth or in a circle through the environment. Because this behavior is easy to evolve and guarantees better fitness than moving very little, the population can easily be lured into a region of the solution space that manifests this strategy. Once in this pathological region, it is unlikely that a good solution will be found since the non-mechanical, general behavior required to accomplish the task can be very different from the mechanical behavior.

Complex behaviors are difficult to evolve primarily because usually a reactive response to the current state is not enough; such behaviors require memory–that is, an ability to predict future states based on information about previous states. For example in prey capture, when the prey goes outside the sensory range, the agent must remember where it last saw the prey and where it is likely to be in the future. In terms of neuro-evolution it means that the neural network architectures evolved must be recurrent, which makes them harder to deal with.

This paper presents a method for evolving complex general behavior incrementally. Instead of evaluating a population on the same task throughout the course of evolution, it is first evaluated on a relatively easy task and then on increasingly harder tasks, in an effort to build towards the goal behavior. There are two techniques that allow us to do this: (1) Enforced Sub-Populations makes it possible to evolve neural networks that have memory, and (2) Delta-Coding makes it possible to adapt to each new task. The results of incremental evolution presented in this article show that it is possible to achieve complex general behavior with evolutionary neural networks.

2 Prey Capture

Prey Capture, the task used in this paper to demonstrate incremental evolution, is a special case of a class of problems known as Pursuit and Evasion. Pursuit and Evasion contests consist of an environment containing a minimum of two entities: a predator and a prey. The predator moves through the environment trying to capture the prey while the prey attempts to avoid capture by fleeing from the predator. Scenarios of this kind are a particularly interesting arena for the study of adaptive behavior because they are ubiquitous in natural ecosystems and offer a relatively simple objective that requires complex sensory-motor coordination with respect to both the environment and another moving entity (Miller and Cliff 1994). To accomplish the Prey Capture task, an *agent*, moving through a simulated environment, must be able to apprehend another entity, the *prey*, within a fixed number of time-steps. The agent can detect the prey only within a limited distance. If the prey moves out of sensor range the agent must remember where it was last detected. Prey Capture is simple in both description and implementation, yet demands a level of behavioral sophistication that is susceptible to the emergence of mechanical strategies.

2.1 Environment

The environment consists of a square spatially and temporally discrete grid-world (figure 1a). Both the agent and the prey occupy a single grid space and can move in one of four directions {N,S,E,W} at each time-step. The agent is considered to have captured the prey when they both occupy the same grid-cell. The prey moves probabilistically with a tendency to move away from the agent that grows stronger as the agent gets closer to it (see Appendix A for a definition of the enemy algorithm, due to Lin (1992)). The prey moves at a speed that is set between 0 and 1. This value is the probability of the prey taking an action each time-step. If the prey has a speed of 0.5 will do nothing 50% of the time. Note that if the environment were continuous, a speed of 0.5 would make the task quite easy because the prey would always be moving at the same leisurely rate. In the discrete world of Prey Capture, however, a prey moving at a speed of 0.5

is really moving at the same speed as the agent but only part of the time.

This environment provides a non-trivial task with a relatively high complexity. The agent must learn to move into the grid-cell occupied by the prey by mapping its sensory input to appropriate actions. When the prey moves outside the sensory range, the agent no longer receives direct sensory stimulus from the prey and must rely on short-term memory to decide which action will bring the prey closer and ultimately into sensor range.

More formally, the environment can be described by the *sensory-state machine* (SSM)(Wilson 1991):

$$\{E(t+1)\} = f(E(t), A(t)), \tag{1}$$

where $E(t)$ and $A(t)$ are the agent's sensory stimulus and action at time t, respectively, and f is the mapping that describes the environmental dynamics from the agent's perspective. For a given sensory stimulus (sense vector) $E(t)$ and an action $A(t)$ there is a finite set of sensory stimuli $\{E(t+1)\}$ which *may* result. If $cardinality(\{E(t+1)\}) > 1$ for any pair $(E(t), A(t))$, then the environment is non-deterministic. When the sense vector $E(t)$ does not denote a unique global state, that is, the agent cannot sense the entire environment with total accuracy, some form of memory is required to help the agent predict where the prey is located. In other words, the uncertainty about $E(t+1)$ can be reduced by keeping track of previous sense and action vectors:

$$\{E(t+1)\} = f_2(E(t), A(t), E(t-1), A(t-1)...E(t-k), A(t-k)), \tag{2}$$

If the environment is deterministic and contains a finite number of states, then there exists some k for which $cardinality(\{E(t+1)\}) = 1$ for all possible pairs $(E(t), A(t))$ and the agent can thereby reliably predict $\{E(t+1)\}$. The Prey Capture environment presented here is particularly difficult because the prey's actions are partly random: there does not exist a k for which the non-determinism in (2) is eliminated. The agent can never learn to predict $E(t+1)$ exactly, although it can reduce the uncertainty with memory.

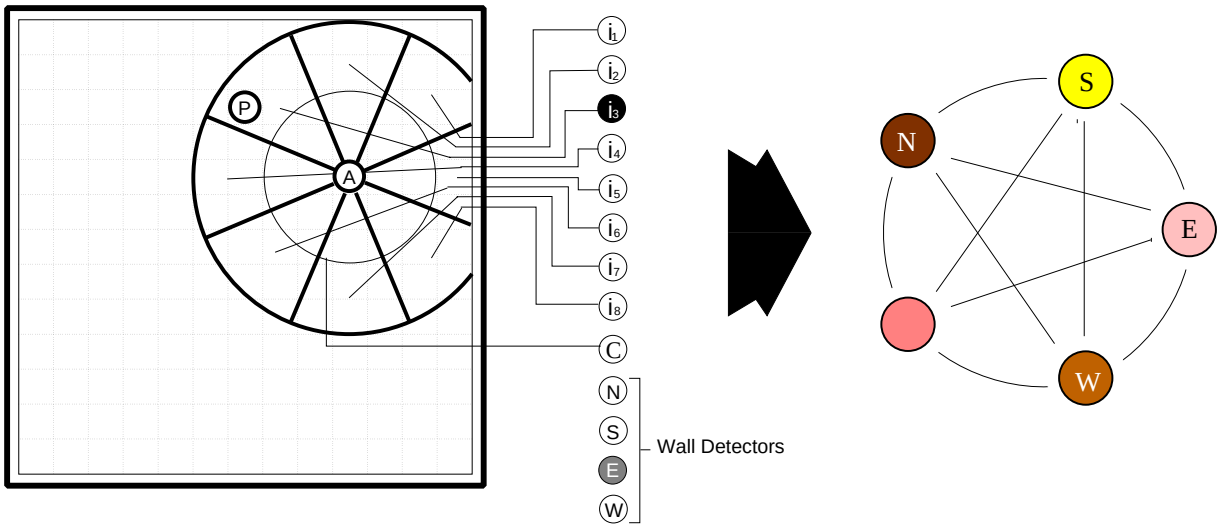
2.2 Agent Representation

The agent is controlled by a fully connected recurrent neural network with sigmoidal units (Figure 1b). At each time step each unit receives input from the input layer and from all other units. Such recurrency allows the agent to maintain temporal information that is necessary for performing the task.

As the agent moves through the environment it can detect the presence of the prey within a specified sensor range (figure 1a). There is one input unit (i_i) assigned to each of the 8 sectors in the sensory array. When the prey is in an area covered by the sensory array, the unit corresponding to that sector is set to 1. An additional unit (C) is set when the prey is within the closer half of the sector. The units i_1 through i_8 and C therefore afford a coarse encoding of relative enemy position. The radial nature of the sensory apparatus gives greater sensitivity to prey movement at close range, where it is most crucial. Four more units are used to detect the walls in the N, S, E and W directions. As a wall comes within sensor range the corresponding unit is activated to a degree that is proportional to the wall's distance from the agent. There is one output unit for each of the four possible actions. At each time step the agent selects the action corresponding to the unit with the highest activation. This representation provides the agent with sensory input that is both imprecise and of limited range. In order to perform well in Prey Capture, an agent must learn to associate sensory input with the appropriate pursuit actions and exploit its short-term memory (recurrent connections) to remember where it last saw the prey if it moves out of sensor range.

3 Incremental Evolution

By attacking a complex general task such as Prey Capture "head-on," from the outset of the evolution process, conventional evolutionary methods suffer from inefficient performance. The task can be too demanding



(a) Environment (b) Input layer (c) Recurrent network

Figure 1: **The Prey Capture environment and the agent network.** (a) The grid world occupied by the agent and the prey. The sectors and circles around the agent represent its sensory array. There are 8 sectors divided into two levels of proximity. Each sector is represented by a node in the input of the neural network controlling the agent, as shown in (b). An input unit i_i is activated when the prey enters the corresponding sector. If the agent brings the prey within the inner circle of the array, the C unit will also be activated. Each of the wall detector units is activated proportional to the agent's distance to the wall in that direction, provided the wall is within the sensor range. In the situation shown here, the input unit i_3 is activated but C is not, because an enemy is in the far NW area. The E wall-detector unit is also activated by a small amount because the east wall is just within sensor range. This input is fed into the fully recurrent neural network along with the network's activation from the previous time-step. In this case, the agent will move north because the north (N) output unit has the highest activation.

to exert significant selective pressure on the population during the early stages of evolution. All of the individuals perform poorly and therefore the GA gets trapped in an unfruitful region of the solution space. If a population is first evolved on an *easier* version t' of the complex task t, it may be possible to discover a region of the solution space from which t is more accessible. If this is not the case, it may be possible to evolve a population on an even easier version t'' from which t' is more accessible, and so on. In this way, the ultimate task can be achieved by evolving incrementally on a sequence of tasks starting with a task that can be evolved directly.

In order to formulate a scheme for incremental evolution, it is necessary to make a distinction between the *evaluation-task*, which is used to evaluate the Agent's fitness for reproduction, and the *goal-task*, which the agent is ultimately evolved to perform (Nolfi and Parisi 1994; *task* connotes both the environment and the behavioral objective). The goal-task can then be seen as the culmination of a series of progressively more demanding evaluation-tasks: $\{t_1, t_2, t_3, \ldots, t_n\}$ where n is the total number of evaluation-tasks and t_n is the goal-task. This set of tasks is ordered so that t_i is easier than t_{i+1} for all i: $0 < i \leq n$. The number of tasks required (n) must be determined experimentally.

Each evaluation-task is derived by transforming a goal-task specification to one where the appropriate behavior is more readily "evolvable." A task transformation must preserve the underlying structure of the environment as well as the overall goal-task objective. The following two heuristics can be applied to derive effective evaluation-tasks: (1) Increase the "density" of the relevant experiences within a trial so that a

network can be evaluated based on greater information in a shorter amount of time; and (2) Make the evaluation-task easier so that the acquisition of fundamental goal-task skills is more feasible. A combination of these two heuristics is used in the evolution of Prey Capture. The evaluation-task environments are designed to temporally concentrate experience and to facilitate skill acquisition.

Once the set of evaluation-tasks has been derived, evolution of the goal-task behavior can proceed by evolving on each evaluation-task in succession. First, t_1 is evolved until a satisfactory level of performance is achieved. After t_1 is completed, t_2 is instantiated and evolution resumes. This process continues until the goal-task is completed.

A number of researchers have applied task decomposition, or shaping, to make learning complex tasks tractable (Colombetti and Dorigo 1992b; Lin 1993; 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 (1990, 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; Thrun 1996), and motivated by learning in real life. If, for instance, the goal-task is that of driving a Formula-1 race car at Grand Prix level, we can imagine a lattice of tasks arranged in order of increasing difficulty leading up to the goal-task. Tasks are arranged in a lattice because there may be many epistemic paths to accomplishing the goal-task. For this particular case, tasks might include: learning to drive a car safely, learning to drive fast, competing against novice drivers, competing in stock cars, and competing in rallies. One would not endeavor to drive a Formula-1 car without first having gained competence in other forms of racing and, more importantly, learning how to drive a car in the first place. In the experiments that follow, this general learning principle is exploited in evolution by having an Agent compete against an environment that becomes more challenging in response to its growing expertise.

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

The Neuro-Evolution method used is similar to Symbiotic, Adaptive Neuro-Evolution (SANE; Moriarty 1997; Moriarty and Miikkulainen 1996a, 1996b). SANE has been shown to be a very powerful reinforcement learning method for tasks with sparse reinforcement. In order to apply this method to evolving complex general behavior, two extensions are necessary: Enforced Sub-Populations to build recurrent networks, and Delta-Coding to assist in transferring to new tasks.

4.1 SANE

SANE differs from other NE systems in that it evolves a population of neurons instead of complete networks (figure 2). These neurons are combined to form hidden layers of feed-forward networks that are then evaluated on a given problem.

Evolution in SANE proceeds in the following sequence of steps:

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.

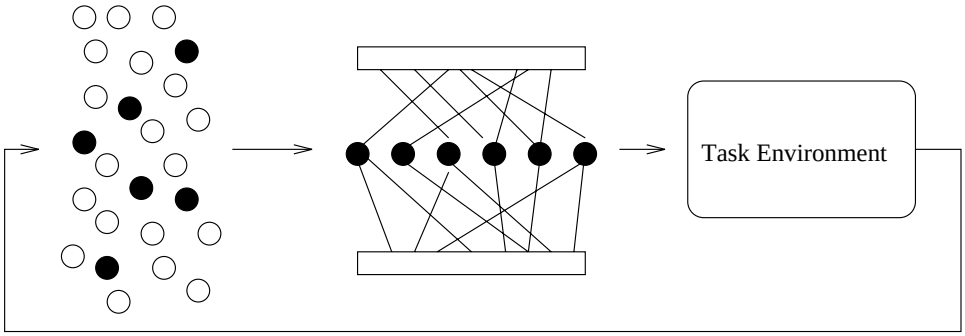


Figure 2: **Symbiotic, Adaptive Neuro-Evolution (SANE).** The population consists of hidden neurons, each with its own input and output connections. The networks are formed by randomly choosing u neurons for the hidden layer. Networks are evaluated in the task, and the fitness is distributed among all the neurons that participated in the network. After all neurons are evaluated this way, recombination is performed in the neuron population.

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. Goto 2.

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 (Moriarty and Miikkulainen 1996a; Moriarty 1997). The reason is that evolving partial solutions (neurons) instead of full solutions (networks) automatically maintains diversity in the population. If one type of neuron genotype begins to take over the population, networks will often be formed that contain several copies of that genotype. Because difficult tasks usually require several different hidden neurons, such networks cannot perform well. They incur low fitness, and the dominant genotype will be selected against, bringing diversity back into the population. As a matter of fact, in the advanced stages of SANE evolution, instead of converging the population around a single individual like the standard GA approaches, the neuron population clusters into "species" or groups of individuals that perform specialized functions in the target behavior (Moriarty 1997).

4.2 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 3).

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

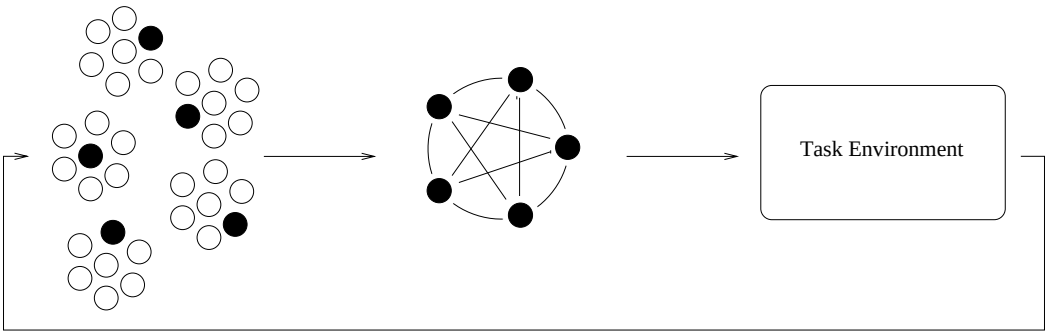


Figure 3: **The Enforced Sub-Populations Method (ESP)**. The population of neurons is segregated into sub-populations shown here as clusters of circles. The network is formed by randomly selecting one neuron from each subpopulation.

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. A neuron's behavior in a recurrent network is critically dependent upon the neurons to which it is connected. 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, and SANE cannot obtain accurate information about the fitness of recurrent neurons. The sup-population architecture of ESP makes the evaluation of the neuron more consistent. A neuron's recurrent connection weight r_i will always be associated with neurons from subpopulation 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.

4.3 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 hyper-space 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

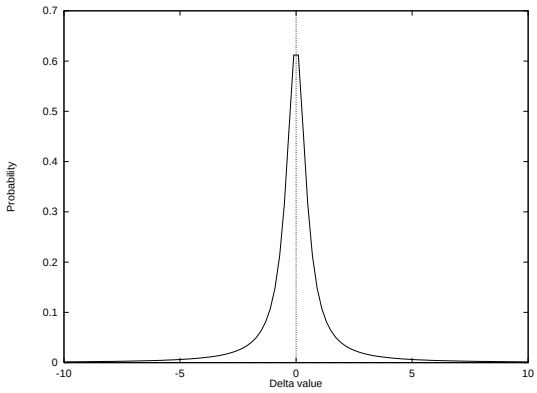


Figure 4: **The** *Cauchy distribution* **for** $\alpha = 0.5$. Most of the Δ-values represent small modifications to the best solution, but large values are also possible.

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 developed by Whitley et al. (1991) to enhance the fine local tuning capability of Genetic Algorithms for numerical optimization. However, its potential for adaptive behavior lies in the facilitation of task transfer. Delta-Coding provides a mechanism for transitioning the evolution into each progressively more demanding task:

$$t_1 \xrightarrow{\Delta} t_2 \xrightarrow{\Delta} t_3 \xrightarrow{\Delta} \cdots \xrightarrow{\Delta} t_n \tag{3}$$

Each $t_i \xrightarrow{\Delta} t_{i+1}$ (Delta transition) involves saving the best network $best(t_i)$ from the last generation of the evaluation task t_i, and initializing sub-populations of Δ-chromosomes that are then evolved to adapt $best(t_i)$ to the next task t_{i+1}. When t_{i+1} has been evolved, $best(t_{i+1})$ (the best modification to $best(t_i)$) is formed by adding the best Δ-chromosomes to $best(t_i)$. The solution $best(t_{i+1})$ will then be adapted to t_{i+2}, and so on.

In some task transitions, some weights in the best solution may need to change radically. This is especially true if the new task introduces novel input patterns. Therefore, initial Δ-values need to be generated using a probability density function that concentrates most of the values in the local search space while occasionally permitting values of much larger magnitude. One distribution that has these desirable properties is the *Cauchy* distribution (figure 4):

$$f(x) = \frac{\alpha}{\pi(\alpha^2 + x^2)} \tag{4}$$

With this distribution 50% of the Δ-values will fall within the interval $\pm\alpha$ and 99.9% within the interval $318.3 \pm \alpha$.

5 Prey Capture Performance

The effectiveness of incremental evolution based on ESP and Delta-Coding was tested in the Prey Capture task and compared to direct evolution. To determine how difficult tasks could be solved, the prey speed and short-term memory requirements of the task were varied.

5.1 Simulation Setup

For all simulations the following parameter settings were used:

Sup-population size← 40
Sensor range← 5
Chromosome mutation probability← 0.1
Initial random weight range← [-6.0,6.0]
Units← 5
α (of the Cauchy distribution)← 0.3

The networks were fully connected, and neuron chromosomes were encoded as strings of floating point numbers. Arithmetic crossover was used to generate new neurons. Each chromosome was mutated with probability 0.1, 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.

At each generation during evolution, 400 networks are constructed and evaluated in the following way: the agent is placed in the center of a 24×24 grid world and the prey is placed in a random position just within the agent's sensor range. The agent and prey alternate in taking an action each time-step until either the prey has been captured or a maximum number of time-steps N has been reached. If the agent captures the prey then the prey is moved to a new initial position just within the sensor range, and the agent is allowed another N moves to capture the prey. This process continues until the agent fails to capture the prey within N moves (the value N can thus be interpreted e.g. as the maximum time that the agent can survive without feeding). The number of times the agent captured the prey is used as its fitness score. For an agent to receive high fitness, it must be able to catch the prey from many initial states and deal favorably with the prey's non-deterministic behavior.

The difficulty of the task can be adjusted in two ways: To force the network to follow the prey tighter and predict its behavior more accurately, the prey speed (i.e. the probability of it making a move) can be increased. Second, to enforce believable behavior based on short-term memory, the prey can be allowed to make a greater number of moves before the agent is allowed to make its first move. During these m moves the prey moves with a speed of 1.0. The prey's head start guarantees that each trial will contain situations that require memory, thereby maximizing the density of task-relevant experiences per trial.

Evaluation-tasks will be referred to by the notation E_m^s, where m is the number of initial moves the prey can take and s is the prey's speed. agents are evolved both directly and incrementally to accomplish $E_4^{1.0}$ (i.where the prey makes four initial moves before the agent is allowed to move, and then continues to move at the same speed as the agent). An agent is considered to have accomplished the task if it can capture the prey more than 100 times in single trial.

5.2 Direct Evolution

For the Direct evolution simulations, the evaluation-task remains constant throughout evolution. In other words, the networks are subjected to the goal task $E_4^{1.0}$ from the beginning.

The lower plot in figure 5 shows the results for direct evolution. As can be clearly seen in this figure, direct evolution is not powerful enough to solve $E_4^{1.0}$. All of the networks in the first generation perform too poorly to provide adequate differentiation for reproduction; the environment is simply too difficult for any single individual to perform significantly above average. The networks improve slightly over the first 20 generations but, invariably, become trapped in a region of the weight space where the sub-populations have converged before basic task skills have been acquired. The best of these individuals move around the environment a few times in a mechanical fashion. In order for an individual to perform well it must know both how to chase a fast-moving prey and remember its location. The likelihood of encountering an individual with such proficiency in a random population is extremely low and the direct evolution failed every single time in our simulations.

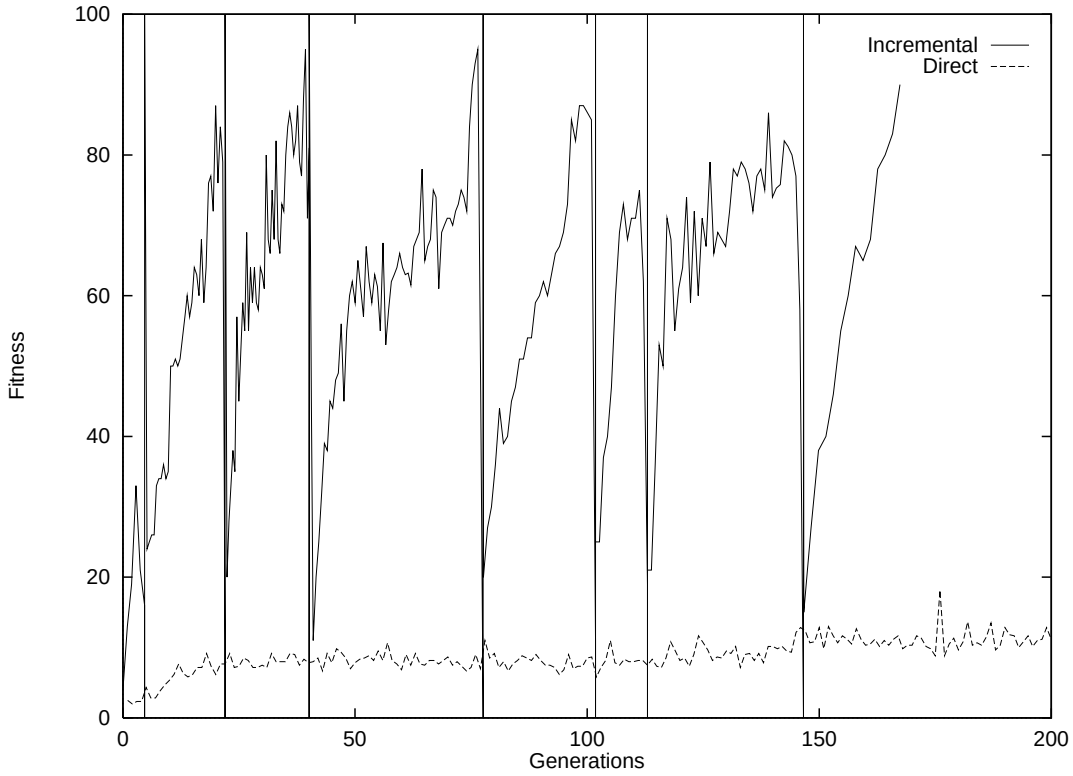


Figure 5: **Performance of direct and incremental evolution in the Prey Capture task**. The maximum fitness per generation is plotted for each of the two approaches. The direct evolution (bottom plot) makes slight progress at first but stalls after about 20 generations. The plot is an average of 10 simulations. Incremental evolution, however, proceeds through several task transitions (seen as abrupt drop-offs in the plot), and eventually solves the goal-task. The incremental plot is an average of 5 simulations. Each of these included a different number of generations for each evaluation-task, so time was stretched or shrank for each so that the transitions could be lined up.

5.3 Incremental Evolution

In incremental evolution the population is first evolved on the task $E_0^{0.0}$, i.e. capturing a stationary pray within its sensory range. Once this initial task has been accomplished, the best-performing network is saved and Delta-Coding invoked to evolve $E_2^{0.0}$. After $E_2^{0.0}$, the number of initial steps m is further increased to 3 and 4, and then the prey speed from 0.0 to 1.0 in four steps. In other words, the incremental evolution schedule is:

$$E_0^{0.0} \xrightarrow{\Delta} E_2^{0.0} \xrightarrow{\Delta} E_3^{0.0} \xrightarrow{\Delta} E_4^{0.0} \xrightarrow{\Delta} E_4^{0.3} \xrightarrow{\Delta} E_4^{0.6} \xrightarrow{\Delta} E_4^{0.8} \xrightarrow{\Delta} E_4^{1.0}$$

This sequence of tasks forces the agent to first develop its short-term memory and then learn to deal with a fast moving prey. While other sequences are possible, this is a natural one. To be able to pursue a pray at all, the agent first has to be able to know where it is. Each phase of this evolutionary regime will be discussed separately in the following subsections. Figure 5 summarizes the results.

5.3.1 Capturing an immobile prey ($E_0^{0.0}$)

This initial stage serves to bootstrap the entire incremental evolution process by presenting a task that can be evolved from an initial random population. When $E_0^{0.0}$ is used as the initial evaluation-task, there is sufficient variation in the performance of networks to direct the genetic search. No memory is needed to accomplish $E_0^{0.0}$ so the agent only need to concern itself with the state of its sensory array, and no pursuit is involved so that the agent only needs to be able to predict the results of its own actions.

In this easier environment, some networks are able to survive significantly longer than others. Importantly, they survive longer by performing a primitive form of the fundamental skills required for Prey Capture. Some agent may do well capturing prey from the east, another from the west, while another from the north or south. Over the course of evolution the genetic recombination of these skills eventually produces a well-adapted individual that can capture the prey from all directions.

5.3.2 Increasing initial prey moves ($E_0^{0.0} \stackrel{\Delta}{\rightarrow} E_2^{0.0} \stackrel{\Delta}{\rightarrow} E_3^{0.0} \stackrel{\Delta}{\rightarrow} E_4^{0.0}$)

As the number of initial prey moves m is incremented, the ability of the agent to remember the position of the prey becomes increasingly important. When the evaluation task is $E_2^{0.0}$, the prey will often move out of sensor range. However, because of its probabilistic policy, the prey will also sometimes remain within the sensor range after m moves. As m is increased, the probability of the prey moving out of sensor range, and its distance from the agent, increases.

Because situations that demand memory are introduced gradually, an agent can still capture the prey most of the time even if it does not have the ability to always remember the prey's position. If the tasks were rapidly transitioned from $E_0^{0.0}$ to $E_4^{0.0}$, an agent would have to possess a general memory right away. When $E_4^{0.0}$ has been completed, the best network can capture the prey regardless of what direction it disappeared, and how far (within 4 moves).

5.3.3 Increasing prey speed ($E_4^{0.0} \stackrel{\Delta}{\rightarrow} E_4^{0.3} \stackrel{\Delta}{\rightarrow} E_4^{0.6} \stackrel{\Delta}{\rightarrow} E_4^{0.8} \stackrel{\Delta}{\rightarrow} E_4^{1.0}$)

After evolving an agent that can reliably remember where the prey is and can get to it, the prey is made mobile. Until now, the prey's position has been encoded in the agent's recurrent network, and when the prey moves, its sensory inputs do not match its internal representation of the situation and it does not perform well. However, at first the prey moves only one third of the time, and it is still sometimes possible for the agent to capture it because the prey is unlikely to make many moves during the time it takes the agent to capture it. Those agents that can follow the prey even just one move will have an advantage and will be selected for. Over several task transitions, the prey becomes gradually faster, and evolution will favor networks that pay more attention to the current sensory input in determining the prey's location. Eventually networks emerge that can pursue and capture the prey even when it is moving every time step, solving the goal-task.

Throughout incremental evolution, therefore, the changes made in the task are small enough so that the networks formed from the previous population can occasionally perform well. This makes it possible for evolution to discriminate between good and bad genotypes, and make progress towards the goal task.

6 Experimental Analysis

Given that general prey capture behavior was evolved, what do the solutions look like? That is, what kind of networks resulted, and what kind of behaviors they exhibit? Also, the result that incremental evolution outperforms direct evolution is significant only if the neuro-evolution method involved, ESP+delta-coding,

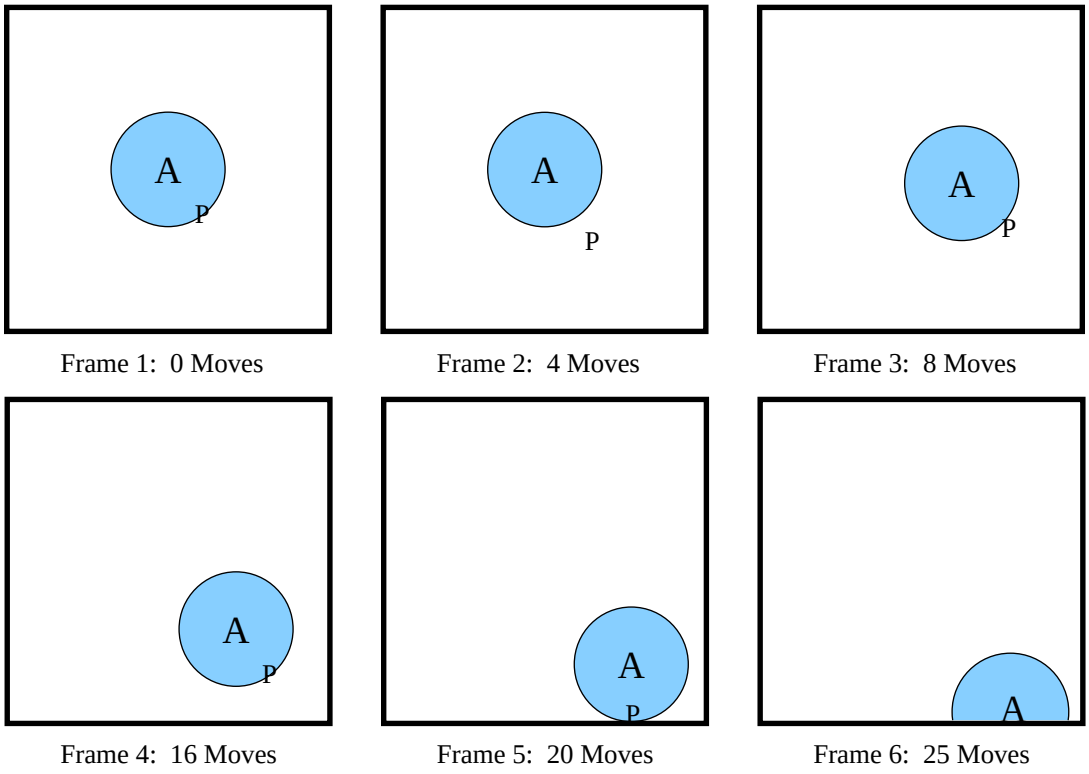


Figure 6: **An example of Prey Capture behavior.** The prey gets a head start of 4 moves and moves outside the sensory array. In 12 moves, however, the agent catches sight of it again, relying on its memory of where it last saw the prey. Eventually the agent pins the prey down against the wall and captures it. Similar scenarios occur from virtually all initial states, although the individual moves vary due to the stochastic nature of the prey.

is a powerful method on its own right. These issues will be examined in the following subsections.

6.1 Prey Capture Behavior

Figure 6 shows a sequence of "snap-shots" that illustrate a typical Prey Capture scenario in the goal task. In the first frame, the agent (denoted by the letter "A") is in its initial position, and the prey ("P") has been placed in a random position just within sensor range. At this point, the agent can see the prey. Frame 2 is taken four prey moves later. The prey is now outside the agent's sensory array, and the agent has not yet moved. In Frame 3, the agent has made four moves. The first move was selected while the prey was still in the SE sector. The next three moves, however, had to rely on a recollection of where the agent last saw the prey.

As the agent approaches the prey, it may not see it for several moves as the prey begins to flee. By move 16 (Frame 4), the agent has re-acquired the prey in its sensory array, and can begin to bear down on it. Since the prey will move every time-step, the agent can only capture it by trapping it against a wall. This behavior can be seen in Frame 5: The agent pursues the pray towards the wall, where its moves are limited and it is captured (Frame 6).

Similar prey capture behavior evolved in all simulations. Although behavior is easy to describe, it involves sophisticated components: remembering the likely location of the prey for several time steps, driving the

Lesioned Neuron	1	2	3	4	5
% Network Performance	63.2	21.9	47.7	48.1	25.1

Table 1: **Prey Capture performance of a lesioned network.** One of the successful networks was systematically lesioned by removing the input weights of each of its neurons in turn. The lesioned network was tested in the Prey Capture task and its performance was compared to that of the original network. For example, when the first neuron of this network was lesioned, it was still able to capture the prey 63.2% as many times as the complete network in a single trial. The results are averages over 100 trials. Similar results were obtained for all networks tested.

prey towards a wall, and capturing it by the wall. What is most important, though, is that the successful agents can perform this strategy from all different initial states, and with a prey that behaves stochastically. In this sense, the agents display believable and complex general behavior.

6.2 Network analysis

What do the successful Prey Capture networks look like, that is, how do the different specializations contribute and interact in Prey Capture? One way to analyze the contributions of individual neurons is to perform a lesion study: remove one of the neurons from the network and observe the effects on the network's behavior. The Prey Capture networks, however, are fully recurrent, and 4 out of the 5 units also serve as output units. Such units cannot be completely removed from the network. Removing for example the "north" output unit would only have the obvious and uninteresting effect of preventing the agent from moving north. Instead, a unit can be lesioned by disabling only its input connections (i.e. the connection from the sensory array), while still allowing it to receive recurrent signals from the other neurons. The functional role of the lesioned neuron may then be inferred by observing the behavior of the damaged network in Prey Capture.

The main result of the lesion study is that the networks are quite robust (table 1). When a neuron (any neuron) is lesioned, the behavior does not completely break down: the agent's tendency to pursue the prey is preserved to large extent, and it is still able to perform significant Prey Capture. When two neurons are lesioned simultaneously there is a corresponding double degradation in performance (varying between 38.1% for neurons 1 and 4 and 5% for 2 and 5).

It is difficult to attribute a particular behavior to any particular neuron. The coding of behavior seems to be distributed across the network. These results are in line with those of feedforward SANE networks for controlling a mobile robot (Moriarty 1997), where elementary behaviors such as advancing and turning and stopping in front of obstacles were also found to be distributed across multiple units. Recurrency apparently makes the behaviors even more distributed. Very few of the recurrent weights of a successful network are close to zero, which means that each neuron modulates the behavior of all other neurons. As a result, the functions are distributed across the whole network, and the system is very robust against degradations such as lesions, noise, and inaccurate weights values.

6.3 Pole-balancing

Prey Capture performance results (section 5) showed that incremental evolution is more powerful than direct evolution. However, ESP could simply be a weak method, in which case the result would have little significance. The result is important only if shown on a strong neuro-evolution method that could be chosen for a range of tasks. To this end, ESP was implemented in the standard reinforcement learning task of pole balancing, and shown that it outperforms the best known neuro-evolution methods.

The basic pole balancing system consists of a pole hinged to a wheeled cart on a finite stretch of track. The objective is to apply force to the cart at regular time intervals such that the pole is balanced indefinitely

Method	Pole Balance Attempts				Failures
	Mean	Best	Worst	SD	
GENITOR	1846	272	7052	1396	0
SANE	535	70	1910	329	0
ESP	285	11	1326	277	0

Table 2: **Performance of ESP on balancing a single pole compared to other neuro-evolution methods.** The results for SANE and GENITOR were obtained by Moriarty and Miikkulainen (1996a), who also showed that they are faster than the standard temporal difference methods of reinforcement learning such as the Adaptive Heuristic Critic (Anderson 1989; Barto et al. 1983) and Q-learning (Pendrith 1994; Watkins and Dayan 1989). The results are averages over 50 simulations.

and the cart stays within the track boundaries. The state of this system is defined by four variables: the angle of the pole from vertical θ, the angular velocity of the pole $\dot{\theta}$, the position of the cart on the track x, and the velocity of the cart $\dot{x}$. In the most common configuration the force is restricted to be of constant magnitude ("bang-bang" control), and the pole has a length of 1 meter (see Appendix B for the equations and parameters used in this task and its variations below).

Table 2 shows a comparison of ESP with two other neuro-evolution methods: the basic SANE system of Moriarty and Miikkulainen (1996a), and the GENITOR system of Whitley and Kauth (1988). GENITOR is an agressive search method with adaptive mutation that evolves full neural networks. In these simulations, a pole-balance attempt consisted of placing the cart in the center of the track with the pole in vertical position and allowing the network to control the system until θ exceeded 12 degrees or the cart moved off the track. ESP built 200 feed-forward networks in each generation, each with 5 units, resulting in approximately the same number of weights per network as in SANE. On average, ESP found solutions with only about half the pole balance attempts of SANE, and 1/6 of the attempts of GENITOR. It was also the most consistent of the three methods.

The basic pole balancing setup can be extended in many ways to make the problem more difficult. First simple modification is to allow the force to be continuous within a specified range. Second, the controller may be provided with only x and θ, so that it has to use recurrent connections to compute the derivatives $\dot{x}$ and $\dot{\theta}$ in order to balance the pole. Using a conventional NE method on networks with six recurrent units, Wieland (1990, 1991) was able to evolve a controller for this problem in an average of 10 generations. ESP solved the same task in as many generations on average (over 10 simulations), but tested less than half as many networks (200 instead of 512 per generation).

An even more challenging problem is to place a second pole next to the first. In this task, Saravanan and Fogel (1995) used Evolutionary Computation techniques to evolve a feed-forward network with 10 hidden units to balance the poles (1m and 0.1m) in an average of 800 generations. ESP was able to perform the same task in an average of 45 generations (averaged over 10 simulations).

These results show that ESP is a very powerful neuro-evolution method. It was able to solve both easy and difficult pole balancing tasks faster than the other NE techniques. However, as was seen in section 5.2, it still cannot solve the Prey Capture task directly. It is therefore likely that complex general behavior can be achieved with current evolution methods only through incremental evolution.

7 Discussion and Future Work

Prey Capture experiments illustrate the potential of incremental genetic search. The method is applicable to any problem that can be naturally decomposed into a sequence of increasingly complex tasks. However, the decomposition must be performed by the user and provided to the system before evolution commences. The user has to determine the sequence of tasks that will reliably afford successful transitions for a population

of a given size. This may not always be easy to do: There may not always be a clear way to simplify a task without decorrelating it completely from the goal-task, and it may not always be easy to tell what the relative difficulties of the tasks are.

However, at least for Artificial Life and robot control, the task sequences are usually easy to come by, because the goal-task often subsumes natural layers of behavior (Brooks 1986). For example, avoiding moving obstacles while following a wall could be evolved incrementally. The robot could first be evolved to move around avoiding stationary obstacles, then obstacles moving at slow speeds. Later, the speed of the obstacles would be increased, and the robot would be evolved to follow a wall. As long as any two consecutive tasks are not too distant from each other the system should be able to perform the transition. We have found that in situations where this is not the case, Delta-Coding can be invoked even within a task when performance ceases to improve significantly. In other words, if Delta-Coding is used more often, it is not as crucial to design the task sequence just right.

Once evolved, the controller can always serve as a starting point from which to build increasingly sophisticated behavior through continued application of ESP/Delta-Coding. In fact, the ESP/Delta-Coding approach can be used to modify any neural network regardless of how it was trained. This way it might be possible to start the evolution with basic behaviors already built in to the initial best network.

Ideally, the system should discover the sequence of evaluation tasks automatically with the user only providing the dimensions, such as prey speed and number of initial moves. The system could start with a base-level task where the environmental parameters are set to their trivial values (e.g. speed and initial moves to 0). When this task is achieved these values would be incremented by a small amount until the individual that performed best on the initial task can no longer cope with the environment. These parameter settings would then serve as the next evaluation task for the population. It might also be possible to vary the environment parameters continuously (from generation to generation) in response to average performance.

The Prey Capture experiments constitute a starting point for research on methods that evolve complex general behavior. Many extensions to the algorithm are possible, such as automating the task transitions, and extending the ESP/delta-coding algorithm to networks that vary in architecture. There are many tasks in the real world that require complex general behavior, including game playing and motor control. A major direction of future work will be to apply incremental evolution to these tasks.

8 Conclusion

This paper shows that even when a task is too difficult to evolve directly, Neuro-Evolution can be applied incrementally to achieve the desired complex behavior. In this approach, Enforced Sub-Populations allows evolution of recurrent networks, which are necessary for tasks that require memory. The Delta-Coding technique allows evolution to transition between tasks even when the population has lost diversity during the previous task. The approach should be applicable to many real world domains such as game playing and robot control, as well as Artificial Life , where often a natural hierarchy of behaviors from simple to complex exists.

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.

Appendix A: The prey movement algorithm

The prey's actions are chosen stochastically. On each step, $(1-v)\%$ of the time (where $v \in [0,1]$ is user-defined) the prey will not move, and $v\%$ of the time it will choose one of the four actions, A_0 (north), A_1 (south), A_2 (east), and A_3 (north), according to the following distribution:

$$prob(A_i) = P_i/(P_0 + P_1 + P_2 + P_3),$$

where

$$\begin{aligned}
P_i &= exp(0.33 \cdot W(angle) \cdot T(dist)) \\
angle &= \text{angle between the direction of action } A_i \text{ and the direction from the prey to the agent,} \\
dist &= \text{distance between the prey and the agent,} \\
W(angle) &= (180 - |angle|)/180, \\
T(dist) &= \begin{cases} 15 - dist & \text{if } dist \leq 4, \\ 9 - dist/2 & \text{if } dist \leq 15, \\ 1 & \text{otherwise.} \end{cases}
\end{aligned}$$

Appendix B: Pole-balancing parameters

The equations of motion for N unjointed poles balanced on a single cart are

$$\ddot{x} = \frac{F - \mu_c sgn(\dot{x}) + \sum_{i=1}^{N} \tilde{F}_i}{M + \sum_{i=1}^{N} \tilde{m}_i}$$

$$\ddot{\theta}_i = -\frac{3}{4l_i}(\ddot{x}\cos\theta_i + g\sin\theta_i + \frac{\mu_{pi}\dot{\theta}_i}{m_i l_i})$$

Where $\tilde{F}_i$ is the effective force from the ith pole on the cart

$$\tilde{F}_i = m_i l_i \dot{\theta}_i^2 \sin\theta_i + \frac{3}{4} m_i \cos\theta_i (\frac{\mu_{pi}\dot{\theta}_i}{m_i l_i} + g\sin\theta_i)$$

and $\tilde{m}_i$ is the effective mass of the ith pole

$$\tilde{m}_i = m_i(1 - \frac{3}{4}\cos^2\theta_i)$$

Parameters for basic single pole problem (i=1).

Sym.	Description	Value
x	Position of cart on track	[-2.4,2.4] m
θ	Angle of pole from vertical	[-12,12] deg.
F	Force applied to cart	-10,10 N
g	Gravitational acceleration	-9.8m/s^2
l	Half length of pole	0.5m
M	Mass of cart	1.0 kg
m	Mass of pole	0.1 kg

Parameters for the double pole problem.

Sym.	Description	Value
x	Position of cart on track	[-2.4,2.4] m
θ	Angle of pole from vertical	[-15,15] deg.
F	Force applied to cart	[-10,10] N
g	Gravitational acceleration	-9.8m/s^2
l_i	Half length of ith pole	$l_1 = 0.5$m $l_2 = 0.05$m
M	Mass of cart	1.0 kg
m_i	Mass of ith pole	$m_1 = 0.1$ kg $m_2 = 0.01$ kg
μ_c	Coefficient of friction of cart on track	0.0005
μ_p	Coefficient of friction if ith pole's hinge	0.000002

References

Anderson, C. W. (1989). Learning to control an inverted pendulum using neural networks. *IEEE Control Systems Magazine*, 9:31–37.

Barto, A. G., Sutton, R. S., and Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13:834–846.

Brooks, R. A. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(10).

Cliff, D., Harvey, I., and Husbands, P. (1992). Incremental evolution of neural network architectures for adaptive behavior. Technical Report CSRP256, School of Cognitive and Computing Sciences, University of Sussex, Brighton, UK.

Colombetti, M., and Dorigo, M. (1992a). Learning to control an autonomous robot by distributed genetic algorithms. In Meyer, J. A., Roitblat, H. L., and Wilson, S. W., editors, *From Animals to Animats 2, Proceedings of the 2nd International Conference on Simulation of Adaptive Behavior*. MIT Press.

Colombetti, M., and Dorigo, M. (1992b). Robot shaping: developing situated agents through learning. Technical Report TR-92-040, International Computer Science Institute, Berkeley,CA.

Elman, J. L. (1991). Incremental learning, or The importance of starting small. In *Proceedings of the 13th Annual Conference of the Cognitive Science Society*, 443–448. Hillsdale, NJ: Erlbaum.

Law, D., and Miikkulainen, R. (1994). Grounding robotic control with genetic neural networks. Technical Report AI93-223, Department of Computer Sciences, The University of Texas at Austin.

Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning, and teaching. *Machine Learning*, 8:293–321.

Lin, L.-J. (1993). Hierarchical learning of robot skills by reinforcement. In *Proceedings of the International Joint Conference on Neural Networks*. IEEE.

Miller, G., and Cliff, D. (1994). Co-evolution of pursuit and evasion i: Biological and game-theoretic foundations. Technical Report CSRP311, School of Cognitive and Computing Sciences, University of Sussex, Brighton, UK.

Moriarty, D. E. (1997). *Symbiotic Evolution of Neural Networks in Sequential Decision Tasks*. PhD thesis, Department of Computer Sciences, The University of Texas at Austin, Austin, TX. Technical Report UT-AI97-259.

Moriarty, D. E., and Miikkulainen, R. (1996a). Efficient reinforcement learning through symbiotic evolution. *Machine Learning*, 22:11–32.

Moriarty, D. E., and Miikkulainen, R. (1996b). Evolving obstacle avoidance behavior in a robot arm. In Maes, P., Mataric, M., Meyer, J.-A., and Pollack, J., editors, *From Animals to Animats: The Fourth International Conference on Simulation of Adaptive Behavior (SAB96)*.

Nolfi, S., Elman, J. L., and Parisi, D. (1990). Learning and evolution in neural networks. Technical Report 9019, Center for Research in Language, University of California, San Diego.

Nolfi, S., and Parisi, D. (1994). Good teaching inputs do not correspond to desired responses in ecological neural networks. *Neural Processing Letters*, 1(2):1–4.

Nolfi, S., and Parisi, D. (1995). Learning to adapt to changing environments in evolving neural networks. Technical Report 95-15, Institute of Psychology, National Research Council, Rome, Italy.

Pendrith, M. (1994). On reinforcement learning of control actions in noisy and non-Markovian domains. Technical Report UNSW-CSE-TR-9410, School of Computer Science and Engineering, The University of New South Wales, Sydney, Australia.

Perkins, S., and Hayes, G. (1996). Robot shaping–principles, methods, and architectures. Technical Report No. 795, Department of Artifical Intelligence, University of Edinburgh.

Ring, M. B. (1994). *Continual Learning in Reinforcement Environments*. PhD thesis, Department of Computer Sciences, The University of Texas at Austin, Austin, Texas 78712.

Saravanan, N., and Fogel, D. B. (1995). Evolving neural control systems. *IEEE Expert*, 23–27.

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

Thrun, S. (1996). *Explanation-Based Neural Network Learning: A Lifelong Learning Approach*. Kluwer Academic Publishers.

Watkins, C. J. C. H., and Dayan, P. (1989). Q-learning. *Machine Learning*, 8:279–292.

Whitley, D., and Kauth, J. (1988). Genitor: A different genetic algorithm. In *Proceedings of the 1988 Rocky Mountain Conference on Artificial Intelligence*. Computer Science Department, Colorado State University.

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

Wieland, A. (1990). Evolving controls for unstable systems. In Touretzky, D. S., Elman, J. L., and Sejnowski, T. J., editors, *Proceedings of the 1990 Connectionist Models Summer School*, 91–102. San Mateo, CA: Morgan Kaufmann.

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

Wilson, S. W. (1991). The animat path to AI. In Meyer, J., and Wilson, S., editors, *Proceedings of the First International Conference on Simulation of Adaptive Behavior*, 15–21. MIT Press/Bradford Books.

Yamauchi, B., and Beer, R. (1994). Integrating reactive, sequential, and learning behavior using dynamical neural networks. In Cliff, D., Husbands, P., Meyer, J. A., and Wilson, S., editors, *From Animals to Animats 3, Proceedings of the 3rd International Conference on Simulation of Adaptive Behavior*. MIT Press/Bradford Books.