Analysis of Neurocontrollers Designed by Simulated Evolution *

Karthik Balakrishnan and Vasant Honavar

Artificial Intelligence Research Group

Iowa State University

Ames, IA - 50011.

balakris@cs.iastate.edu, honavar@cs.iastate.edu

Abstract

Randomized, adaptive, greedy search using evolutionary algorithms offers a powerful and versatile approach to the automated design of neural network architectures for a variety of tasks in artificial intelligence and robotics. In this paper we present results from the evolutionary design of a neuro-controller for a robotic bulldozer. This robot is given the task of clearing an arena littered with boxes by pushing boxes to the sides. Through a careful analysis of the evolved networks we show how evolution exploits the design constraints and properties of the environment to produce network structures of high fitness. We conclude with a brief summary of related ongoing research examining the intricate interplay between environment and evolutionary processes in determining the structure and function of the resulting neural architectures.

1 Introduction

Artificial neural networks offer an attractive paradigm for the design of behavior and control systems in robots and autonomous agents for a variety of reasons [7, 5, 2] including: ability to adapt and learn, potential for resistance to noise, faults and component failures, potential for real-time performance in dynamic environments (through massive parallelism and suitable hardware realization).

The task of designing a good neuro-controller for a given robotic application presents an instance of a difficult multi-criterion optimization problem, often requiring complicated trade-offs among different measures of the network like performance, cost, complexity etc. Even if performance is the only objective of interest, the design is still difficult owing to the multi-objective tradeoffs inherent in realizing even the simplest of behaviors. For example, a robot might be required to overlook actions leading to immediate rewards in order to maximize eventual payoffs a scenarios that occur often in real-world applications.

Evolutionary Algorithms (EA) that are loosely inspired models of natural evolution, offer a particularly attractive tool for the design of neuro-controllers for a number of reasons [2]: They have been shown to be effective in searching several vast, complex, multi-modal and deceptive search spaces [8, 6]; EA constitute a form of population-based *reinforcement learning* and hence can deal with problems where precise knowledge of the desired actions in response to particular sensory inputs is lacking (and hence it is difficult to use conventional supervised neural network learning algorithms); More importantly, unlike most reinforcement learning techniques, EA can potentially search the space of network architectures (in addition to the more common search in the space of weights within an otherwise a priori fixed architecture) which makes it possible to apply them to problems where neither the shape nor the size of the solution (in this case, a suitable neuro-controller) is known in advance.

In this paper we summarize our attempts at evolving neuro-controllers for a robotic bulldozer which is given the task of clearing a square arena by pushing boxes to the sides. We analyze the evolved networks to show how the constraints and properties of the environment are exploited by the networks in order to attain high fitness.

*This is a summary version of ISU CS TR 95-25, available through on WWW through the URL: http://www.cs.iastate.edu/~ honavar/publist.html. This research was partially supported by the National Science Foundation through grant NSF IRI-9409580 to Vasant Honavar.

2 The Task Domain

The task environment, proposed by Teller [9], consists of an arena of $N \times N$ squares strewn with M boxes in the inner $(N-2) \times (N-2)$ grid. The arena is enclosed by impenetrable walls.

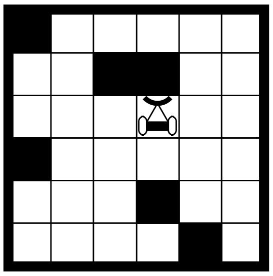

Figure 1: The operating environment. Shaded squares represent boxes and unshaded ones, empty space

The robot is started off in a random position, with the task of clearing the arena by moving the boxes to the sides. For simplicity, each box and the robot are assumed to occupy one complete square, as shown in Figure 1. The robot is equipped with eight sensors which provide the robot with specific inputs so that it can detect the presence of boxes, walls or empty spaces. The sensors are *fixed* to the robot, i.e., turn with the robot, and each sensor *senses* one square in each of the eight directions surrounding the robot. The robot is assumed to have two wheels, and is capable of performing the following actions — forward move, 90 degree anti-clockwise turn, and 90 degree clockwise turn. At any given time, the robot faces one of four directions — north, south, east or west.

The robot determines its output action based on the information provided by the sensors, and possibly the history of its past actions. Turns are in-place, while move forward actions result in the robot moving ahead by one square. If there happens to be a box immediately in front of the robot when it moves forward, the action results in the box being pushed ahead by one square. This is the primitive operation that the robot needs to clear the arena.

Attempts to push a box fail under two circumstances: the box may be already adjacent to a wall; or there is another box adjacent to it in the direction in which the box is being pushed. The latter is tantamout to saying that the robot has power to push *at most one* box at a time. The fitness of the robot is measured by the number of boxes it pushes to the walls within an allocated period of time. Our objective then, is to design a neuro-controller for this robotic bulldozer that attains as high a fitness as possible. For reasons mentioned in section 1, we use to EA to search for such a neuro-controller in a suitably chosen space of neural architectures.

3 Implementation Details

In our simulations we use *Genetic Algorithms* (GA) [8, 6] to evolve neuro-controller designs. We used a *genetic representation* that codes for the input connectivity of each of the units of the network as is shown in Figure 2 (ref [1, 3] for details).

The genetic operator *crossover* was defined to swap the input connectivities of units, while *mutation* was defined to either exchange two input connections of a unit, or randomly change the value of one input connection. Our simulations used populations of size 100, and the evolutionary runs lasted 100 generations. We used *uniform-crossover* with crossover rate set at 0.5, and mutation at the rate of 0.1 per gene. The weights in the network were constrained to be *integers* in the range [-100$\cdots$+100].

Each sensor provides one of three input values: $Box = 1$, $Empty\ Space = -1$ and $Wall = -2$, to the corresponding input unit of the neural network. At each time-step, the network updates its inputs to reflect the sensor readings and each neuron in the network computes its output (by applying the threshold function to its inputs) and passes it on to each of the neurons to which it is connected via its output links. Thus, at each time step, signals are propagated through a depth of one link through the network. The output of the network at each time-step results in an (instantaneous) action by the robot.

This model of neural computation crudely approximates a continuous-time discrete input-output neural network. The computational power of such networks is equivalent to that of finite state automata. The mapping of network output to actions is shown in Table 1. Thus, output Unit 1 codes for the *action* (move vs turn), while Unit 2 determines the direction of the turn.

The general framework for the simulation experiments was borrowed from [9]. Thus we used 6×6 arenas, with 6 boxes randomly distributed in the inner 4×4 grid. In each trial, the robot was placed in a randomly chosen empty cell, facing a random direction and allowed to perform $O(N^2) = 80$ actions (where each action takes one time step).

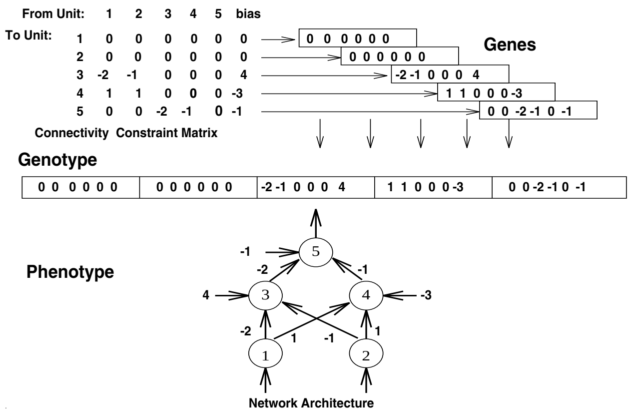

Figure 2: Genetic representation used

At the end of 80 time steps each box against the wall was awarded one point, while boxes in the corners fetched one extra point. The fitness of a neural network was evaluated by averaging its fitness over 40 random trials, where each trial involved placing the bulldozer in a random environment and simulating it through 80 time steps as described above. This fitness value was used by evolution in the *fitness-proportionate selection* step. The complete evolution experiment was repeated 10 times, with different initial random seeds, and the results were averaged.

4 Simulation Results

Evolution of the two classes of neural networks, feedforward and recurrent, were studied. In addition, we also investigated a third scenario wherein evolution started off with feedforward networks, but had the ability to add recurrent links if needed, through low probability mutation. This was meant to examine the effect of such a bias on evolutionary search. We label this class of networks — *initially feedforward.*

Unit 1	Unit 2	Robot Action
+1	-1	Move Forward
+1	+1	Move Forward
-1	-1	Clockwise 90^o
-1	+1	Anti-clockwise 90^o

Table 1: Interpretation of neuro-controller outputs

Figure 3 shows the course of one evolutionary run, where the system was evolving recurrent networks without hidden units. As can be observed, the networks of the initial population (generation 0) are highly unfit, with a best fitness of only ≈ 0.6 and a population average of ≈ 0.15. Fitness can be seen to improve over generations and in generation 100 the best network has a fitness of ≈ 2.75 and the population an average fitness of ≈ 1.81.

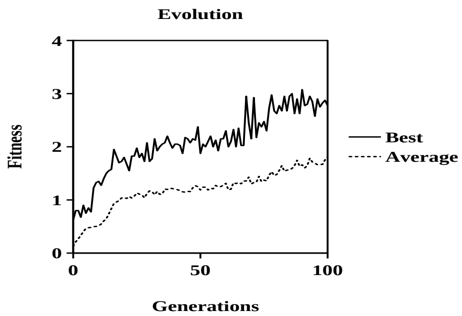

Figure 3: Demonstrating Evolution

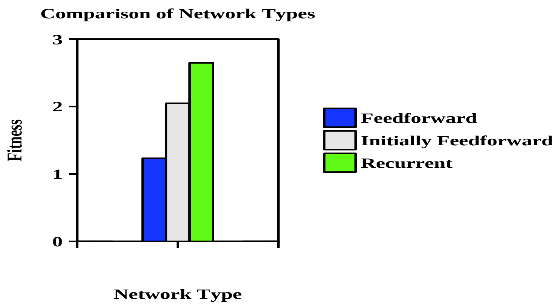

Figure 4: Performance of the bulldozer with different classes of neuro-controllers

Figure 4 shows a comparison of the three different network classes. In this experiment the networks did not have any hidden units and the fitness reported is the average of the fitness values for the best network found in each of the 10 evolutionary runs. Recurrence seems to have its advantages, since both recurrent and initially feedforward neuro-controllers perform well. The average fitness of the best recurrent network evolved was ≈ 2.65, significantly higher than the ≈ 1.23 fitness of the best feedforward network evolved. The class of initially feedforward networks evolve to an average fitness of ≈ 2.1. Interestingly, the networks in these experiments have on an average 30% (and sometimes even 70%) fewer recurrent links when compared to networks that were recurrent from the start of evolution. This might be a simple, yet useful, way of evolving recurrent networks with relatively few recurrent links.

4.1 Evolved Behavior of the Robot

The initial populations have extremely low fitness. This is due to the fact that the robots get into states in which they remain *stuck* till the end of simulation. For example, the robot might choose to keep spinning in the same location, try to (unsuccessfully) move into a wall, attempt to push two boxes at the same time, etc. In each of these cases, the robot is unable to *detect* that it has in fact been doomed to this state for eternity. The lack of such a detection mechanism is a constraint imposed by the design of the environment.

As evolution proceeds, fitness improves dramatically, suggesting that evolution finds a way of dealing with this environmental constraint. A careful analysis of the evolved recurrent networks indicates that over generations, these networks develop a *strong negative recurrent loop* at the output unit that codes for move/turn action, as shown in Figure 5.

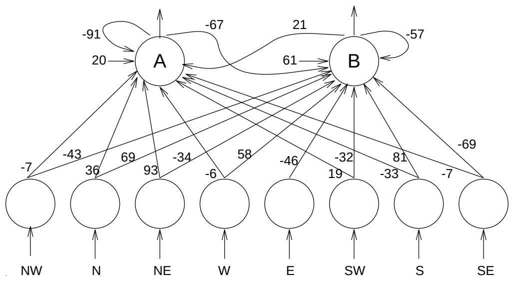

Figure 5: A strong negative self-loop at the action output unit (A) prevents the robot from getting into stuck states

In this network, if the current action is determined to be a forward move, owing to the strong negative self-loop, the network is automatically biased towards a turn action at the next time step. Strong sensory inputs can however override this tendency. Thus, it provides a potentially effective safeguard against getting into states in which it might be stuck forever. Since purely feedforward networks cannot take advantage of feedback provided by recurrent links, they perform quite poorly in comparison. This is borne out by Figure 6, which shows a plot of the actions performed by the robot in the course of one simulation run.

Figure 6: Actions taken by the robot at each time step

As can be observed in Figure 6, the feedforward network (with no hidden units) repeatedly moves forward till at some point it gets stuck. Once stuck, it remains in that state till the end of simulation. The recurrent network, on the other hand, frequently switches actions. By interleaving motions of moving forward and turning, this network avoids getting stuck indefinitely, thereby attaining a much higher fitness.

Even though recurrent networks manage higher fitnesses over their feedforward counterparts by evolving an intuitively appealing strategy, the fitness attained is still very low ($\approx$ 25% of the maximum attainable). Possible reasons for this include limited memory for previous sensory inputs and actions, limited simulation time, the possibility of getting stuck, limited sensor range and limited power. One way to increase memory is by allowing more hidden units, and such a robot can be expected to benefit from its history of past actions. This effect of increased memory is demonstrated in Figure 7.

In our simulations the robot is allowed 80 time steps within which it is to clear the arena. This choice was based on an $O(N^2)$ simulation time proposed by Teller [9]. However, a little deliberation shows that even under idealistic circumstances the robot needs $O(N)$ time for each of the $M = O(N^2)$ boxes, leading to an $O(N^3)$ time requirement. Figure 8 shows how the best network produced by evolution performs if only it were allowed more time to complete its task. Clearly, the network performs better when given more

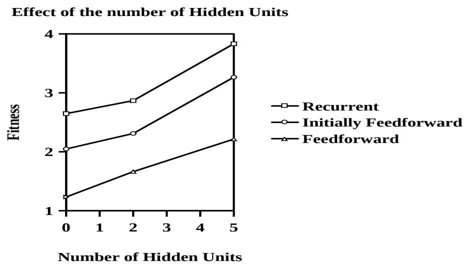

Figure 7: Performance of the robot with different numbers of hidden units

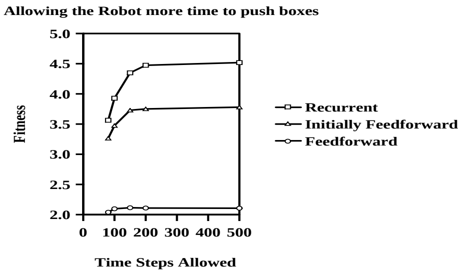

Figure 8: Effect of increased simulation time

time. However, its observed fitness saturates after a while as the robot still gets stuck often. Interestingly, the *knee* of the curves can be observed to be around *time steps allowed = 200*, seemingly corroborating our $O(N^3)$ conjecture for required simulation time.

5 Discussion

In this paper we have presented an evolutionary system for designing neural architectures for controlling the behavior of a robotic bulldozer. In this domain, the environmental constraint was the lack of any kind of a feedback mechanism which could inform the robot that it had not moved from its current position for a while. This situation might arise in a variety of ways, for example, the robot might be spinning endlessly in place, or trying to move into a wall etc. This constraint leads to robots that often get permanently stuck in a position, thus being highly unfit for the task on hand.

Our experimental results show how evolution overcomes this limitation by producing networks with large negative recurrent loops at the output unit coding for action. This strategy biased robots endowed with such networks to continually switch actions, thereby reducing their chances of getting infinitely stuck. This suggests that evolutionary approaches to design of neuro-controllers can potentially discover novel and perhaps even counter-intuitive solutions when faced with a diverse set of design, performance, resource, and environmental constraints.

Table 2 shows a different mapping of the neural network outputs to actions. inspired by Braitenberg's thought experiments with vehicles [4].

Unit 1	Unit 2	Robot Action
-1	-1	Move Forward
-1	+1	Anti-clockwise 90^o
+1	-1	Clockwise 90^o
+1	+1	Move Forward

Table 2: A different interpretation of neuro-controller outputs

With this modified scenario, the fitness of the networks produced through evolution was comparable to those described earlier. More interestingly, the networks exploit the same behavior — continually switching between actions to avoid getting stuck – albeit using different network architecture (to accomodate the change in mapping of network outputs to actions) in this case. For example, some recurrent networks developed strong self-loops of *opposite* signs at the two output units, which, in the new network model, alternates between the actions of move and turn. This offers an example of evolution discovering the same beneficial behavior under different sets of constraints.

Our ongoing research aims at examining the intricate interplay between the environment and the evolutionary processes in determining the structure and function of the resulting neural architectures. Our results are suggestive of the possibility that as the contraints are changed, evolution can potentially discover behaviors that are tailored for success within the given set of constraints.

As an example, consider a minor change in the environment such that the robot has an ability to detect when it spends over several (say 4) consecutive time-steps in the same position. In a limited sense, this can be interpreted as the availability of a mechanism by which the robot can determine when it is stuck. Once the robot detects it is stuck, it *randomly* changes its action, i.e., if its current action was to move forward instead it decides to turn and vice versa. This primitive strategy has interesting consequences on the evolved networks, as shown in Figure 9.

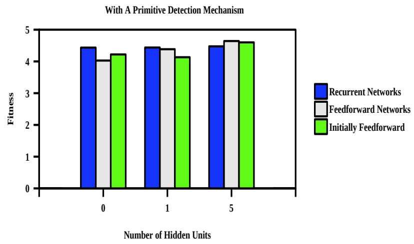

Figure 9: Performance of the robot when a primitive detection mechanism is available

As can be seen, feedforward networks are sufficient in this domain. Indeed, the networks evolve a curious strategy — they quickly learn to move straight ahead, and have little or no ability to turn. This works out fine, since if they do get stuck moving forward, they simply wait for 4 time steps before the detection mechanism causes a random turn. Since the moving forward behavior can be implemented in a network with no hidden units, the fitness can be seen to remain almost constant even with an increase in the number of hidden units.

The detection mechanism could be more complex than a 4 time-step countdown. Furthermore, once the robot detects that it is stuck, it could react in a variety of ways, not necessarily in a random fashion as above. This suggests the possibility of a range of built-in or evolved control strategies that evolution can exploit to produce networks of specific designs in each of the cases.

Exploration of evolutionary design strategies, genetic representations, genetic operators, and fitness functions for designing robust and flexible neuro-controllers under a variety of design, performance, and environmental constraints is the subject of ongoing and future research.

References

[1] K. Balakrishnan and V. Honavar. Analysis of neurocontrollers designed by simulated evolution. Technical Report CS TR 95-25, Department of Computer Science, Iowa State University, Ames, IA - 50011, November 1995.

[2] K. Balakrishnan and V. Honavar. Evolutionary design of neural architectures — a preliminary taxonomy and guide to literature. Technical Report CS TR 95-01, Department of Computer Science, Iowa State University, Ames, IA - 50011, January 1995.

[3] K. Balakrishnan and V. Honavar. Properties of genetic representations of neural architectures. In *Proceedings of the World Congress on Neural Networks*, pages 807–813, Washington D.C, July 1995.

[4] V. Braitenberg. *Vehicles: Experiments in Synthetic Psychology.* MIT Press, Cambridge, MA, 1984.

[5] D. Floreano and F. Mondada. Automatic creation of an autonomous agent: Genetic evolution of a neural-network driven robot. In D. Cliff, P. Husbands, J-A Meyer, and S. Wilson, editors, *From Animals to Animats 3: Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, pages 421–430, Cambridge, MA, 1994. MIT Press.

[6] D. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, Reading, MA, 1989.

[7] I. Harvey, P. Husbands, and D. Cliff. Issues in evolutionary robotics. In J. Meyer, H. Roitblat, and S. Wilson, editors, *From Animals to Animats II: Proceedings of the Second International Conference on Simulation of Adaptive Behavior*, pages 364–373, Cambridge, MA, 1992. MIT Press-Bradford Books.

[8] J. Holland. *Adaptation in Natural and Artificial Systems.* The University of Michigan Press, Ann Arbor, 1975.

[9] A. Teller. The evolution of mental models. In Kim Kinnear, editor, *Advances in Genetic Programming*, pages 199–219. MIT Press, Cambridge, MA, 1994.