Exploring evolutionary learning in a simulated hockey environment

Alan D. Blair[1]
Dept. of Computer Science &
Electrical Engineering
University of Queensland
4072, Australia
blair@csee.uq.edu.au

Elizabeth Sklar
DEMO Lab
Dept. of Computer Science
Brandeis University
Waltham, MA 02454-9110 USA
sklar@cs.brandeis.edu

Abstract- As a test-bed for studying evolutionary and other machine learning techniques, we have developed a simulated hockey game called *Shock* in which players attempt to shoot a puck into their enemy's goal during a fixed time period. Multiple players may participate – one can be controlled by a human user, while the others are guided by artificial controllers. In previous work, we introduced the Shock environment and presented players that received global input (as if from an overhead camera) and were trained on a restricted task, using an evolutionary hill-climbing algorithm, with a staged learning approach. Here, we expand upon this work by developing players which instead receive input from local, Braitenberg-style sensors. These players are able to learn the task with fewer restrictions, using a simpler fitness measure based purely on whether or not a goal was scored. Moreover, they evolve to develop robust strategies for moving around the rink and scoring goals.

1 Introduction

We have developed a test-bed for studying evolutionary and other machine learning techniques – a simulated hockey game called *Shock*, in which players attempt to shoot a puck into their enemy's goal during a fixed time period. Multiple players may participate and can be controlled either directly by humans or artificially, by any kind of intelligent software. One of our objectives in building the system was to create an environment for comparing and evaluating various learning techniques, as well as human interfaces. The work presented here focuses on the evolution of artificial controllers, trained to play Shock using a simple evolutionary algorithm.

Although our long-term aim is to co-evolve teams of players, for the present we focus on one-player scenarios, where the task is for a single player to learn the low-level motor commands necessary for manoeuvring the puck into its enemy's goal from a random initial condition and within a fixed time period. In order to perform the task well, a player must develop subtle skills, for example to nudge the puck away from a wall or out of a corner or to recover when the puck rebounds behind it, requiring the player to double back behind the puck. We have been exploring several methods for training artificial controllers to learn these techniques; and here, we concentrate on evolutionary computation.

In previous work [Blair and Sklar, 1998], we showed how neural network controllers could be trained in the Shock environment using a hill-climbing algorithm. In that work, the networks received complete information about their own position and that of the puck, expressed in the *global* coordinate system of the rink. Although these players were able to develop reliable goal-scoring strategies (given certain restrictions), a number of arguments can be made for trying to develop players which instead receive input only from simulated *local* sensing devices. While the use of global information conforms to the spirit of competitions such as Robocup soccer[2] [Kitano et al., 1995], nevertheless from an artificial life or adaptive behaviour perspective, creatures that observe the world from their own frame of reference come closer to the ideal of embodied cognition [Pfeifer, 1998]. Additionally, if Shock is to be considered as a simulation domain within the context of evolutionary robotics, it is preferable to develop controllers which carry their sensors on-board rather than relying on input from external sources. Finally, from a machine learning standpoint, we may enquire as to whether local sensors will provide a more natural representation and therefore improve generalization and robustness of evolved controllers.

With these motivations in mind, we now describe our efforts to evolve Shock players that take their input from local, Braitenberg-style sensors [Braitenberg, 1984] which respond when a stimulus appears in their field-of-view and return an activation value dependent on the distance to the stimulus. We show how these players are able to evolve robust strategies in general conditions, using only the raw fitness function of whether or not a goal is scored.

2 The *Shock* environment

Shock is a type of hockey game played in a rectangular rink on a near-frictionless surface (see figure 1). Collisions between the players, puck and walls are calculated by the "spring" method of collision handling [Keller et al., 1993] and are totally elastic. This means that each experiences a

[1]Current address: Department of Computer Science, University of Melbourne, Parkville, VIC 3052, Australia.

[2]In the small sized league, players may receive information from a camera positioned above the playing field.

restoring force in the normal direction, proportional to the depth of penetration. The puck collides frictionlessly with both players and walls, and therefore never acquires any spin. Each player experiences sliding friction when it collides with another player or a wall – a force in the direction opposite to the relative tangential motion and proportional to the restoring force. The players and puck also experience a frictional force as they slide on the rink, proportional to their velocity and in the opposite direction.

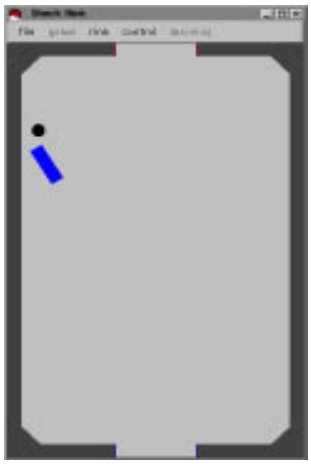


Figure 1: The Shock Rink.

The rink is 1m wide by 1.5m long, with 100mm diagonals at the corners, and a 300mm goal at each end.[3] Each player has a rectangular body 150mm by 50mm and a mass of 500 grams. The puck is circular in shape with a radius of 25mm and a mass of 100 grams.

Each player has a "skate" at either end of its body with which it can push on the rink in two directions, as illustrated in figure 2. This is reflected in the output of the simulator's controller, which is in the form (x_L, y_L, x_R, y_R) where (x_L, y_L) and (x_R, y_R) represent the forces exerted at the left and right skate, respectively. A restriction on the magnitude of the applied forces is imposed as follows:

$$x_L^2 + y_L^2 + x_R^2 + y_R^2 < \mathrm{F}_{\max}^2,$$

where $\mathrm{F}_{\max} = 0.1\mathrm{N}$. Note the redundancy in the format of the output; each player actually has only three degrees of freedom.

The simulation is updated in time intervals of 0.01 seconds, which runs in approximately real-time. A non-graphical version of the simulator, used for evolutionary runs, executes much faster and can complete a game of 15 simulated seconds' duration in about a quarter of a second[4].

[3]Earlier work [Blair and Sklar, 1998] used rectangular corners and 150mm goals. The new configuration was introduced because it discourages players from getting stuck in the corners, and prevents the goal-keeper from simply blocking the entire goal.

[4]on one dedicated processor of an SGI Power Challenge.

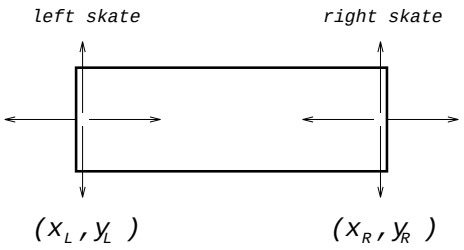


Figure 2: The Shock Player.

Shock differs from other real-time game simulators in some significant ways:

- It uses a *near-frictionless* environment, which provides special challenges distinct from those of traditional surfaces [Brooks, 1986, Sims, 1995].
- The rink is *compact*, distinguishing the playing arena from games that take place in an open plane [Miller and Cliff, 1994, Funes et al., 1998].
- A strategy for *collision management* (as opposed to *collision avoidance* [Xiao et al., 1997, Lee et al., 1996]) is needed in order to maneouvre successfully within the confines of the rink space and given the "zero-gravity" conditions provided by the near-frictionless surface.
- The players have rectangular bodies, which makes *orientation* a consideration and allows players to develop techniques different from those of circular players.
- The controllers employ two-dimensional *holonomic* actuators, enabling the players to move with more dexterity than standard one-dimensional wheels would allow.

Shock has been implemented in Java, and the controllers are coded in extensible classes. The entire package is available so others can download it, create controllers of their own and test them in our environment. Shock also includes a graphical user interface that allows us to view the controllers that have evolved and also to engage in games between a human player and a software controller. This gives us both visual and "hands-on" methods for evaluating players, to supplement statistical results of training runs.

3 Artificial controller development

We have been developing a number of neural network controllers, all of which take input information about the current game condition and return four output values indicating the magnitude and direction of forces to be applied at each skate (as described in section 2).

3.1 Input

We equip the player with six simulated Braitenberg-style sensors, each with a range of one meter and spanning an arc of $90°$, with an overlap of $30°$ between neighboring sensors (see figure 3). Each of the six sensors responds to three different kinds of stimuli: puck, enemy goal and friendly goal. This provides 18 input values; three additional inputs act as "encoders", informing the player of its current "observed" velocity (the difference between its current position and its previous position, divided by the time interval, expressed in the player's local frame of reference). As our work progresses from single- to multi-player scenarios, we plan to include two additional types of sensors which detect enemy and friendly players, respectively.

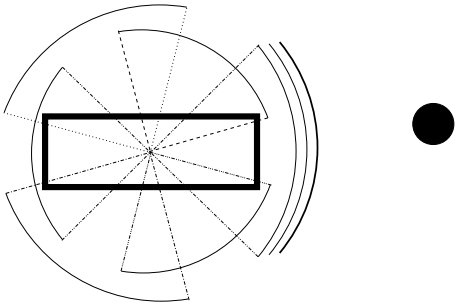


Figure 3: Local sensors.

These local, Braitenberg-style sensors respond to a stimulus in their field-of-view. A sensor reading near 1.0 indicates that the stimulus is nearby, while a reading near 0.0 indicates that it is some distance away. The six overlapping cones emanating from the center of the player show the field-of-view of each sensor. The varying diameters are used for illustrative purposes only; in actuality, the range of each sensor is uniform.

3.2 Architecture

We explore two different architectures: one-layer networks, which implement a linear function (as shown in figure 4), and two-layer networks with 10 nodes at the hidden layer (as shown in figure 5). Both architectures are fully connected. In the two-layer networks, the hyperbolic tangent function is employed at the hidden layer.

The output of each controller is a four-dimensional vector, $\mathbf{z}$, and is converted to an applied force vector as follows:

$$(x_L, y_L, x_R, y_R) = \mathcal{G}\left(\,||\mathbf{z}||\,\right)\,\frac{\mathbf{z}}{||\mathbf{z}||}\,\mathrm{F}_{\max}$$

where $\mathcal{G}$ is a bounded monotonic "transfer" function, and $\mathrm{F}_{\max}$ is the maximum force (defined in section 2). This preserves the direction of the output vector, altering only its magnitude. It also allows the network sometimes to apply a very small force by choosing x_L and x_R to be nearly opposites of each other. In the results reported here, the hyperbolic tangent function is used for $\mathcal{G}$ (although later experiments have revealed that a linear threshold function produces nearly identical results).

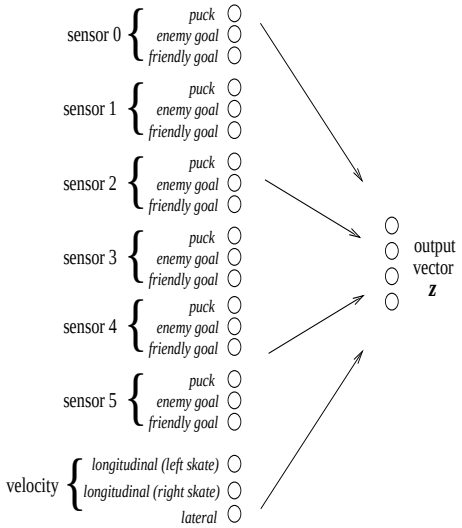


Figure 4: Single-layer (linear) network controller.

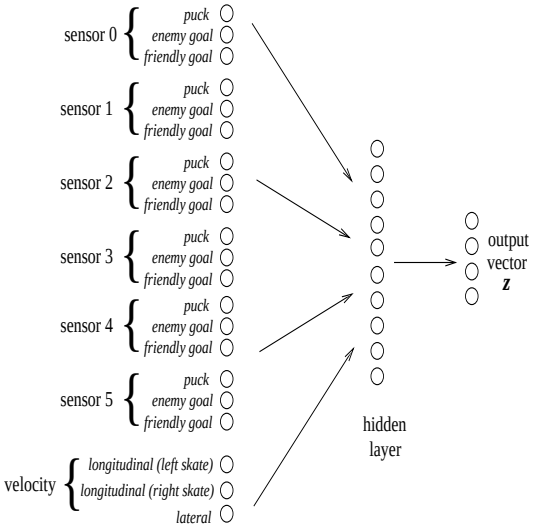


Figure 5: Two-layer neural network controller.

In our first experiments with the local coordinate system and the neural network controller, we found that in the initial stages of evolution, the outputs were dominated by the bias activation at the final layer; consequently, players would spin uncontrollably and were unable to learn. To counteract this problem, we removed the bias connections to the output layer, and kept them only at the hidden layer.

4 Evolutionary learning paradigm

The work presented here uses hill-climbing as a learning paradigm, applying a method which may be thought of as a simple evolutionary algorithm with a population of two. A

champ neural network is challenged by a series of *mutant* networks until one is found to do "better" than the champ; the champ's weights are then adjusted in the direction of the mutant. The initial weights of the champ are all 0. The basic steps for evaluating a single generation are as follows:[5]

1. mutant ← champ + Gaussian noise
   (with standard deviation σ for each weight)
2. champ and mutant play up to n games[6]
3. if mutant does better than champ,
   then champ ← $(1-\alpha) * \text{champ} + \alpha * \text{mutant}$

Each game begins with a random *game initial condition* (GIC) – a vector specifying a starting location and orientation for the player, and a starting location for the puck. These positions may be anywhere in the rink. The player then moves about the rink until either the puck goes into one of the goals, or the allotted time of 15 seconds elapses. A score is then awarded as follows:

$+1$ for hitting the puck into the *enemy* goal

-1 for hitting it into the *friendly* goal (i.e. the player's own goal)

0 for failing to hit it into either goal

Note that, even with no opponent, the need to avoid hitting the puck into the friendly goal is an important part of the task, since it prevents the player from learning to win by simply batting the puck around randomly until any goal is scored.

The specific criteria we chose in order for the mutant to be considered "better" than the champ were that the mutant must score strictly higher than the champ in the first game, and must continue to score at least as high as the champ in all subsequent games. In this way, no further games are played if the champ scores as high as the mutant in the first game, or higher than the mutant in any of the other games. Thus, the average number of games per generation is actually much lower than n.

The practice of making only a small adjustment in the direction of the mutant, modulated by the parameter α (which we call the *mutant influence factor*) was introduced in previous work on backgammon [Pollack and Blair, 1998] on the assumption that most of the strategies of the (well-tested) champion would be preserved, with only limited influence from the mutant – since a small number of games are not enough to determine absolutely whether the mutant is really better, or just lucky. In related work [Blair et al., 1998], applying a similar learning algorithm to the deterministic domain of *Tron* [Funes et al., 1998], we found that a larger value of α generally led (within limits) to a more robust player, by exposing the champ to a greater variety of challengers. Of course the optimal value of α will likely vary from one task domain to another.

In our earlier Shock experiments – which used controllers with input and output expressed in global coordinates – we found it necessary to introduce a restriction on the GIC's in order for the evolution to succeed. The restriction was as follows: the player must begin in the lower third of the rink, and the puck in the middle third. In the current, local coordinate system, we found that this restriction was unnecessary – i.e. the player and puck may begin anywhere in the rink. Although these unrestricted GIC's ultimately make the task harder, nevertheless in the early stages of evolution the opposite effect may be true, since occasionally a GIC will occur in which the puck and player are lined up close to the goal.

Additionally (in the previous work) we found it helpful to modify the fitness function, since in the early stages of evolution, the players are very unlikely to score at all. Therefore, we introduced *partial credit* adjustments in the fitness measure which rewarded a player for moving closer to the puck or for moving the puck closer to the enemy goal (and away from the friendly goal). This kind of "staged learning" or "shaping" [Dorigo and Colombetti, 1994, Perkins and Hayes, 1997, Digney, 1996] has proven very useful in a variety of contexts. However, there are potential drawbacks: first, its implementation may require the use of domain-specific knowledge, potentially compromising the generality of the learning algorithm; second, the shaping process may introduce distortions or perverse incentives which distract the learner from its primary objective [Angeline and Pollack, 1992]. For example, a player seeking partial credit might push the puck close to the goal without bothering to score, or might hover near to the puck without actually touching it – for fear of moving it in the wrong direction. We therefore attempt with the present work to determine whether the task can be learned with only a raw fitness function based purely on the number of goals scored.

5 Results

We present here results from four controllers: two single-layer networks (**lin0** and **lin1**) and two 2-layer networks (**nn0** and **nn1**). Each pair of networks was evolved using the exact same algorithm and parameters; only their starting seeds were different. First, we show the percentage of successful mutants, as the evolution proceeds, for all four controllers (figure 6). This is a good indication of how the pace of learning accelerates. Very few mutants are successful in the early stages of the 2-layer network evolution, since the network outputs are very small and the player is unable to score unless it is lucky in the assignment of GIC. This will be the case for both the champ and the mutant; recall that the mutant is considered successful only if it does strictly better than the champ – if neither champ nor mutant score, then the champ is not replaced. After 100K generations or so, the advancement begins in earnest.

[5] The results presented here used parameter values: $\sigma = 0.3$, $\alpha = 0.067$ and $n = 5$.

[6] The mutant is given the same GIC's as the champ.

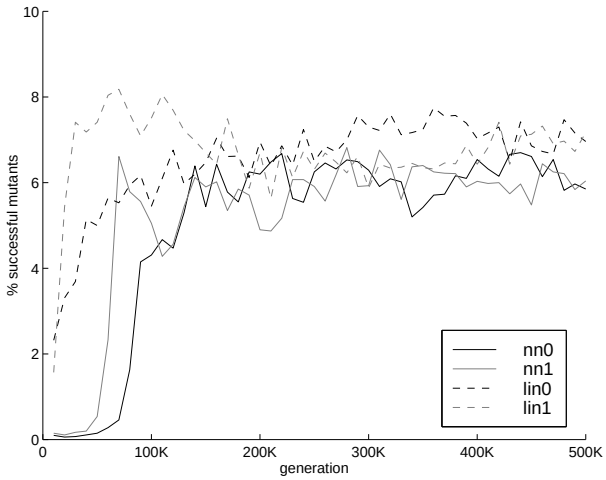


Figure 6: Percentage of successful mutants.

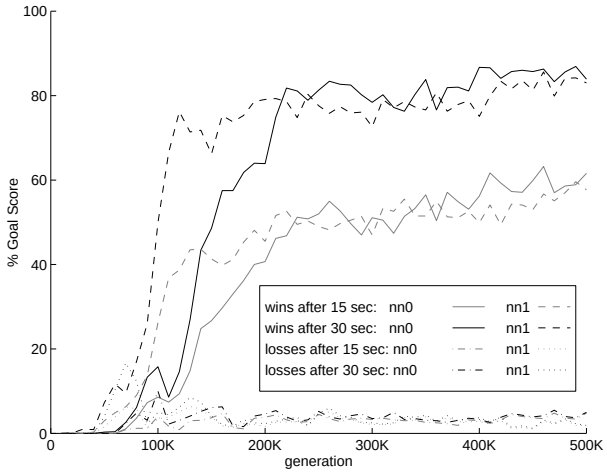
We tested the players on 1000 random GIC's, after every 10000th generation. Figures 7a and 7b show the percentage of goals scored out of these 1000 games, for contests lasting both 15 and 30 seconds of elapsed time. Both win rate and loss rate are shown. A "win" means that the puck went into the enemy goal; a "loss" implies that the puck went into the friendly goal. The win and loss rates are asymmetric because in some games, the contest time passed before the puck went into either goal.

The win rate for the 2-layer neural network players (figure 7a) reaches about 50% after generation 200K for contests of 15 seconds; for games lasting 30 seconds, it approaches 80%. The win rate continues to climb to around 60% and 85% by generation 500K. The loss rate, after reaching briefly into the teens within the first 100K generations, finally settles down to around 4%. As for the "linear" players (figure 7b) although **lin0** improves much more rapidly than **lin1** in the early stages, in the long run, both achieve a win rate of around 55% for the shorter games and 85% for the longer games.

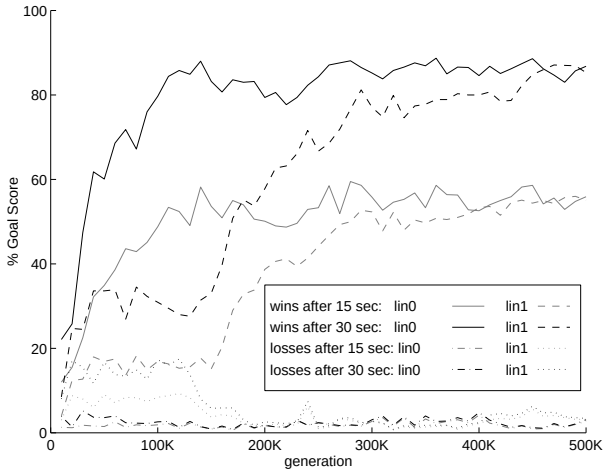
Figure 8 illustrates the progress of player **nn0** over 500K generations. The blackened player and puck highlight the GIC. The grey shapes are the final positions of each. We outline the player at regular intervals during the course of the game, to emphasize the changes in the player's orientation. In order to highlight the evolution of the player's abilities, we use the same GIC throughout the figure.

Initially (8a), the player spins uncontrollably and does not even approach the puck. By generation 200K (8b), the player has learned to move towards the puck, but has not yet developed adequate control over the direction of its approach. It narrowly misses the friendly goal, recovers well enough to come close to scoring, but then fumbles the puck at the edge of the enemy goal. Fortunately, it is able to swing around for a second shot.

The network at generation 300K (8c) is able to recover from the less serious folly of simply nudging the puck in



a. Two-layer neural network controllers.

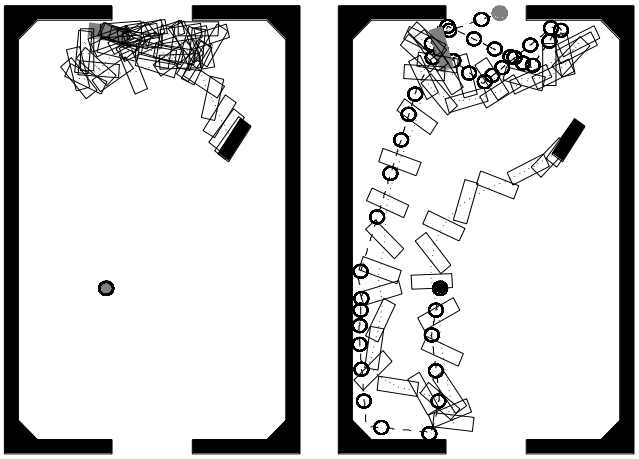


b. Single-layer (linear) network controllers.

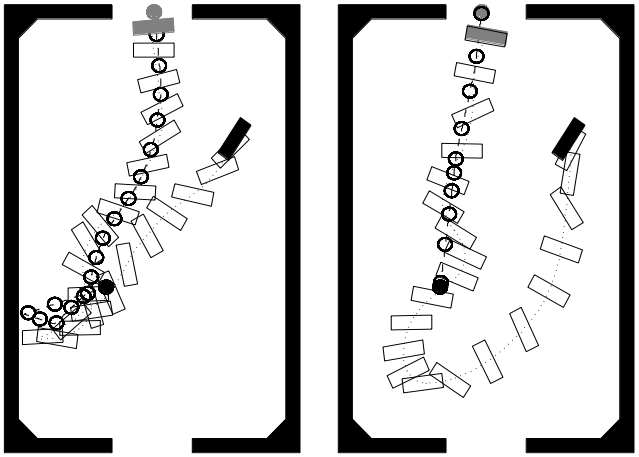
Figure 7: Number of wins and losses for contests lasting 15 and 30 seconds of elapsed time.

the wrong direction initially. By generation 500K (8d), the network has learned to integrate sensor information from the three different stimuli and comes around behind the puck to guide it smoothly into the goal. Training on the full repertoire of GIC's encourages players to develop robust strategies which help them to recover from their own blunders.

Finally, in Figure 9 we compare the abilities of some of the most advanced players. As an example, we chose one of the more challenging GIC's that was used in our previous work. Controllers **nn0** (9a) and **lin0** (9c) as well as **lin1** (not shown) learn to handle this GIC with relative ease, while **nn1** (9b) eventually scores, but not before fumbling the puck in two corners, doubling back and finally dribbling the puck into the goal. The controller in figure 9d was evolved in our earlier work, where some of the environmental and learning parameters were different. In particular, the maximum force allowed by the controller was double that of the value used in the current scenario (which accounts for the fewer number

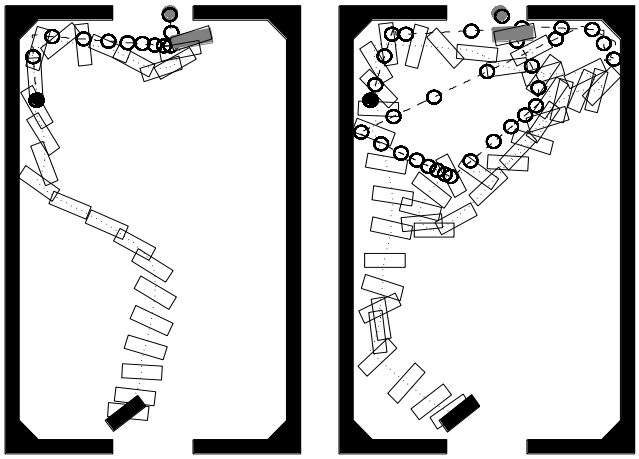


a. generation 100K b. generation 200K

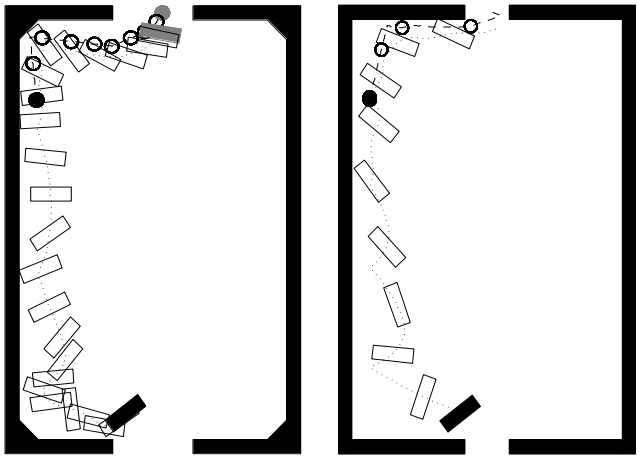


c. generation 300K d. generation 500K

Figure 8: The progress of player **nn0**.



a. **nn0**, 500K b. **nn1**, 500K



c. **lin0**, 300K d. global coordinates, nn 300K

Figure 9: A comparison of controllers.

of rectangles, implying a shorter amount of elapsed time occurred). Despite these differences, it makes for an interesting comparison – recall our aim to construct an environment in which to contrast various learning techniques.

6 Further work

Using simple evolutionary techniques, we have developed neural network controllers that learn to perform effectively in a simulated hockey environment. We are currently investigating how other training methods may be applied to this task – including supervised and reinforcement learning – which may reduce the amount of computation currently required.

We have found the Shock environment to be an interesting and non-trivial domain, allowing exploration of many factors that may contribute to the success or failure of evolutionary algorithms and other machine learning techniques. Our earlier work (with global coordinates) required shaping the task and modifying the fitness function during the preliminary stages of evolution. However, these adjustments were not necessary in the present work (with local coordinates). With further studies, we hope to understand better the issue of how a change in input method and reference frame can affect the learning process.

Aside from these ongoing efforts examining learning techniques, we are using Shock for several other avenues of research. These include (1) team play, (2) on-line human training, and (3) web-based tournament play.

In the area of team development, the next step is to introduce two-player scenarios. These will allow us to make direct comparisons by initiating contests between two players that use different controllers and/or training techniques. Eventually, teams with two or more players will lead us into the realm of multi-agent systems, where issues like communication and specialization become important factors.

Also, we are using the Shock environment to study the effect of active human input to a machine learner while it is adapting. We are building an interactive version of Shock which enables humans to train players using a supervised learning method.

Finally, we are in the process of creating a Shock tournament on the web. The purpose is to establish a proving

ground for different controllers, trained using a variety of methods. Users may submit controllers that will then compete in a continually running off-line tournament, and visitors to the web site will be able to check on their progress. Humans can also select any artificial controller and play games against it. Please visit our web site and participate...

`http://www.demo.cs.brandeis.edu/shock`

7 Acknowledgements

Thanks to Gordon Wyeth, Janet Wiles, Jordan Pollack, David Blair and Brett Browning for helpful comments. This work was funded by a University of Queensland Postdoctoral Fellowship and by the Office of Naval Research under N00014-98-1-0435.

Bibliography

[Angeline and Pollack, 1992] Angeline, P. and Pollack, J. (1992). Evolutionary induction of subroutines. In *Proceedings of the 14th Annual Conference of the Cognitive Science Society*.

[Blair and Sklar, 1998] Blair, A. and Sklar, E. (1998). The evolution of subtle manoeuvres in simulated hockey. In *Proceedings of the Fifth International Conference of the Society for Adaptive Behavior*.

[Blair et al., 1998] Blair, A., Sklar, E., and Funes, P. (1998). Co-evolution, determinism and robustness. In *Proceedings of the Second Asia-Pacific Conference on Simulated Evolution and Learning (also to appear as a volume of Lecture Notes in Artificial Intelligence, Springer-Verlag)*.

[Braitenberg, 1984] Braitenberg, V. (1984). *Vehicles: experiments in synthetic psychology*. MIT Press.

[Brooks, 1986] Brooks, R. (1986). A robust layered control system for a mobile robot. *IEEE Transactions on Robotics and Automation*, 2:14–23.

[Digney, 1996] Digney, B. (1996). Learning and shaping in emergent hierarchical control systems. In *Proceedings of Space'96 and Robots for Challenging Environments II*.

[Dorigo and Colombetti, 1994] Dorigo, M. and Colombetti, M. (1994). Robot shaping: Developing autonomous agents through learning. *Artificial Intelligence*, 71:321–370.

[Funes et al., 1998] Funes, P., Sklar, E., Juillé, H., and Pollack, J. (1998). Animal-animat coevolution: Using the animal population as fitness function. In *Proceedings of the Fifth International Conference of the Society for Adaptive Behavior*.

[Keller et al., 1993] Keller, H., Stolz, H., Ziegler, A., and Bräunl, T. (1993). Virtual mechanics – simulation and animation of rigid body systems. Computer Science Dept. Report 8/93, U. Stuttgart.

[Kitano et al., 1995] Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., and Osawa, E. (1995). Robocup: The robot world cup initiative. In *IJCAI-95 Workshop on Entertainment and AI/ALife*.

[Lee et al., 1996] Lee, W.-P., Hallam, J., and Lund, H. (1996). A hybrid gp/ga approach for co-evolving controllers and robot bodies to achieve fitness-specified tasks. In *Proc. 1996 IEEE Int'l Conf. Evolutionary Computation*.

[Miller and Cliff, 1994] Miller, G. and Cliff, D. (1994). Protean behavior in dynamic games: Arguments for the co-evolution of pursuit-evasion tactics. In *Proceedings of the Third International Conference on Simulation of Adaptive Behavior*.

[Perkins and Hayes, 1997] Perkins, S. and Hayes, G. (1997). Incremental acquisition of complex behaviour using structured evolution. In *Proceedings of International Conference on Neural Networks and Genetic Algorithms.*

[Pfeifer, 1998] Pfeifer, R. (1998). Embodied system life. In *Proceedings of the 1998 International Symposium on System Life*.

[Pollack and Blair, 1998] Pollack, J. and Blair, A. (1998). Co-evolution in the successful learning of backgammon strategy. *Machine Learning*, 32:225–240.

[Sims, 1995] Sims, K. (1995). Evolving 3d morphology and behavior by competition. In *Proceedings of Artificial Life 4*.

[Xiao et al., 1997] Xiao, J., Michalewicz, Z., Zhang, L., and Trojanowski, K. (1997). Adaptive evolutionary planner/navigator for mobile robots. *IEEE Transactions on Evolutionary Computation*, 1(1).