Co-evolution, Determinism and Robustness

Alan D. Blair[1] Elizabeth Sklar[2] Pablo Funes[2]

[1]Dept. of Computer Science
and Electrical Engineering
University of Queensland, 4072 Australia
+61-7-3365-1195
fax: +61-7-3365-1999
blair@cs.uq.edu.au

[2]Dept. of Computer Science
Brandeis University
Waltham, MA 02254 USA
+1-781-736-3366
fax: +1-781-736-2741
sklar,pablo@cs.brandeis.edu

Abstract. Robustness has long been recognised as a critical issue for co-evolutionary learning. It has been achieved in a number of cases, though usually in domains which involve some form of *non-determinism*. We examine a deterministic domain – a pseudo real-time two-player game called Tron – and evolve a neural network player using a simple hill-climbing algorithm. The results call into question the importance of determinism as a requirement for successful co-evolutionary learning, and provide a good opportunity to examine the relative importance of other factors.

Keywords: co-evolution, neural networks

1 Introduction.

In 1982, Walt Disney Studios released a film called Tron which featured a game in a virtual world with two futuristic motorcycles running at constant speed, making only right angle turns and leaving solid wall trails behind them. As the game advanced, the arena filled with walls and eventually one opponent would die by crashing into a wall. This game became very popular and was subsequently implemented on many computers with varying rules, graphic interpretations and configurations.

In earlier work [Funes et al., 1998], we built an interactive version of Tron (using Java) and released it on the Internet. With this setup, we created a new type of *co-evolutionary* learning where one population consists of software agents controlled by evolving genetic programs (GP) [Koza, 1992] and the other population is comprised of human users. This experiment has been fascinating for many reasons and from various points of view. From a human factors standpoint, the fact that this simple game has attracted a large number of users and that many of them return to play multiple games is unexpected. From an evolutionary programming standpoint, the fact that the robot players have evolved to embody a robust set of strategies, capable of overcoming a wide range of human behaviours, is a powerful result.

We have been studying co-evolutionary learning environments in several contexts [Blair and Pollack, 1997,Juillé and Pollack, 1996,Ficici and Pollack, 1998], trying to understand the reasons why this paradigm works very well for some tasks [Hillis, 1992,Sims, 1995,Tesauro, 1995] but poorly for others. In particular, we have developed a "minimalist" co-evolutionary learning method that consists of a neural network which evolves using a simple hill-climbing algorithm. We have found this to be a very useful tool for studying the effect of co-evolutionary learning in various task domains.

Previously, we have applied this method successfully to backgammon [Pollack and Blair, 1998] as well as a simulated robotic hockey game called *Shock* [Blair and Sklar, 1998]. Tron is similar to these domains in some respects but differs in other, significant, aspects. Backgammon is a *stochastic* domain in which the outcome of each game is influenced by random dice rolls as well as choices made by the players. In the Shock domain, each game is started from a different random initial condition. Tron, on the other hand, is totally *deterministic* in the sense that two games played by the same two opponents will necessarily be identical. Since many authors have cited non-determinism as a critical factor in the success of co-evolutionary learning systems, particularly in relation to backgammon, we were interested to apply our hill-climbing procedure to a deterministic domain, hence Tron.

This paper is organised as follows: the first two sections describe our Tron implementation and the network architecture and algorithm. Then we detail some experiments that we conducted to compare the neural network player with the GP players evolved in the Internet experiment. We conclude with some discussion and ideas for extending this work further.

2 Tron

Our interpretation of Tron abstracts the motorcycles and represents them only by their trails. Two players – one human and one robot – start in the middle region of the screen, heading in the same direction. Players may move past the edges of the screen and re-appear on the opposite side to create a wrap-around, or *toroidal*, game arena. The size of the arena is 256×256 pixels. The robots are provided with 8 simple sensors with which to perceive their environment.

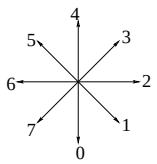


Figure 1 Robot sensors. Each sensor evaluates the distance in pixels from the current position to the nearest obstacle in one particular direction. Every sensor returns a maximum value of 1.0 for an immediate obstacle (i.e. a wall in an adjacent pixel), a lower number for an obstacle further away, and 0.0 when there are no walls in sight.

The robot can move in only two directions – horizontally and vertically. The game runs in simulated real-time, where each player can select one of the following actions: LEFT, RIGHT or STRAIGHT.

In the Internet experiment, data has been collected since August 1997 and is still accumulating. Over 2500 users have logged into the system and played at least one game. The average number of games played by each human is 53 games; the most games played by one player is 5028. Sixteen players have played over 1000 games.

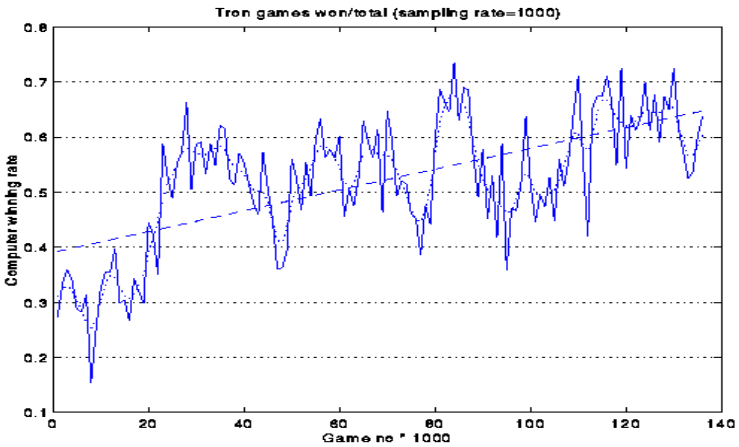


Figure 2 Internet Tron Results.

Our basic measure of performance is the *win rate* – the percentage of games that the GP players have won in playing against humans. As shown in figure 2, this rate has steadily risen from approximately 30% initially to more than 60% over a period of several months, resulting in a robust robot population capable of beating a wide variety of opponents. This "database" of players, along with the Java Tron environment, provide an excellent resource for testing and comparison with other methods.

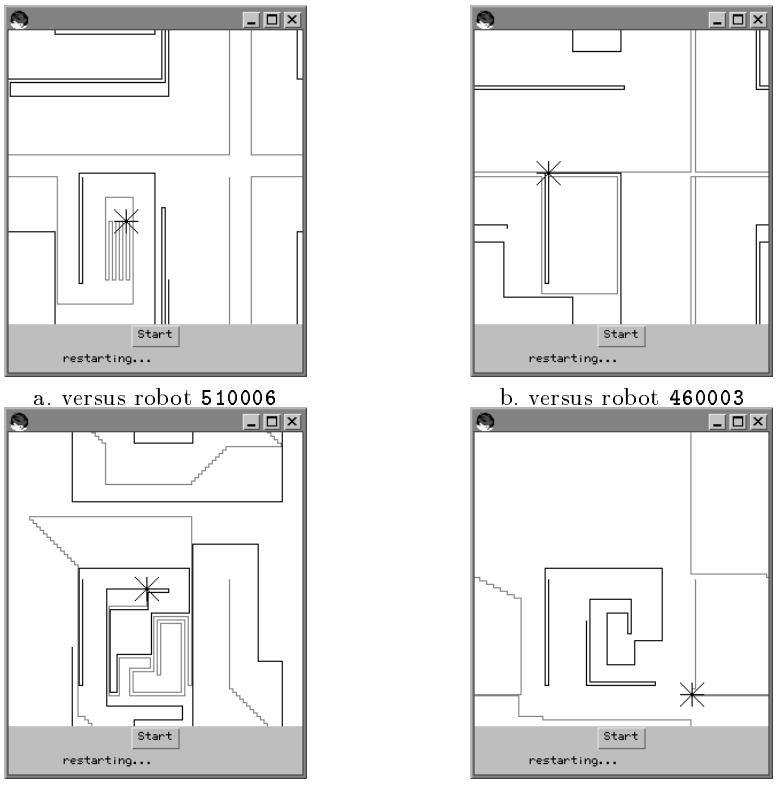
3 Implementation and Results.

In the present work, we develop Tron players controlled by two-layer feed-forward neural networks with 5 hidden units. Each network has 8 inputs – one for each of the sensors described earlier. There are 3 output units, representing each of the three possible actions (as above). Of these, the largest is selected as the "move" to make for the current time step.

We train the network using an evolutionary hill-climbing algorithm in which a *champ* neural network is challenged by a series of *mutant* networks until one is found that beats the champ; the champ's weights are then adjusted in the direction of the mutant:

1. mutant $\leftarrow$ champ + gaussian noise
2. mutant plays against champ
3. if mutant beats champ, champ $\leftarrow (1-\alpha) * \text{champ} + \alpha * \text{mutant}$

Using this neural network architecture, three players were evolved. Network `nn-0` was evolved for 1200 generations, networks `nn-1` and `nn-2` for 50000 generations each. The parameter α, which we refer to as the *mutant influence factor*, was set to 0.5 for `nn-0` and 0.33 for `nn-1` and `nn-2`. The network weights were saved every 100 generations and tested against five of the best GP players selected from our Internet experiment (referred to as robot players `510006`, `460003`, `480001`, `540004` and `400010`).[1] Note that the GP players were used purely for diagnostic purposes and had no effect on the evolving networks.



a. versus robot `510006` b. versus robot `460003`

c. versus robot `480001` d. versus robot `540004`

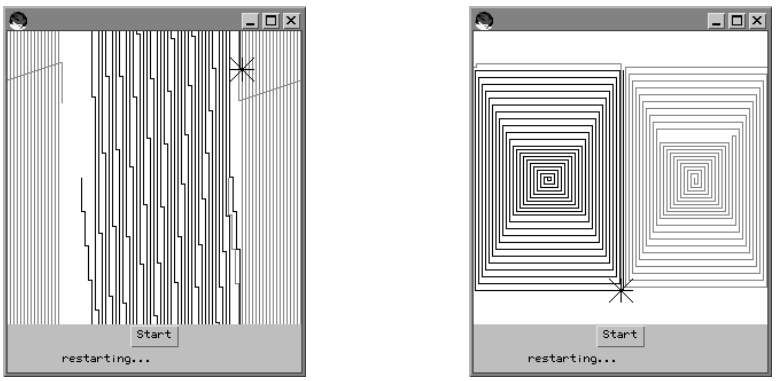
Figure 3 Network `nn-1`, generation 40500

Each of the robot players has a very distinctive behaviour (see figure 3). Players `510006` and `460003` follow similar strategies of trying to fill the arena in a contracting spiral, first carving an outline and then gradually moving inward, attempting to reduce the area available to the opponent. They exhibit a consistent inter-line spacing of approximately 12 and 4 pixels, respectively. When

[1] Note that this numbering is consistent with our previous papers on this work [Funes et al., 1998]; [Funes et al., 1997].

confined, both players seem to "panic", making a series of tight turns until either crashing or out-lasting their opponent.

Player `480001` often performs a kind of "coat-hanger" manoeuvre, turning at angles of 45° or 135° and proceeding diagonally by alternating left and right turns in rapid succession. Player `540004` is more aggressive, darting about the space in a seemingly erratic manner looking for opportunities to confine its opponent. Finally, player `400010` (shown in figure 4b) is very defensive, gradually moving outward in a tight spiral pattern with an inter-line spacing of 1 or 2 pixels.



a. generation 20000 vs robot `480001` b. generation 10000 vs robot `400010`

Figure 4 Defensive strategies.

The results of playing every 100th generation network against the five GP players are shown in figure 5, smoothed out by aggregation. The performance of network `nn-1` can be seen to gradually improve, peaking at around 70% after 40000 generations. In particular, the network sampled at generation 40500 was able to beat all 5 robot players (refer to figure 3).

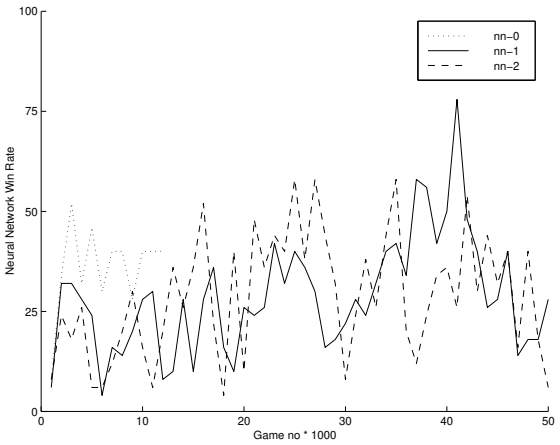


Figure 5 Neural network results.

It is interesting to note that the neural network players do not seem to evolve individual "traits" in quite the same way as the GP players; rather, they adapt to their opponent in a more *reactive* style (see figure 3).

Figure 6 illustrates the evolution of network `nn-1`. Each game shown is against robot player `510006`. The network makes early mistakes (a), but quickly learns a defensive strategy (b) and gradually (c) masters an offensive ability to "box in" its opponent (d).

a. generation 10000 b. generation 20000

c. generation 30000 d. generation 40000

Figure 6 Evolution of Network `nn-1`, versus robot player `510006`.

Network `nn-0` (not shown) developed a fragile defensive strategy similar to robot `400010`, filling the screen as slowly as possible in a series of expanding spirals. This method works well against 400010, an opponent with a similar strategy. It also happens to beat 510006 consistently, but loses almost all the time to the other three players.

4 Discussion

Co-evolutionary systems – particularly self-learning hill-climbers – often develop brittle strategies that perform well against a narrow range of opponents but are not robust enough to fend off strategies outside their area of specialisation. This brittleness has been overcome in a number of instances, but usually in domains that involve some form of non-determinism. Even though Tron is a deterministic domain, our self-learning hill-climbers seem to have learned the task quite well – performing capably against a selection of high quality GP players with a variety of different strategies.

It is interesting to note that `nn-0`, with a mutant influence factor of $\alpha = 0.5$, developed a fragile strategy which plays an almost identical game against every opponent, while `nn-1`, with $\alpha = 0.33$, developed an ability to react to different opponents in a robust manner. The practice of making only a small adjustment in the direction of the mutant – determined by the parameter α – was originally introduced in [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. However, it may also be that a lower value of α improves the robustness of the champion by exposing it to a greater variety of mutant challengers. Indeed, we conjecture that there may be an optimal value for α – which likely varies from one task to another. We plan to explore these issues in further experiments.

In future work we intend to make more extensive studies of Tron and other domains, in the hope of gaining more insight into the role of non-determinism in co-evolutionary learning, and the relative importance of other factors. We also plan to make the neural network players available in the Tron Internet system. Look for them on our web site... *http://www.demo.cs.brandeis.edu/tron*.

Acknowledgements

Thanks to Jordan Pollack, Janet Wiles and Brad Tonkes, and to Hugues Juillé for his help in providing the genetic program players. This research was funded by a University of Queensland Postdoctoral Fellowship and by the Office of Naval Research under grant N00014-98-1-0435.

References

[Blair and Sklar, 1998] Blair, A. and Sklar, E. (1998). The evolution of subtle manoeuvres in simulated hockey. In *Proc. of SAB-5*.

[Blair and Pollack, 1997] Blair, A. D. and Pollack, J. (1997). What makes a good co-evolutionary learning environment? *Australian Journal of Intelligent Information Processing Systems*, 4(3/4):166–175.

[Ficici and Pollack, 1998] Ficici, S. and Pollack, J. (1998). Challenges in coevolutionary learning: Arms-race dynamics, open-endedness, and mediocre stable states. In *Proc. of ALIFE-6*.

[Funes et al., 1997] Funes, P., Sklar, E., Juillé, H., and Pollack, J. (1997). The internet as a virtual ecology: Coevolutionary arms races between human and artificial populations. Computer Science Technical Report CS-97-197, Brandeis University.

[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 *Proc. of SAB-5*.

[Hillis, 1992] Hillis, W. D. (1992). Co-evolving parasites improve simulated evolution as an optimization procedure. In et al., L., editor, *Proc. of ALIFE-2*, pages 313–324. Addison Wesley.

[Juillé and Pollack, 1996] Juillé, H. and Pollack, J. B. (1996). Dynamics of co-evolutionary learning. In *Proc. of SAB-4*, pages 526–534. MIT Press.

[Koza, 1992] Koza, J. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA.

[Pollack and Blair, 1998] Pollack, J. B. and Blair, A. D. (1998). Co-evolution in the successful learning of backgammon strategy. *Machine Learning (to appear)*.

[Sims, 1995] Sims, K. (1995). Evolving 3d morphology and behavior by competition. In *Proc. of ALIFE-4*. MIT Press.

[Tesauro, 1995] Tesauro, G. (1995). Temporal difference learning and td-gammon. *Commun. of the ACM 39(3)*.