Evolution versus Temporal Difference Learning for Learning to Play Ms. Pac-Man

Peter Burrow and Simon M. Lucas, *Member, IEEE*

***Abstract*—This paper investigates various factors that affect the ability of a system to learn to play Ms. Pac-Man. For this study Ms. Pac-Man provides a game of appropriate complexity, and has the advantage that in recent years there have been many other papers published on systems that learn to play this game. The results indicate that Temporal Difference Learning (TDL) performs most reliably with a tabular function approximator, and that the reward structure chosen can have a dramatic impact on performance. When using a multi-layer perceptron as a function approximator, evolution outperforms TDL by a significant margin. Overall, the best results were obtained by evolving multi-layer perceptrons.**

I. Introduction

Ms. Pac-Man is a classic computer game which has the advantage of being complex enough to challenge current computational techniques while still being simple to implement. This paper uses a Java implementation as a testbed for investigating the influence of various design choices on a system learning to play the game. The utility of this investigation is in demonstrating how such factors interact to influence performance in a complex but easily reproduced test environment.

Design choices that have been explored include the performance of two algorithms – namely Temporal Difference Learning (TDL) and an Evolutionary Algorithm (EA). Evolution and TDL are arguably the most important approaches for learning to play games, and this research gives further insight into which methods work best in each case. The influence of the choice of reward structure in the case of TDL is also explored, especially its interaction with the choice of a function approximator. In the case of evolution, comparisons are also made between Darwinian and Lamarkian evolution.

II. Background

A. Algorithms

Both algorithms considered in this paper are widely used in the field of Computational Intelligence. Previous work on the combination of the two has also been performed.

1) Evolution: Evolutionary Algorithms are an approach inspired by biological evolution. They include the concepts of variation, selection, reproduction and survival of the fittest. A population exhibits variation, and after being assessed on a task the fittest individuals are selected to produce the next generation. This allows adaptation to the task over multiple generations.

Peter Burrow and Simon M. Lucas are with the School of Computer Science and Electronic Engineering, University of Essex, Colchester CO4 3SQ, United Kingdom. (emails: prburr@essex.ac.uk, sml@essex.ac.uk).

Notable early work on artificial evolution includes work by Fogel et. al. [1] on evolutionary programming, and work on genetic algorithms by Holland [2] as a way of performing a massively parallel search through the fitness landscape.

Evolving artificial neural networks, or *neuroevolution* [3] is a particularly popular use of evolutionary algorithms. This traditionally involves selection of the network architecture and evolution of the connection weights, although there has also been work on evolving the network architecture [4][5].

A good overview of Evolutionary Algorithms is provided by Mitchell [6].

2) Temporal Difference Learning: Temporal Difference Learning works using feedback from the task in the form of rewards. It also adjusts predictions of state values using estimates of the values of subsequent states (i.e. it uses temporally different predictions in order to bootstrap its estimates).

Temporal Difference Learning has previously been used on a wide variety of problems. Early use can be traced back to Samuel [7] and Michie [8]. One of the most famous applications of TDL was TD-Gammon, which used TDL to learn to play the board game Backgammon [9][10][11]. Despite starting with little Backgammon knowledge, it learned to play at a level similar to the leading existing programs.

A good overview of Temporal Difference learning can be found in [12].

3) Hybrid Approaches: Previous work on the subject of combining learning and evolution includes the following:

Whiteson and Stone [13] combine NeuroEvolution of Augmenting Topologies (NEAT) [5] – a neuro-evolutionary optimisation technique, with Q-learning – a type of TDL. They test this combination on two problem domains – the "Mountain Car" task, and the "Server Scheduling" task. Their results suggest an increase in performance over the individual methods in isolation. They compare the performance of Darwinian and Lamarkian approaches to evolution, and outline an approach to online Evolutionary Computation.

Lucas and Togelius [14] investigate co-evolutionary learning and TDL in the context of simulated car racing. They also experiment with a simple combination of the two approaches, running an evolutionary algorithm on a TDL trained solution, and vice-versa. They found seeding TDL with evolved controllers improved performance, but seeding evolution with TDL-trained controllers made little difference over straight evolution.

978-1-4244-4815-9/09/$25.00 ©2009 IEEE

B. Pac-Man

Pac-Man and Ms. Pac-Man have in recent years been used as a test environment by other researchers. While in Pac-Man the ghosts moved in a deterministic fashion, allowing a player learn a strategy that always works, in Ms. Pac-Man the ghosts move in a non-deterministic fashion.

Gallagher and Ledwich [15] used a simple evolutionary algorithm to train a neural network to play Pac-Man. They used a simplified version of the game (often with just one ghost), and experimented with both deterministic and non-deterministic ghost behaviour. Minimal input processing was performed – the inputs to the neural networks expressed the presence of pills, wall, and ghosts in each cell of a (typically 5×5) grid centred on Pac-Man. There were also four inputs expressing global concepts – namely the number of pills remaining in each of the four directions from Pac-Man. Performance was very modest, but showed that significant learning could take place using essentially raw input to a large neural network.

Wittkamp et. al. [16] also evolved neural networks for Pac-Man, but to learn ghost strategies rather that to control Pac-Man itself. They used Neuro-Evolution of Augmenting Topologies (NEAT) with a number of populations, and demonstrated successful development of team strategies. Using a hand-coded 'pacbot' to control Pac-Man, they evolved ghost strategies that can outperform those from the original game. It was shown that the metrics used for fitness evaluation have a significant effect on the final behaviour. Different evaluation functions can produce ghost strategies that outperform those of the original game in first-level score, or in number of lives lost during the first level.

Lucas [17] used evolution to train a neural network to play Ms. Pac-Man. Features used include distances to notable items, such as ghosts, nearest pill/power pill and junction. The distances were calculated along the maze paths rather than geometrically, as this would be misleading for the learning algorithm and could lead to getting stuck in corners.

Szita and Lörincz [18] demonstrated good performance (scores of up to $\sim$8000 – similar to that of inexperienced human players) using reinforcement learning to play Ms. Pac-Man by choosing a strategy based on either hand-coded or random rules.

Ms. Pac-Man has also been used in recent competitions, for example at WCCI 2008 [19]. This competition was run using a screen capture interface to the Ms. Pac-Man game. Despite timing delays due to the screen capture and image processing, the winning entries regularly scored around 15,000 – significantly better than inexperienced players.

C. Previous experiments

Previous unpublished experiments by the authors on a simpler problem – that of the Mountain Car, also using the same code base – have suggested some general points to keep in mind during these experiments.

1) Convergence speed & stability: It was found that for the Mountain Car problem, a state table learned both more quickly and more reliably than a Multi-Layer Perceptron (MLP) (see Section III-C2) when using TDL.

2) Problem specification: In the case of the Mountain Car problem, the original description of the problem mentions clamping the velocity at one end of the problem space, but not at the other. It was found that the decision to clamp velocity to zero at the other end (upon reaching the goal state), caused an improvement in learning performance. This shows it is important to describe any problem domain as fully as possible in order to allow systematic comparison of results.

3) Stabilising the MLP using explicit end state value: When comparing state tables and MLPs as function approximators for TDL, it was noted that MLPs were less reliable in learning the correct form of the value function. In the case of the state table, TDL learned the correct form of the value function 100% of the time. In the case of a randomly initialised MLP, it would struggle to converge to the correct form of the value function.

It was hypothesised that this was partly due to a lack of correct values for any state to use as a reference. For example, in the state table case, the value of the end state remains zero, and TDL uses this to calculate the correct value for the penultimate state, and so on back through the possible state values. In the case of an MLP however, upon being initialised randomly it is unlikely that any state value is correct, and so the state value used in the "next state" term of the TDL update is unlikely to be correct. In addition, altering the MLP weights to change a value at one point in the state space affects the value at all other points.

In order to improve the process, code was added so that if the current state is also the end state, the value zero is used explicitly for this state in the TDL update rule. This helps to give the MLP training process an accurate point of reference and results in increased chance of learning the state value function effectively. This practice of explicitly using zero for the value of the end state was therefore used in the experiments in this paper.

III. Experimental Specifics

A. The Ms. Pac-Man Game

1) Game Implementation: For this paper, a pure Java implementation of the game by the second author is used. This has the advantage of easy integration with existing Java code for other aspects of computation learning.

As in the original game, there are four different ghosts. Each ghost has a different colour, and slightly different (non-deterministic) behaviour, with the red ghost being the most aggressive, and the orange ghost the least. Ms. Pac-Man must navigate the maze and eat pills and power pills, while avoiding being eaten by the ghosts. There are 220 pills and 4 power pills in this implementation, and the maze layout corresponds to the first maze of the original game (which had four mazes in total). The power pills allow Ms. Pac-Man to eat the ghosts for a short time. Ms. Pac-Man has three lives, and to simplify the game there is no fruit to eat

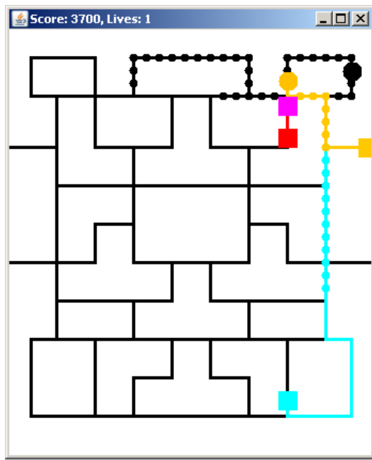

Fig. 1. Estimating ghost paths. These are the worst-case assumption that all ghost take the shortest path straight to Ms. Pac-Man. This screenshot shows that all the ghosts are predicted to come from the same direction.

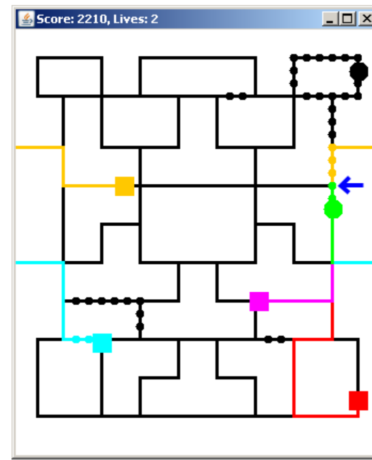

Fig. 2. Identifying the nearest escape point. In this example, ghosts are predicted to approach Ms. Pac-Man from both directions. In this case, the nearest escape point is the junction shown by the arrow.

– in the original game Ms. Pac-Man could eat moving fruit for extra points.

Ms. Pac-Man scores 10 points for eating a pill, 50 points for a power pill, and a variable number of points for eating edible ghosts. The first ghost eaten after a power pill scores 200 points, which doubles for every ghost eaten until the power pill wears off.

Because the code is accessible, it is easy to allow a rudimentary form of lookahead to simulate the results of actions taken by the controller, which can then be exploited by the state value controllers used in this experiment.

It should be noted that the implementation as it currently stands does not return a perfect prediction of the results of an action. The prediction is formed by moving Ms. Pac-Man, but not updating the ghost and pill positions. This allows for easy simulation of the next state, but could be improved with further work in order to give a true prediction of the state resulting from a particular action. This limitation means that the TDL described in this paper is not strictly true TDL, since the estimate of the next state is inaccurate.

2) Feature Extraction: A separate Java class is used to extract features of the game state to be presented to the controller. In these experiments only two features are used – mainly to allow easy visualisation of state values in a 2D feature space.

The code uses shortest path information to make a prediction about the likely path of the ghosts. This prediction is a significant oversimplification, since it simply assumes that all four ghosts will always take the shortest path to Ms. Pac-Man. This represents the most aggressive possible course of action, although in reality all four ghosts have different (non-deterministic) behaviours. Using the shortest path to Ms. Pac-Man from each of the ghosts, the predicted route of each ghost can be traced.

Figure 1 shows the Ms. Pac-Man game with the predicted ghost paths coloured. The lines represent the maze, and the small circles represent pills. Ms. Pac-Man is the lighter coloured large circle, and the darker circle is a power pill. The squares correspond to the four ghosts. Each predicted path is coloured to correspond to the colour of its ghost. When a predicted path appears to stop before reaching Ms. Pac-Man, this is due to the paths overlapping.

This prediction of ghost paths allows the feature extractor to tell what directions ghosts are likely to come from. In the game state shown in Figure 1, all the ghosts are predicted to approach Ms. Pac-Man from the same direction.

It is useful to define the concept of an 'escape node' This is defined as a node where ghosts are not coming from all directions. This is useful because such a node allows Ms. Pac-Man to pick a route away from where *all* the ghosts are likely to come from, and thus stay ahead of them.

The first feature for a particular node (the candidate node) is therefore formed as follows. The distance from the candidate node to the nearest escape node is calculated. Either the candidate node is itself an escape node, or the escape node nearest to it is found. Figure 2 shows a situation where ghosts are predicted to approach Ms. Pac-Man from both directions. The nearest escape node is indicated by the arrow.

If the distance from the candidate node to the nearest escape node is termed P, and the distance from that escape node to the nearest ghost is termed G, the first feature is the result of $G - P$. Thus values larger than zero imply that Ms. Pac-Man is closer to the escape node than all the ghosts, and so is less likely to be trapped. A negative value implies that the ghosts will be able to cut off Ms. Pac-Man prior to reaching this escape node, and thus Ms. Pac-Man has a good chance of being eaten. This feature corresponds to the x-axis in Figure 7.

The second feature used for these experiments was simply the distance from the candidate node to the nearest pill along the shortest maze path. These two features are used as input to a function approximator (see Section III-C) to give an estimate of the value of moving to the candidate node.

This choice of features is clearly a vast simplification of the state of the game, but it was decided to use only these two features because it allows the resulting (2D) feature space to be easily visualised when analysing the relative state values that are learned. It also allows the inclusion of some rudimentary global and geometrical information in the form of the predictions about the ghost paths and thus (hopefully) the possibility of being trapped.

With this choice of features, it should not be expected that performance be comparable with leading implementations. The aim in this paper was to allow analysis of learning behaviour. It is however important to point out some of the limitations of this simple model. One aspect that is particularly worth noting is that the formation of the first feature allows the possibility of a small change in game state causing a large jump in feature space. This became apparent during the experiments discussed in Section IV, and may be expected to have an effect on how easy it is to learn a good strategy.

B. Learning Algorithms

Two learning algorithms are used in this investigation – Temporal Difference Learning and Evolution.

1) Temporal Difference Learning: A simple TD(0) version of TDL is used in this paper, with the standard update rule of:

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$$

$V(s_t)$ is the value of the state at time t, and is updated using the reward r_{t+1} from the state at $t+1$. In these experiments, the learning rate α is set to 0.2, and the discount factor γ to 1.0.

2) Evolution: A standard ES(15 + 15) evolutionary algorithm is used for this investigation. A population of 30 controllers (see Section III-C) is evaluated on the task, then the best-performing half of the population are kept and mutated versions of each are used to replace the other half of the population.

Only mutation is used here - crossover is not used - although this would be an interesting experiment for future work. Initial unpublished experiments showed that good learning performance could be achieved just using mutation. The values in the function approximator are mutated by adding an amount drawn at random from a gaussian distribution (with mean 0 and standard deviation of 0.1).

In some of the experiments, Lamarkian and Darwinian versions of this algorithm are used. Both these variations allow TDL learning during the 'lifetime' of an individual. In the Lamarkian case, these adjustments to the state values are retained over generations, whereas during Darwinian evolution, these changes are discarded after each generation.

C. Function Approximators

A state-value controller is used in all the experiments. To use a state-value controller is perfectly legitimate, and it should be easier to learn because (other things being equal) it should have fewer parameters than an equivalent action-value controller.

A function approximator is used to store the value function over the feature space. The controller selects an action at each time step based on which state resulting from the possible actions has the highest state value. Two function approximators are used in this investigation.

1) An interpolated table: At its simplest, a table simply divides up the feature space, and stores a state value for each division. However, this leads to large tables with a high storage cost, or smaller tables where many states effectively have the same value. Since this can have a detrimental effect on performance, a table with bilinear interpolation is used as one of the function approximators. This combines the advantage of small data storage cost with the ability to allow all different states to have different values associated with them. It should be noted that since only the nearest four data points are used for the interpolation, and bilinear interpolation is used, the state values will not change smoothly at all points in the feature space. However, the benefits of this improvement in state representation are easily demonstrated by improved performance. See also work by Lucas [20].

The experiments in this paper use a multi-layer interpolated table. In this type, multiple interpolated tables of different sizes are averaged to obtain the value of a point in the 2D feature space. This scheme has the advantage of giving a smoother function approximation, and the ability to easily represent features at different scales. Preliminary experiments showed this type significantly outperformed single-layer interpolated tables. It should be noted however, that lookup in the multi-layer table case is correspondingly slower, since multiple interpolations must be performed. In these experiments a four-layer interpolated table was used, with tables of sizes 32, 19, 11, and 6. The value of a state was found by averaging the values from each of these tables.

Preliminary experiments demonstrated that evolution as described in Section III-B2 did not perform very well using an interpolated table. This is understandable as only mutation is used in the experiments. Since the mutations are random and independent of each other, it is hard for a table to discover large scale features accurately by this method. In this paper use of the interpolated table is therefore confined to TDL.

2) A Multi-layer Perceptron: The Multi-Layer Perceptron (MLP) is a neural network used widely in machine learning and AI applications. In the case of these experiments, an MLP with one hidden layer is used. The configuration can be succinctly described as a (2, 6, 1) MLP, with two inputs, six hidden units, and one output unit. This corresponded to a rule of thumb used in initial unpublished experiments that the number of hidden units was twice the sum of the number of inputs and outputs. While this is an arbitrary choice, it was found to give reasonable results in terms of performance and speed of evaluation. A $tanh()$ activation function is used for the network units. Adjustment of the weights by reinforcement is performed using the standard

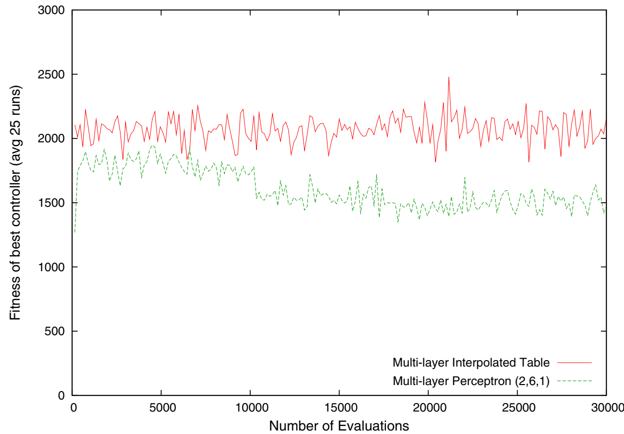

Fig. 3. Comparison of the interpolated state table and MLP using TDL. While the performance of the state table stays almost constant, the MLP shows a slow decline.

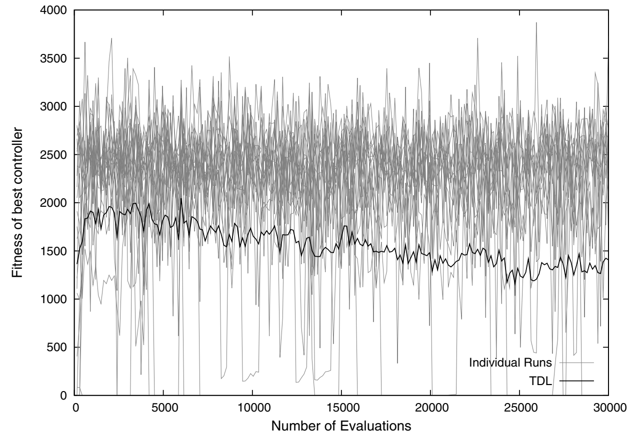

Fig. 4. Individual TDL runs (in grey), showing gradual decline in the average (thick black line). This shows the rapid drop to zero performance of an increasing number of runs over time.

Backpropagation algorithm.

IV. RESULTS

In order to directly compare the performance of evolution and TDL, the following graphs plot the 'fitness of the best controller' against the 'number of evaluations'. The fitness corresponds to the average score of the controller - which in the case of evolution is the highest scoring member of the population. In the case of TDL, only one controller is used at a time, and thus it is this controller's average score which is plotted. The number of evaluations corresponds to the number of games played up to that point in the run. In the case of evolution, this is the total number of games played by all members of the population.

A. *Function Approximators*

An interpolated table and an MLP (as described in Section III-C) were both used to learn to play Ms. Pac-Man with TDL learning. Figure 3 shows the results. Both function approximators have fairly similar performance, although the interpolated table fares slightly better. It is also notable that after an initial (quick) learning period, the performance of the MLP actually declines slightly over time.

The results of comparing the two function approximators are roughly what was expected, which was that they would have similar performance. What is interesting is that there is a slight decline in MLP performance as more games are played. The most likely explanation for this is that TDL with the MLP tends to be slightly unstable under the simple reward scheme used.

Because local changes in the value function can change the function globally, it is possible for the MLP to reconfigure so that being close to pills has a low value and being far away from pills has a high value. This will discourage the controller from eating pills, and it will therefore not see any positive rewards that might change the value function back to a more sensible configuration. It is harder to get out of this state than to get into it, and it can happen at any time during the run. The average performance of many runs therefore appears to decline smoothly as more of the runs drop into a state where their score is approximately zero.

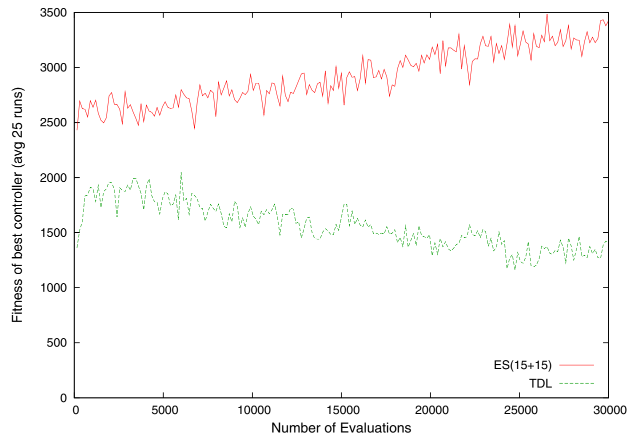

Fig. 5. Comparison of evolution and TDL using the MLP. TDL shows the same trend as in Fig. 3, whereas evolution shows a consistent improvement in performance over time.

Figure 4 shows both the average for a TDL MLP run, and the individual runs. While a very busy graph, it is possible to make out points where performance of a particular run will drop abruptly to zero. It should be noted that these runs tend to stay at zero, making them hard to see on the graph. This is why the average drops despite most of the grey lines appearing to stay around a fitness of 2500.

B. *Evolution vs. TDL*

The same MLP was used to explore the relative performance of evolution and TDL. No learning occurred during an episode in the case of evolution. As can be seen from Figure 5, evolution performed better than TDL, which once again showed a slight decline as it played more games. Evolution on the other hand steadily increased in performance.

It is interesting that evolution performs significantly better than TDL in this particular experimental configuration. As

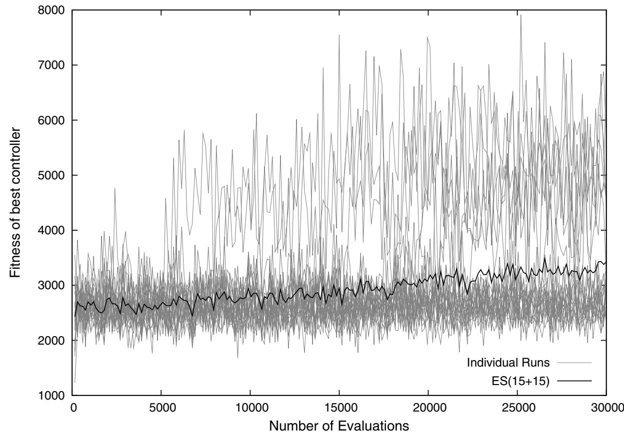

Fig. 6. Individual evolution runs (in grey), showing a gradual increase in the average (thick black line). This is due to increasing numbers of runs finding a high-performing configuration (fitness of ∼5000).

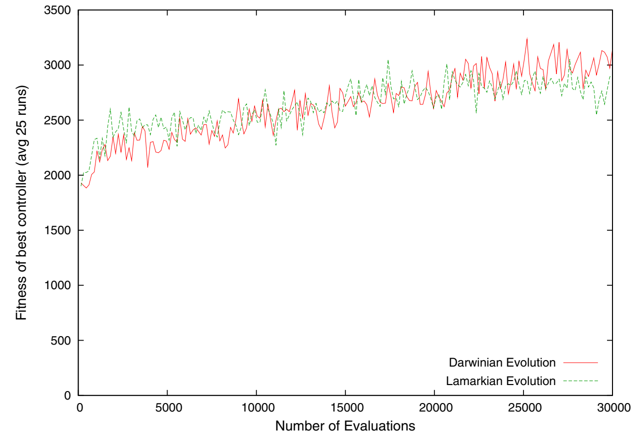

Fig. 8. Comparison of Darwinian and Lamarkian evolution. There was no significant difference in performance between the two.

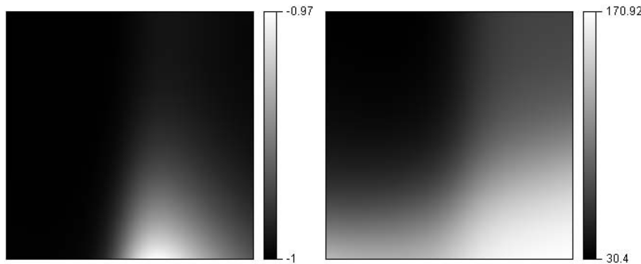

Fig. 7. Feature space of best performing evolved (left) and TDL (right) controllers. The horizontal axis is Feature 1 – further to the left means Ms. Pac-Man is closer to being cut off by the ghosts. The vertical axis is Feature 2, higher indicates further from a pill. While the two spaces have different value ranges, this is unimportant for the comparison. The action of the controller is determined by relative values of the different resulting states, so only the form of the two spaces is important.

noted in Section IV-A above, the performance of TDL of the MLP initially rises and then drops off slowly. In the case of evolution however, there is a continuous rise in performance.

In a similar way to the explanation of the continuous drop in performance in the TDL case, the continuous rise in the case of evolution seems to be due to progressively more of the runs finding a better configuration, causing performance to jump to ∼5000 (see Figure 6).

It is unknown why TDL fails to find this better form of the value function – it's possible this may be due to the reward structure or the features used. Other work on learning in noisy game environments (Runarsson & Lucas [21] and Lucas & Runarsson [22]) has shown that TDL may fail to converge in practice, depending on the set up of the experiment. It is possible something similar is happening here.

Figure 7 shows the feature space of the best performing evolved and TDL-trained controllers. The evolved controller on the left averaged a score of approximately 5500, whereas the TDL-trained controller on the right averages approximately 2500. Clearly the form of the state value function found by evolution performs much better, although it is not clear why. Observing the two controllers playing does not show up any obvious behavioural difference.

C. Comparison of Darwinian and Lamarkian evolution

Figure 8 compares Darwinian and Lamarkian evolution, again using the same size MLP. The figure shows that there is no significant difference in performance between the two schemes. This can be explained by the fact that evolution performs better than TDL on the current setup. It would therefore be expected that the addition of TDL learning would not improve performance significantly, and thus the two schemes perform equally well.

D. Variation of reward structure for TDL

The reward structure for TDL was varied by hand to a range of values. Figure 9 shows the performance of the MLP for some selected reward values. The relative rewards gained when eating a pill and being killed were varied. K represents the reward for being killed (which is negative!), and P is the reward for eating a pill. Interestingly, the best performance was seen in the case where the positive reward for eating a pill was greater than or similar in magnitude to the negative reward for being eaten. As the magnitude of the reward for being eaten became larger relative to the pill reward, performance declined.

When originally choosing values by hand for the TDL rewards, the first author chose a reward of $+1$ for eating a pill, and -100 for being eaten. It was expected that having a large penalty for being eaten would improve performance, as obviously it is hard to score highly if Ms. Pac-Man is quickly eaten.

However, Figure 9 clearly shows that the performance of these values is below that of configurations with a higher relative reward for eating a pill. Those configurations should encourage the controller to value eating pills higher than being cautious and staying away from the ghosts – leading to more daring play. That this causes an increase in performance is interesting.

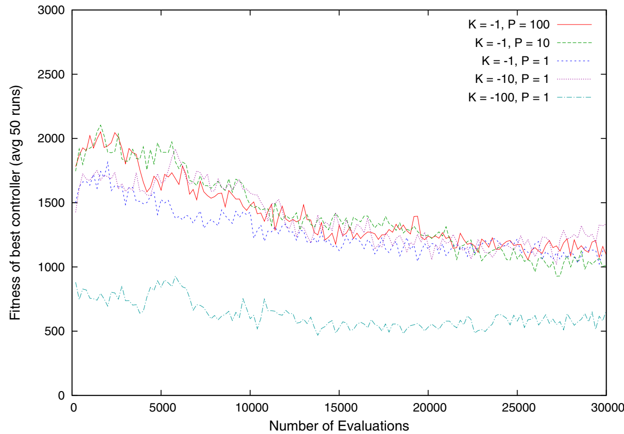

Fig. 9. Performance of the MLP with different TDL reward schemes. K is the reward for being killed, P is the reward for eating a pill. After some initial differences in learning speed, all but one of the reward schemes performed the same. The only exception was the scheme K=-100, P=1, which performed much worse.

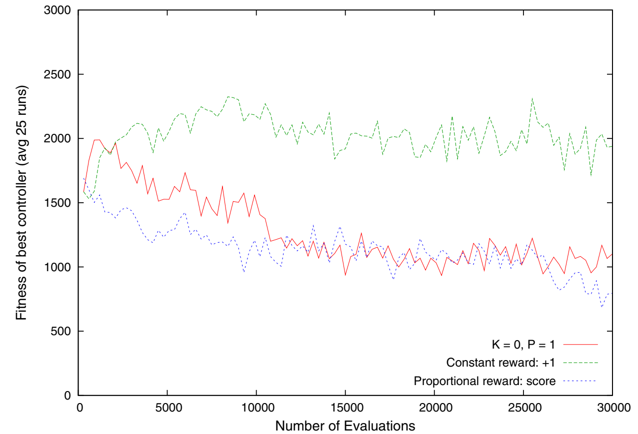

Fig. 10. Performance of the MLP with some simple TDL reward schemes, most notably a constant reward of $+1$, which avoided the slow decline of the other schemes.

It's possible this could be due to the features chosen to be presented to the controller, especially the first feature consisting of the relative distance to an escape node. Because of the way this feature is calculated, if Ms. Pac-Man moves slightly the ghosts could be predicted to come from different directions. The nearest escape node might then be in a different place. This can cause the controller to oscillate between a small number of contiguous nodes, and thus eventually be eaten. Encouraging the controller to be less cautious about reducing its relative distance to the ghosts could mean that this problem is alleviated.

The same set of reward schemes was tried with the interpolated table, and yielded very similar results which therefore have not been shown. The only differences from Figure 9 were that there was no long-term decline in performance (as discussed in Section IV-A), and that average performance of the scheme K=-100, P=1 was worse at $\sim$100 rather than $\sim$500.

Also, some simpler reward structures were tried with the MLP. The first of these was a reward of $+1$ for each pill eaten, and no reward (i.e. a reward of zero) for being eaten by a ghost. The other two schemes tried were a constant reward of $+1$ every time step, and a reward proportional to the increase in game score for each action. Figure 10 shows the results for these three simple schemes.

This final experiment showed an interesting result – namely that the series with a constant $+1$ reward per time step showed no decline in performance over time. In the case of this reward scheme, it seems that none of the individual runs get caught in the low performance configuration described in Section IV-A. This is presumably because the reward scheme means it is impossible to get into a situation where Ms. Pac-Man avoids events that can give a positive reward.

V. Conclusions

These experiments have shown that under the conditions described, evolution of the MLP currently outperforms TDL. Also, comparison on TDL between the MLP and interpolated table has shown that the state table has more reliable performance than the MLP. While the MLP can match the performance of the interpolated table under certain reward schemes, many choices of reward scheme show a gradual decline in performance.

The brief comparison of Darwinian and Lamarkian evolution presented here is less useful than expected, presumably due to the difference in performance attainable by evolution and TDL. It would be interesting to re-run this comparison when performance is more similar, through adjustment of the experiments to improve TDL performance.

It has been shown that using a $+1$ constant reward allows the TDL performance of the MLP to match the interpolated table, and avoid the gradual decline in performance seen in other schemes. It is interesting that this reward scheme outperforms schemes that are more complex, and encapsulate more information about the game (such as being eaten is bad, eating pills is good). This is possibly due to the interaction of feature choice and reward scheme.

It would be an interesting opportunity for further work to try combinations of the reward structures tested in this paper. One example might be a reward scheme with a constant positive reward, but also with an extra reward proportional to any score increase. This might also shed further light on why TDL fails to discover the better-performing configurations of the feature space.

There are several possible factors that might contribute to TDL failing to learn these better-performing configurations, the improvement of which would be a useful opportunity for further work:

- Simulation of the next state is only an approximation – the ghost and pill positions are not updated.
- The choice of features used for learning – especially the fact that a small change in state can cause a large leap

in feature space.
- The form of the reward scheme, including its interaction with feature selection.

It is unknown at this time which (combination?) of these is responsible. This unanswered question is possibly the most interesting to come out of this work.

Acknowledgements

This research is partly funded by a postgraduate studentship from the Engineering and Physical Sciences Research Council.

References

[1] L. J. Fogel, A. J. Owens, and M. J. Walsh, *Artificial Intelligence through Simulated Evolution*. New York, USA: John Wiley, 1966.
[2] J. H. Holland, *Adaptation in natural and artificial systems*. Ann Arbor: University of Michigan Press, 1975.
[3] X. Yao, "Evolving artificial neural networks," *Proceedings of the IEEE*, vol. 87, no. 9, pp. 1423 – 1447, 1999.
[4] E. Alba, J. F. A. Montes, and J. M. Troya, "Full automatic ANN design: A genetic approach," in *IWANN '93: Proceedings of the International Workshop on Artificial Neural Networks*. London, UK: Springer-Verlag, 1993, pp. 399–404.
[5] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, 2002.
[6] M. Mitchell, *An Introduction to Genetic Algorithms*. MIT Press, 1996.
[7] A. L. Samuel, "Some studies in machine learning using the game of checkers." *IBM Journal on Research and Development*, vol. 3, no. 3, pp. 210–229, July 1959.
[8] D. Michie, "Trial and error," in *Science Survey, Part 2*. Penguin, 1961, pp. 129–145.
[9] G. Tesauro, "Practical issues in temporal difference learning," in *Advances in Neural Information Processing Systems*, J. E. Moody, S. J. Hanson, and R. P. Lippmann, Eds., vol. 4. Morgan Kaufmann Publishers, Inc., 1992, pp. 259–266.
[10] ——, "TD-Gammon, a self-teaching backgammon program, achieves master-level play," *Neural Computation*, vol. 6, no. 2, pp. 215–219, March 1994.
[11] ——, "Temporal difference learning and TD-Gammon," *Commun. ACM*, vol. 38, no. 3, pp. 58–68, 1995.
[12] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998.
[13] S. Whiteson and P. Stone, "Evolutionary function approximation for reinforcement learning," *Journal of Machine Learning Research*, vol. 7, pp. 877–917, May 2006.
[14] S. M. Lucas and J. Togelius, "Point-to-point car racing: an initial study of evolution versus temporal difference learning," in *IEEE Symposium on Computational Intelligence and Games*, 2007, pp. 260–267.
[15] M. Gallagher and M. Ledwich, "Evolving Pac-Man players: Can we learn from raw input?" in *IEEE Symposium on Computational Intelligence and Games*, 2007, pp. 282–287.
[16] M. Wittkamp, L. Barone, and P. Hingston, "Using NEAT for continuous adaptation and teamwork formation in pacman," in *IEEE Symposium on Computational Intelligence and Games*, 2008.
[17] S. M. Lucas, "Evolving a neural network location evaluator to play Ms. Pac-Man," in *IEEE Symposium on Computational Intelligence and Games*, 2005, pp. 203–210.
[18] I. Szita and A. Lörincz, "Learning to play using low-complexity rule-based policies: Illustrations through Ms. Pac-Man," *J. Artif. Intell. Res. (JAIR)*, vol. 30, pp. 659–684, 2007.
[19] S. M. Lucas, "Ms. Pac-Man competition," *SIGEVOlution*, vol. 2, no. 4, pp. 37–38, 2007.
[20] ——, "Reinforcement learning with interpolated table functions," in *UK Workshop on Computational Intelligence*, 2008, p. ??
[21] T. P. Runarsson and S. M. Lucas, "Coevolution versus self-play temporal difference learning for acquiring position evaluation in small-board Go," *IEEE Transactions on Evolutionary Computation*, vol. 9, pp. 628–640, 2005.
[22] S. M. Lucas and T. P. Runarsson, "Temporal difference learning versus co-evolution for acquiring Othello position evaluation," in *IEEE Symposium on Computational Intelligence and Games*, 2006, pp. 52–59.