Temporal Difference Learning Versus Co-Evolution for Acquiring Othello Position Evaluation

Simon M. Lucas
Department of Computer Science
University of Essex, Colchester, UK
sml@essex.ac.uk

Thomas P. Runarsson
Science Institute
University of Iceland, Iceland
tpr@hi.is

Abstract—This paper compares the use of temporal difference learning (TDL) versus co-evolutionary learning (CEL) for acquiring position evaluation functions for the game of Othello. The paper provides important insights into the strengths and weaknesses of each approach. The main findings are that for Othello, TDL learns much faster than CEL, but that properly tuned CEL can learn better playing strategies. For CEL, it is essential to use parent-child weighted averaging in order to achieve good performance. Using this method a high quality weighted piece counter was evolved, and was shown to significantly outperform a set of standard heuristic weights.

Keywords: Othello, temporal difference learning, co-evolution.

I. INTRODUCTION

Both Temporal Difference Learning (TDL) and Co-Evolutionary Learning (CEL) are able to acquire game strategies without reference to any expert knowledge of game strategy, and without using any prior available player to train against. Typically, CEL achieves this by generating an initial random population of strategies which are then played against each other, with the parents for each successive generation being chosen on the basis of their playing ability. Standard TDL achieves this through self-play.

The main difference between the two methods (at least in their most typical forms) is that CEL uses only the end information of win/lose/draw aggregated over a set of games, whereas TDL aims to exploit all the information during the course of a game, as well as at the end of each game when the final rewards are known. Comparisons of this kind are both timely and important, since recent years have seen an explosion of interest in the CEL method, while applications of TDL to the same problem have been less numerous.

In a recent paper [9] the authors investigated temporal difference learning versus co-evolution for learning small-board Go strategies. There it was found that TDL learned faster, but that with careful tuning, CEL eventually learned better strategies. In particular, with CEL it was necessary to use parent-offspring weighted averaging in order to cope with the effects of noise. In this paper a similar set of experiments for Othello are reported and it is found that similar results hold, but to an even greater extent. In particular, without parent-child averaging, CEL performs very poorly. When properly tuned, however, CEL eventually finds strategies that significantly outperform a standard heuristic player [11], and also the best strategies found by TDL.

The game playing strategies are encapsulated in the weights of a weighted piece counter (WPC). Each game is played by using a 1-ply minimax search, with the WPC being used to estimate the value of the game-board after each possible move from the current board.

The paper is organised as follows. In section II a brief description of the game Othello is given and some of the more notable research on learning game strategies for Othello listed. In section III the implementation of TDL and CEL is described in full detail. This is followed by an extensive set of experimental results and evaluation of players learned in section IV. The paper is then concluded with a discussion and summary of main findings.

II. OTHELLO

The game of Othello is played on an $8 \times 8$ board, with a starting configuration as shown in fig. 1 with the middle 4 squares occupied. Black plays first, and the game continues until the board is full (after 60 turns), or until neither player is able to move. Note that a player must move if able to, passing only happens when a player has no legal moves available.

A legal move is one which causes one or more opponent counters to be flipped. Counters are flipped when they lie on a continuous line (horizontal, vertical, or diagonal) between the newly placed counter, and another counter of the placing player. Counters placed in one of the four corners can never satisfy this condition, and can therefore never be flipped. Hence, the corners play a pivotal role in the game, and

![Fig. 1. The initial Othello board, showing the four possible first moves, which are all equivalent under reflection and rotation (black moves first).](image-url)
to play many games. This is particularly true for CEL, which
should be possible.

further improvement in the performance of evolved spatial
on parent-child averaging. These considerations suggest that
take hundreds of thousands of games, and relies heavily
results below show that optimal tuning of such WPCs can
WPC used in the current paper has only
they evolved with around one hundred thousand games. The
for playing checkers [3], and had a dedicated input unit for
can be made. The speed of each game naturally depends on
the type of player. A multi-layered perceptron (MLP) with
hidden units is necessarily slower than a WPC, for example.
Regarding the WPC, there is a trick implemented for the Java
version, which evaluates only the difference in evaluation
score that a move would make, without actually making the
move. This is however, only applicable for a WPC using 1-
ply lookahead. This means that for WPCs the Java version
is the fastest implementation we have, and is able to play
around 1,500 games per second. The C version plays around
1,200 games per second using WPCs, but in the case of
greater ply search and MLPs it is faster than Java. At present,
the Java MLP implementation is not particularly efficient,
and in this mode is only able to manage around 60 games
per second, compared with 500 games per second for the C
version.

Each board is represented as a 10 × 10 array of int., with
a border of ‘off-board’ values surrounding the 8 × 8 board.
This enables efficient checking of off-board positions for
line-search termination, which is much faster than catching
ArrayOutOfBoundsException exceptions, or than explicitly
checking the range constraints.

It would be interesting to investigate the use of a bit-board
representation for further speeding up the Othello engine, as
described by Cowling [5] for the Virus game. This could
conceivably lead to a significant speed increase for weighted
piece counter players, but would make little difference for
more complex players (such as MLPs).

A. Co-Evolution

A number of different versions of the evolution strategy
(ES) as implemented in [9] were tried. It was decided
that the more simplified version, described in fig. 3, was
sufficient for the game Othello. This is the so called (1, λ) ES
using arithmetical averaging between the parent and the best
offspring. In this algorithm the parent is deleted at every
generation (non-elitist). The λ offspring play a single game
against one another both as Black and White. This results
in a total of λ(λ − 1) games per generation. The parent-
child averaging is a standard evolution strategy technique
for dealing with noisy fitness functions. Pollack and Blair [8]
also used averaging when using a random hill-climber (i.e. a
(1 + 1) ES) to successfully learn backgammon strategy, but
for the current paper a (1 + 1) ES with averaging performed
poorly (though without averaging it performed even worse).

Regarding the win/draw/lose payoffs listed in fig. 3 caption,
we also experimented with basing fitness solely on the
number of wins, but found that these different payoffs did

valuing them highly tends to be the first thing learned, a fact
that can be seen easily by inspecting the evolution of weight
values in a WPC. Indeed the WPC [11] used as a benchmark
in that study also reflects this. There the highest value of 1
is given to all four corners. To hinder the possibility of an
opponent getting a corner, the squares next to them should
be avoided. For this reason they are given the lowest value
−0.25. As a consequence the WPC encourages the players
to place its counter at advantageous squares. The total set
of weights for this heuristic player is given in fig. 2. Note
that the weights of this heuristic player are symmetric under
reflection and rotation, and have just 10 distinct values out of
a possible 64. It would be possible to simplify the learning
task by enforcing this kind of symmetry, and evolving just
10 parameters instead of 64. This would mean building in
more expert knowledge however, and could also place
undesirable constraints on the value function. Indeed, the
best weights evolved in this paper are not symmetric (see
Table 10). A direct comparison of learning the reduced set
of 10 weights compared with learning the full 64 weights
would be interesting future work.

<table>
<thead>
<tr>
<th>Weight</th>
<th>1.00</th>
<th>-0.25</th>
<th>0.10</th>
<th>0.05</th>
<th>0.05</th>
<th>0.10</th>
<th>-0.25</th>
<th>1.00</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>-0.25</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
<td>-0.25</td>
<td>-0.25</td>
</tr>
<tr>
<td></td>
<td>0.10</td>
<td>0.01</td>
<td>0.05</td>
<td>0.02</td>
<td>0.02</td>
<td>0.05</td>
<td>0.01</td>
<td>0.10</td>
</tr>
<tr>
<td></td>
<td>0.05</td>
<td>0.01</td>
<td>0.02</td>
<td>0.01</td>
<td>0.01</td>
<td>0.02</td>
<td>0.01</td>
<td>0.05</td>
</tr>
<tr>
<td></td>
<td>0.05</td>
<td>0.01</td>
<td>0.02</td>
<td>0.01</td>
<td>0.01</td>
<td>0.02</td>
<td>0.01</td>
<td>0.05</td>
</tr>
<tr>
<td></td>
<td>0.10</td>
<td>0.01</td>
<td>0.05</td>
<td>0.02</td>
<td>0.02</td>
<td>0.05</td>
<td>0.01</td>
<td>0.10</td>
</tr>
<tr>
<td></td>
<td>-0.25</td>
<td>-0.25</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
<td>-0.25</td>
<td>-0.25</td>
</tr>
<tr>
<td></td>
<td>1.00</td>
<td>-0.25</td>
<td>0.10</td>
<td>0.05</td>
<td>0.05</td>
<td>0.10</td>
<td>-0.25</td>
<td>1.00</td>
</tr>
</tbody>
</table>

Fig. 2. The weights ($w$) for the heuristic player [11].

The first strong learning Othello program developed was
Bill [6], [7]. Later, the first program to beat a human
champion was Logistello [2], the best Othello program from
1993–1997. Logistello also uses a linear weighted evaluation
function but with more complex features than just the plain
board. Nevertheless, the weights are tuned automatically
using self-play. Logistello also uses an opening book based
on over 23,000 tournament games and fast game tree search
[1].

More recently, Chong et al [4] co-evolved a spatially aware
multi-layer perceptron (MLP) for playing Othello. Their
MLP was similar to the one used by Fogel and Chellapilla
for playing checkers [3], and had a dedicated input unit for
every possible sub-square of the board. Together with the
hidden layers this led to a network with 5,900 weights, which
they evolved with around one hundred thousand games. The
WPC used in the current paper has only 64 weights. The
results below show that optimal tuning of such WPCs can
take hundreds of thousands of games, and relies heavily
on parent-child averaging. These considerations suggest that
further improvement in the performance of evolved spatial
MLPs should be possible.

III. IMPLEMENTATION

In order to achieve effective learning it may be necessary
to play many games. This is particularly true for CEL, which
may require hundreds of thousands of games in order to
achieve good performance. Indeed, the experimental work
underlying this paper involved the running of several billion
games of Othello. Therefore, the efficiency of the game
engine plays an important part in this research.

We developed two implementations of all the software,
one written by the first author in Java, the other by the
second author in C. In this way all results are double-checked
and enabled a speed comparison of each implementation to
be made. The speed of each game naturally depends on
the type of player. A multi-layered perceptron (MLP) with
hidden units is necessarily slower than a WPC, for example.

Regarding the WPC, there is a trick implemented for the Java
version, which evaluates only the difference in evaluation
score that a move would make, without actually making the
move. This is however, only applicable for a WPC using 1-
ply lookahead. This means that for WPCs the Java version
is the fastest implementation we have, and is able to play
around 1,500 games per second. The C version plays around
1,200 games per second using WPCs, but in the case of
greater ply search and MLPs it is faster than Java. At present,
the Java MLP implementation is not particularly efficient,
and in this mode is only able to manage around 60 games
per second, compared with 500 games per second for the C
version.

Each board is represented as a 10 × 10 array of int., with
a border of ‘off-board’ values surrounding the 8 × 8 board.
This enables efficient checking of off-board positions for
line-search termination, which is much faster than catching
ArrayOutOfBoundsException exceptions, or than explicitly
checking the range constraints.

It would be interesting to investigate the use of a bit-board
representation for further speeding up the Othello engine, as
described by Cowling [5] for the Virus game. This could
conceivably lead to a significant speed increase for weighted
piece counter players, but would make little difference for
more complex players (such as MLPs).

A. Co-Evolution

A number of different versions of the evolution strategy
(ES) as implemented in [9] were tried. It was decided
that the more simplified version, described in fig. 3, was
sufficient for the game Othello. This is the so called (1, λ) ES
using arithmetical averaging between the parent and the best
offspring. In this algorithm the parent is deleted at every
generation (non-elitist). The λ offspring play a single game
against one another both as Black and White. This results
in a total of $\lambda(\lambda - 1)$ games per generation. The parent-
child averaging is a standard evolution strategy technique
for dealing with noisy fitness functions. Pollack and Blair [8]
also used averaging when using a random hill-climber (i.e. a
(1 + 1) ES) to successfully learn backgammon strategy, but
for the current paper a (1 + 1) ES with averaging performed
poorly (though without averaging it performed even worse).

Regarding the win/draw/lose payoffs listed in fig. 3 caption,
we also experimented with basing fitness solely on the
number of wins, but found that these different payoffs did
not lead to a significant difference in final playing quality. An alternative that was not investigated would be to base the fitness function on the piece difference at the end of each game.

The WPC ($\mathbf{w}$) co-evolved in this manner is described by:

$$f(x) = \sum_{i=1}^{8\times8} w_i x_i + x_0$$

(1)

where $x_i$ is the value at square $i$ on the board, which is 0 when Empty, 1 if Black, and $-1$ for White. The bias term $x_0$ is set to zero for the CEL runs. The single scalar output of function $f(x)$ is interpreted as follows. The value indicates which position is most favorable for a particular player, with larger values favoring Black, and smaller values White.

1. Initialize: $\mathbf{w}^0 = 0$ and $\beta = 0.05$ (or 1.0)
2. While termination criteria not satisfied do
3. for $k := 1$ to $\lambda$ do (replication)
4. $\mathbf{w}_k \leftarrow \mathbf{w}^0 + N(0,1/n)$
5. od
6. Each individual $\mathbf{w}_k$, $k = 1, \ldots, \lambda$ plays another (once each color) for a total of $\lambda(\lambda - 1)$ games,
7. Find the player $i$ with the highest score (breaking ties randomly)
8. $\mathbf{w}^* \leftarrow \mathbf{w}^* + \beta((\mathbf{w}_i - \mathbf{w}^*)$ (arithmetic average)
9. od

Fig. 3. The $(1, \lambda)$ evolution strategy. For each win the player receives a score of 1, 0 for a draw, and $-2$ for loss.

It is also possible to force random moves during game play. The experimental studies show that this slows down learning, but may lead to slightly better strategies in the long run. The best player found in this paper was evolved with forced random play.

B. Temporal Difference Learning

In TDL the weights of the evaluation function are updated during game play using a gradient-descent method. Let $x$ be the board observed by a player about to move, and similarly $x'$ be the board after the player has moved. Then the evaluation function may be updated during play as follows [10, p.199]:

$$w_i \leftarrow w_i + \alpha [v(x') - v(x)] \frac{\partial v(x)}{\partial w_i}$$

(2)

$$v(x) = \tanh(f(x)) = \frac{2}{1 + \exp(-2f(x))} - 1$$

(3)

is used to force the value function $v$ to be in the range $-1$ to 1. This method is known as gradient-descent TD(0) [10]. If $x'$ is a terminal state then the game has ended and the following update is used:

$$w_i \leftarrow w_i + \alpha [r - v(x')] (1 - v(x')^2) x_i$$

where $r$ corresponds to the final utilities: +1 if the winner is Black, $-1$ when White, and 0 for a draw.

The update rule is perhaps the simplest version of temporal difference learning and works quite well on this task. If the step size parameter $\alpha$, in (4), is reduced properly over time this method will also converge [10, p. 13]. During game play, with probability $\epsilon = 0.1$, a random or exploratory move is forced. This is known as an $\epsilon$-greedy policy and is described in fig 4. Note that, TD(0) is attempting to learn the probability of winning from a given state (when following the $\epsilon$-greedy policy), while the ES is only learning the relative ordering of the set of game states.

To satisfy our curiosity, a TDL run with no noise (i.e. $\epsilon = 0$) was tried. In this case, every run is deterministic, with all weight-values initialised to zero, and a deterministic tie-break policy being used (always picking the first encountered move among a set of equal moves). Under these conditions, exactly the same sequence of players is always produced. While this approach did not produce the best TDL players, it did nonetheless produce quite reasonable players. The interesting point here is that the dynamics of the game, and of the weight updates are sufficient to produce a large degree of game strategy exploration.

IV. EXPERIMENTAL RESULTS

For each learning method, some time was spent experimentally tuning the parameters in order to get best performance. The critical factor for TDL is the update rate $\alpha$, while for CEL it is the population size $\lambda$ and smoothing factor $\beta$. In our previous study for Go [9] it was found that a population size of $\lambda = 30$ was needed for a $5 \times 5$ Go board and a setting of $\beta = 0.05$ was necessary. Similar finding are observed here for Othello, however, smaller population sizes are adequate, resulting in much faster CEL learning for Othello than for Go. For TDL an initial step size of $\alpha = 0.01$ worked best, which was then reduced by a factor of 0.95 every 45,000 games played.

A. Evaluation

Each experiment is repeated independently 30 times, with the average and standard deviations reported. During CEL and TDL the players are evaluated by playing against a standard heuristic player (see fig. 2) and a random player at 1-ply. This is repeated 10,000 times for each point on the graphs (the player under test plays 5,000 games as White and 5,000 games as Black).

Secondly, leagues of learned players are played against each other. It is interesting to note that which player is regarded as best depends on the chosen evaluation method. Evaluation against a fixed opponent can give a very quick guide to a player’s ability, but what usually matters more is how well the player fares against a wide variety of players.
In order to get a good measure of a player’s ability, players are forced to make a random move with probability $\epsilon = 0.1$, as shown in fig. 4. This is the same policy used during TDL, and for CEL with noise. Note that when playing two players against each other multiple times with forced random play, the expectation of a particular player winning follows a Bernoulli distribution. If the player has a true probability $p$ of winning, then the variance of $p$ is given by $p(1-p)$. When estimating $p$ from $n$ games, this allows confidence intervals to be placed. The standard error (i.e. the standard deviation of the mean) is given by $\sqrt{p(1-p)/n}$.

Strictly speaking, when forcing the players to make occasional random moves, the game is no longer truly Othello, but a slightly randomized version of it. Nonetheless, it seems likely that playing ability for the randomized game will be highly correlated with playing ability for the true game.

B. Co-evolution

For the CEL runs the following experimental results are presented:

a. (1, 10) ES using $\beta = 1.0$ (no arithmetical averaging). This will illustrate the necessity of using such an average. See fig. 5 and fig. 6.

b. (1, 10) ES with $\beta = 0.05$. See fig. 5 and fig. 6.

c. (1, 10) ES with $\beta = 0.05$ and forced random play with probability $\epsilon = 0.1$. See fig. 5 and fig. 6.

d. experiments b. and c. are repeated with a population size of $\lambda = 5$ (a population size greater than 10 did not show significant improvement in performance). The performance statistic for these runs are only given for when playing against the heuristic player. See fig. 7.

In fig. 5 the average (with one standard deviation) for the 30 independent runs playing 10,000 games against the heuristic player is shown, for experiments a., b., and c. It is interesting to note that without the smoothing ($\beta = 0.05$), the results are significantly poorer. Also, initially, an evolution without forced random moves performs better, but eventually the evolution with forced random moves achieves a slightly higher level of play. Similar results are observed when playing against a purely random player. These results are depicted in fig. 6.

The results for experiment d., the (1, 1) ES, are presented in fig. 7. Similar trends are observed as for the (1, 10) ES in fig. 5, however, the average performance against the heuristic player is now worse. The only difference here is the number of offspring produced per generation and therefore the number of games played per generation. However, the overall total number of games played remains the same.

In fig. 8 a single CEL run is compared with a single TDL run. Both of these runs are subjected to a forced random move with probability $\epsilon = 0.1$. These are snapshots of the 10,000 game performance versus the heuristic player taken every 45,000 games played during learning. It is also interesting to see if there is any significant difference between these 100 players taken as snapshots during learning. To investigate this a league was set up where these 100 players are matched up against each other without any forced random moves. Let these players be labeled as player-1 to player-100. The top three and bottom three ranking players are presented in table I. This clearly shows that the better players are found towards the end of the run and the worst in the beginning.

C. Temporal Difference Learning

For this experiment, an initial value of $\alpha = 0.01$ was used, decreasing by a factor of 0.95 every 45,000 games played. The probability of making a purely random move, $\epsilon = 0.1$. The initial weights are set to 0.0 and 30 independent runs performed. The mean results against the heuristic and random players (along with one standard deviation) is plotted in fig. 9. The ultimate performance of the TDL players against the heuristic and random players are similar, however,
there appears to be a downward trend towards the end of the runs. Recall that the step size is being reduced over time. Furthermore, when observing a single typical TDL run in Fig. 8 one can see that there is greater variation in performance against the heuristic player during learning. This difference can be observed by playing the 100 snapshot players against each other and labeling them as before, from Player-1 to Player-100. Here the top three and bottom three players may be found at any time as shown in Table II.

### Table II

A partial league of players sampled at each epoch during a TDL run, then played against each other.

<table>
<thead>
<tr>
<th>Place</th>
<th>Played</th>
<th>Won</th>
<th>Drew</th>
<th>Lost</th>
<th>Player</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>198</td>
<td>134</td>
<td>4</td>
<td>60</td>
<td>Player-68</td>
</tr>
<tr>
<td>2</td>
<td>198</td>
<td>130</td>
<td>6</td>
<td>62</td>
<td>Player-64</td>
</tr>
<tr>
<td>3</td>
<td>198</td>
<td>129</td>
<td>5</td>
<td>64</td>
<td>Player-2</td>
</tr>
<tr>
<td>98</td>
<td>198</td>
<td>60</td>
<td>6</td>
<td>66</td>
<td>Player-2</td>
</tr>
<tr>
<td>99</td>
<td>198</td>
<td>66</td>
<td>5</td>
<td>127</td>
<td>Player-5</td>
</tr>
<tr>
<td>100</td>
<td>198</td>
<td>62</td>
<td>9</td>
<td>127</td>
<td>Player-64</td>
</tr>
</tbody>
</table>

**D. Champions League**

As a final evaluation six different WPCs are compared in a champions league. These are the heuristic WPC (see Fig. 2), the best TDL and CEL found against the heuristic player (TDL-HB and CEL-HB). Furthermore, the best out of the 30 final players found for the different CEL experiments b. and c. and TDL experiments are tested. These are called CEL, CEL-N, and TDL respectively and were found using a separate champions league among the final 30 players, each playing 10,000 games with forced random moves.

### Table III

Champions league with the selected WPCs.

<table>
<thead>
<tr>
<th>Place</th>
<th>Played</th>
<th>Won</th>
<th>Drew</th>
<th>Lost</th>
<th>Player</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10000</td>
<td>5302</td>
<td>330</td>
<td>4148</td>
<td>CEL-N</td>
</tr>
<tr>
<td>2</td>
<td>10000</td>
<td>5474</td>
<td>314</td>
<td>4212</td>
<td>CEL</td>
</tr>
<tr>
<td>3</td>
<td>10000</td>
<td>5229</td>
<td>301</td>
<td>4470</td>
<td>CEL-HB</td>
</tr>
<tr>
<td>4</td>
<td>10000</td>
<td>4875</td>
<td>308</td>
<td>4817</td>
<td>Heuristic</td>
</tr>
<tr>
<td>5</td>
<td>10000</td>
<td>4243</td>
<td>315</td>
<td>5442</td>
<td>TDL-HB</td>
</tr>
<tr>
<td>6</td>
<td>10000</td>
<td>3735</td>
<td>296</td>
<td>5969</td>
<td>TDL</td>
</tr>
</tbody>
</table>

The results are shown in Table III where these 6 different
WPCs are matched up in a round-robin (i.e. each player played every other player as black and as white) champions league, each playing 10,000 games in total with forced random moves ($\epsilon = 0.1$). The CEL players significantly outperform the other players, with the TDL players on the bottom. The champion WPC is the noisy CEL given in fig. 10.

To show how each player fares against every other player, as black and as white, the round-robin league was run again, but presented in a different way. This time, each player played every other player (including itself) 1000 times as black, and 1000 times as white. Table IV now shows the average score from the point of view of the black player (named for each row of the table), scoring 1.0 for a win, 0.5 for a draw, and 0.0 for a loss. The results again show the superiority of the CEL players over the heuristic player and over the TDL players. Since three possible outcomes are now being measured (win, lose, or draw) the Bernoulli estimate of the variance cannot be directly applied. If it were, it would give a standard error to an average score of 0.5 over 1000 games, of 0.015; this may still give an idea of the statistical significance of the table entries. One point that does seem significant, and was repeated in five runs of the same experiment, is that the best player (CEL-N) plays against itself weaker as black than as white. In the same five runs, CEL-N versus the Heuristic player always ended in a favourable score for CEL-N, either as black or as white.

V. DISCUSSION AND CONCLUSIONS

TDL was able to learn quite good strategies very rapidly on some runs within a few thousand games. CEL learned much more slowly, but eventually significantly out-played not only the TDL strategies, but also a set of standard heuristic weights.

TDL is sensitive to the setting of $\alpha$, although great care was taken in setting its value. The CEL used a simple WPC while the TDL needed to squash its values to be in the range from $-1$ to $+1$ using the $\tanh$ functions. It is possible that learning the relative ordering of the board positions is an easier task than trying to learn a value function which tells us the probability of winning at each state of the board. Clearly, a simple WPC will have difficulties approximating this value function.

A promising future development is a hybrid algorithm, where some TDL runs are used to generate an initial population for CEL. Such a hybrid might exploit the best aspects of each method: the rapid learning of TDL, and the ultimately superior strategies obtainable with CEL.

A surprising aspect of TDL is that for this problem, it could run entirely deterministically and still achieve good performance. Initially, in a perfect information game such as Othello, we supposed that random exploratory moves would be essential in order to achieve sufficient exploration of game space, or game strategy space, which is a widely held belief regarding TDL. At least for a weighted piece counter, this proved to be non-essential. When playing deterministically, the on-line weight updates, together with an arbitrary but consistent tie-breaking strategy provided sufficient exploration.

An important point to note is that for most runs, TDL failed to converge reliably to its best play, and would typically follow some chaotic performance pattern. Varying the $\alpha$ reduction rate would not cure this, since there would be no guarantee of converging to its best play. This therefore provides an important cautionary note against using TDL for a fixed number of iterations, and then taking the final set of weights. This would typically be very hit and miss. A much better approach, is to constantly monitor the performance of the learned weights during training, and then choose those which perform best. If no external agent is available for this purpose, then the best technique is to sample the weights regularly, then play all the samples against each other in the league, and finally return the winning weight vector (assuming that a single best player is required). A possible explanation for why TDL fails to converge is that the true value function for this game is a highly non-linear function of the input vector. The attempt to approximate the value function with a linear function (a weighted piece counter) is therefore doomed to fail. In a similar way, using the delta rule to train a single layer perceptron on a non-linear function such as XOR will also fail to converge.

In order to achieve an even higher level of play a deeper search than 1- ply must be used. Furthermore, more complex features, such as those used by Logistello, must be employed. One notable limitation of TDL when learning evaluation functions is that the functions must be differentiable. The WPCs used in this study satisfy this requirement, but other choices of architecture, such GP-style expression trees, would not necessarily do so (depending on the function set used). Our immediate future work is to compare TDL versus CEL for learning the parameters of more sophisticated architectures to play Othello.

In order to allow direct comparison between various learning methods and value function architectures implemented by different researchers, we are also running a web-based Othello function evaluation league\(^1\). This allows the parameters for a number of standard architectures to be submitted via an on-line form for immediate evaluation against the standard heuristic player of fig. 2. The submitted players will then participate in an Othello competition associated with the 2006 IEEE Congress on Evolutionary Computation.

Finally, perhaps the most significant conclusion of this paper is that standard co-evolution performs very poorly on this problem. To get good performance, we found the use of parent-child averaging to be essential. This was also true for small-board Go [9], and may well be true for many other games.

ACKNOWLEDGMENTS

We are grateful to the anonymous reviewers for their helpful comments. The work was supported by a visiting

\(^1\)http://algoval.essex.ac.uk:8080/othello/html/Othello.html
researcher grant for Thomas Runarsson, funded by the Computer Science Department, University of Essex.

APPENDIX

To allow others to test our best player directly, we list here the weights of our overall champion (see fig. 10). The weights are listed in row order, left to right, top to bottom (i.e. the first eight weights are for the top row).

REFERENCES

TABLE IV
Champions league showing the round-robin results. Each entry shows the score (see text) obtained by the black player averaged over 1000 games. The player named on each row plays as black, against the player named in each column playing as white.

<table>
<thead>
<tr>
<th></th>
<th>CEL-N</th>
<th>CEL</th>
<th>CEL-HB</th>
<th>Heuristic</th>
<th>TDL-HB</th>
<th>TDL</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEL-N</td>
<td>0.455</td>
<td>0.458</td>
<td>0.495</td>
<td>0.5645</td>
<td>0.582</td>
<td>0.601</td>
</tr>
<tr>
<td>CEL</td>
<td>0.486</td>
<td>0.506</td>
<td>0.4845</td>
<td>0.4435</td>
<td>0.5785</td>
<td>0.6545</td>
</tr>
<tr>
<td>CEL-HB</td>
<td>0.4265</td>
<td>0.434</td>
<td>0.534</td>
<td>0.491</td>
<td>0.582</td>
<td>0.684</td>
</tr>
<tr>
<td>Heuristic</td>
<td>0.4425</td>
<td>0.4505</td>
<td>0.4935</td>
<td>0.4975</td>
<td>0.5705</td>
<td>0.545</td>
</tr>
<tr>
<td>TDL-HB</td>
<td>0.387</td>
<td>0.4475</td>
<td>0.412</td>
<td>0.4445</td>
<td>0.4585</td>
<td>0.52</td>
</tr>
<tr>
<td>TDL</td>
<td>0.3035</td>
<td>0.3</td>
<td>0.3075</td>
<td>0.4575</td>
<td>0.5305</td>
<td>0.5155</td>
</tr>
</tbody>
</table>

4.622507 -1.477853 1.409644 -0.066975 -0.305214 1.633019 -1.050899 4.365550
-1.329145 -2.245663 -1.060633 -0.541089 -0.332716 -0.475830 -2.274535 -0.032595
2.681550 -0.906628 0.229372 0.059260 -0.150415 0.321982 -1.145060 2.986767
-0.746066 -0.317389 0.140040 -0.045266 0.236595 0.158543 -0.720833 -0.131124
-0.305566 -0.328398 0.073872 -0.131472 -0.172101 0.016603 -0.511448 -0.264125
2.777411 -0.769551 0.676483 0.282190 0.007184 0.269876 -1.408169 2.396238
-1.566175 -3.049899 -0.637408 -0.077690 -0.648382 -0.911066 -3.329772 -0.870962
5.046583 -1.468806 1.545046 -0.031175 0.263998 2.063148 -0.148002 5.781035

Fig. 10. The weights for the overall champion.