Spatial Processing Layer Effects on the Evolution of Neural Networks to Play the Game of Othello

S.Y. Lin and J.D. White, *Member, IEEE*

***Abstract*— Neural networks (NNs) were evolved to learn to play the zero-sum game Othello (also known as reversi) without relying on a-priori or expert knowledge. Such neural networks were able to discover game-playing strategies through co-evolution, where the neural networks just play against themselves across generations. The effect of the spatial processing layer on evolution was investigated. It was found that the evolutionary process was crucially dependent on the way in which spatial information was presented. A simple sampling pattern based on the squares attacked by a single queen in Chess resulted the networks converging to a solution in which the majority of networks, handicapped by playing Black and playing without using any look-ahead algorithm, could defeat a positional strategy using look-ahead at ply-depth=4 and a piece-differential strategy using look-ahead at ply-depth=6. Improvement and convergence was observed to be accompanied by an gradual increase in the survival time of neural network strategies from less than 10 generations to about 600 generations. Surprisingly, evolved neural networks had difficulty in defeating a simple mobility strategy playing at a ply-depth=2. This work suggests that in deciding a suitable way to spatially sample a board position, it is important to consider the rules of the game.**

I. Introduction

GAMES have been widely used as a testbed for self-learning techniques as there are a specific set of (1) rules that govern the game, (2) behaviors (rules for making legal moves), and (3) goals (to win), as well as, a diverse environment of players. Considerable attention has been received by zero sum board games such as checkers, chess and Othello. In the middle of 1990, Leouski and Utgoff [1] used temporal difference learning (TDL) to aid a neural network to learn to play Othello. Several years later, Fogel and coworkers [2] showed that multilayer perception (MLP) with a spatial layer can be evolved to play checkers at a high level without expert knowledge. Chong *et. al.* [3] showed that a similar architecture having a similar spatial layer could be evolved as an evaluation function for the game of Othello. In addition, they observed changes in strategies being adopted by the neural network by making use of fixed algorithms playing at a variety of ply-depths. In recent research, Lucas [4] showed that a n-tuple systems having ~15,000 weights could be used as position value functions for the game of Othello. Coupled with TDL, good performance was obtained after only 500 generations at the cost of having ~15,000 adjustable weights.

In this paper we report on the effects of replacing the spatial input layer of Chong *et. al.* [3] with one based on the rules of the game of Othello coupled with a modification of the selection method from unfair to fair tournament. We have found that under these changes successful strategies can be evolved that are equal or better than those evolved using the previous architecture. In addition, these evolved strategies require significantly fewer adjustable weights (~4700). Finally, we show that once successful strategies have been identified, there is a gradual reduction of noise and the rate of evolution slows as seen in an extension of the survival times of neural networks which slowly rises from less than 10 for the reported architecture.

I. Background

Othello is a popular zero-sum game in which two-players, designated Black and White, alternatively place their pieces on an eight-by-eight board. Starting from the initial board position in which each player has two pieces on the board, the Black player makes the first move. A legal move is one in which the new piece is placed adjacent (horizontally, vertically, or diagonally) to an opponent's existing piece in such a way that at least one of the opponent's pieces lies between one of the player's existing pieces and the new piece. The move is completed when the surrounded pieces are removed and replaced with pieces of the player's own color. The game is completed when neither player can make a legal move, which, in general, occurs when there are no empty squares remaining on the board. The winner is the player with the most pieces on the board at the end of the game. In the event that both players possess an equal number of pieces, both players are awarded a draw.

This game is an interesting test-case for evolutionary programming as, unlike in other games of skills such as checkers and chess, who is winning, as defined by the number of pieces on the board, can vary drastically from move to move.

Manuscript received October 31, 2008. This work was supported in part by the National Science Council of the Republic of China under Grant #97-2221-E-155-061

S. Y. Lin is with the Department of Electrical Engineering, Yuan Ze University, Taoyuan 320, Taiwan (e-mail: xinyu0123@gmail.com).

J. D. White is with the Department of Electrical Engineering, Yuan Ze University, Taoyuan 320, Taiwan (phone: 886-3-463-8800x7514; fax: 886-3-451-4281 e-mail: whitejd@xiaotu.com).

978-1-4244-2959-2/09/$25.00 ©

II. IMPLEMENTATION

Three modules, consisting of a neural network-based evaluation function, tournament-based selection of the next generation, and a mutation module that is applied to the neural networks, are interfaced together to simulate evolution of a population of neural networks (game-playing strategies). This evolutionary procedure is summarized in Fig. 1 for a population of 4 neural networks. The networks compete against each other in a tournament. Based on the outcome, the two losing networks are eliminated, while the top two ranked networks not only survive to the next generation, but also give birth, by mutation, to a single child each. This becomes the next generation. An additional module, including a search algorithm and a variety of fixed strategies, acts as an external observer, monitoring but not participating in, the evolutionary process.

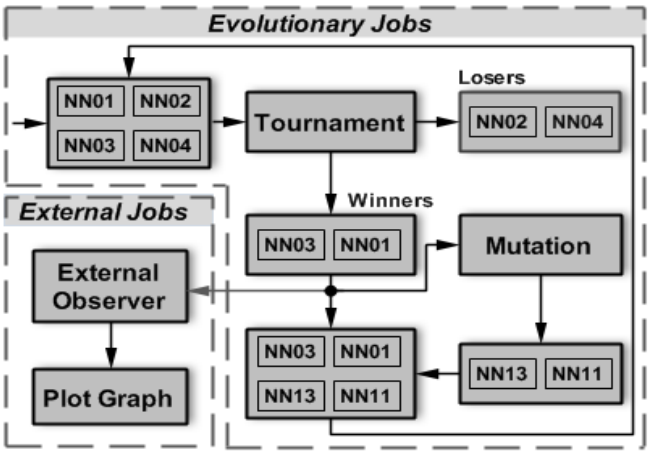

Fig. 1 Flow of evolutionary procedure illustrating key program modules.

In previous works, a search algorithm based on the Fail-Soft Alpha-Beta (FAB) Algorithm [5], a basic variant of the minimax algorithm that employs the alpha-beta pruning method, was used in conjunction with the evaluation function [2][3]. In general, this approach is used to search for the best possible move based on game heuristics (evaluations) provided by the evaluation function. In this work, during evolution, this is disabled -- the neural network based evaluation function is only provided with the present board's spatial configuration and is not allowed to look-ahead at possible future positions. For monitoring and evaluation of the evolution of the neural network based evaluation functions, it is used with a variety of fixed strategies. With each strategy, the ply-depth is gradually increased until the neural-network strategy is defeated.

The complete program was written by ANSI-C and compiled with GCC. The simulation was run for ten thousand generations on 2.4GHz Intel processor in 4 days. Each section of the program is discussed in detail below.

A. Neural Network Architecture for evaluation function

The neural network functions primarily as an artificial brain that provides strategies required to play Othello effectively through its evaluation of the board positions

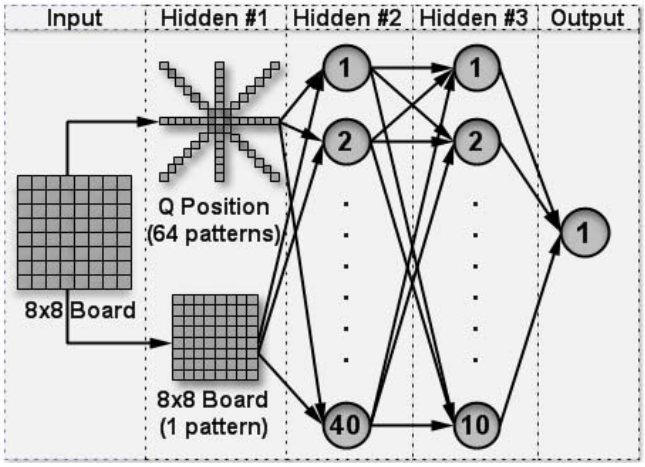

Fig. 2 Model of the neural-network-based evaluation function. Given any board pattern as a 2D vector. The first hidden layer (spatial preprocessing layer) consist of 65 output nodes, 64 output nodes from Q-position patterns, one output nodes from full board pattern which acts as weighted piece counter (WPC). The outputs of these nodes were then passed to two additional hidden layers, consisting of 40 and 10 nodes, respectively. The final output node represents the evaluation of the board position by the neural network. Unlike in previous published implementations, the piece differential is not given as an input to the output node.

presented to it. The neural network model is a feed-forward multilayer perception (MLP) based on that used by Fogel's group [2] for checkers and applied by Chong et al [3] to Othello or Reversi. The model is consists of one input layer, one output layer and three hidden layers. Fig. 2 illustrates the neural network model that is used for the evaluation

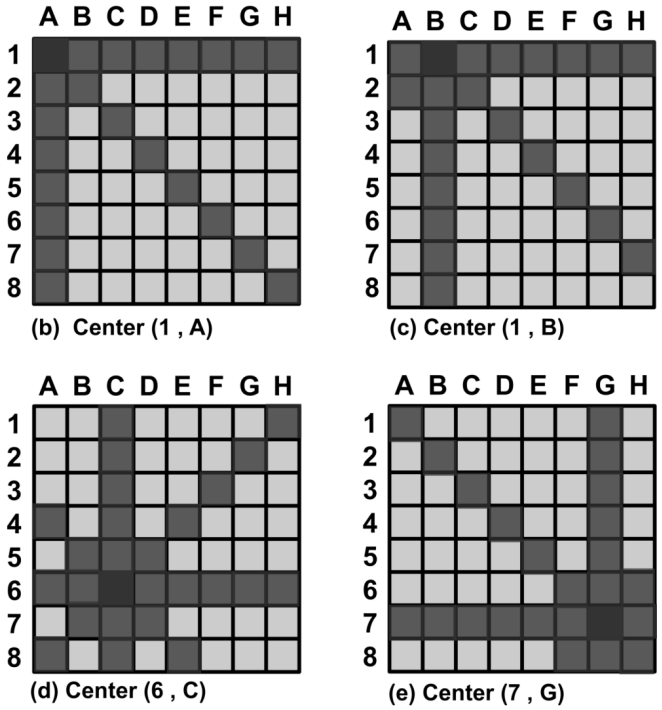

Fig. 3. Sampling procedure of Q-position pattern in the spatial layer. Every single pattern is based on the base pattern shown in Fig. 2. (a) Central Point of the base pattern mapped onto the top left corner of the Othello board (1, A). (b) Central Point of the base pattern mapped onto square (1, B). (c) Central Point of the base pattern mapped onto square (6,

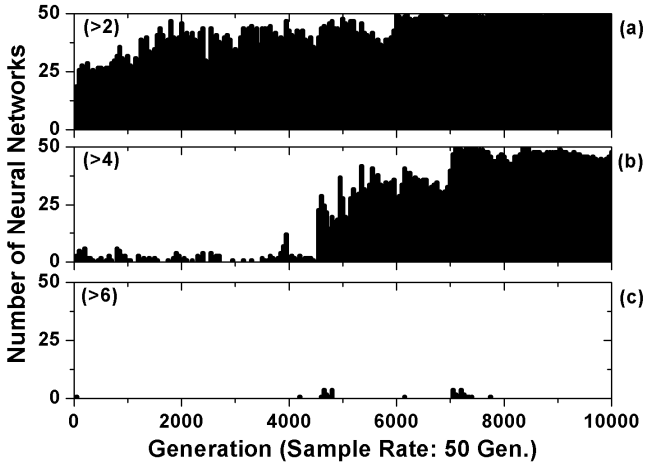

Fig. 4. Evolution in playing ability of the parent neural networks with generation. Networks were evaluated every 50 generations by playing against the fixed positional strategy player, with the ply-depth being increased in steps of two until the network lost. Number of networks that can defeat the fixed strategy player playing at ply-depth of (a) two (b) four and (c) six.

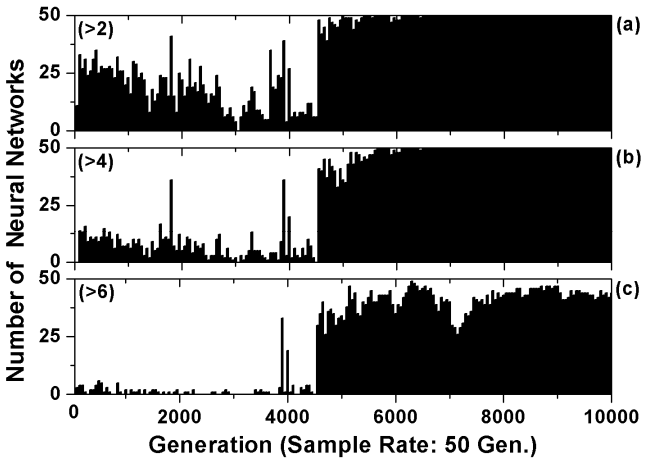

Fig. 5. Evolution in playing ability of the parent neural networks with generation. Networks were evaluated every 50 generations by playing against the fixed piece differential strategy player, with the ply-depth being increased in steps of two until the network lost. Number of networks that can defeat the fixed strategy player playing at ply-depth of (a) two (b) four and (c) six. The comparison stops at a ply-depth of six because at ply-depth of eight, the minimax algorithm was unable to run in a reasonable time.

function.

The input layer is designed to represent to the Othello board. It is an 8x8 vector consisting of 64 components to corresponding to the 8x8 Othello board positions. Each component in the 2D vector can take on the elements {-1, 1, 0}, where "-1" was the value assigned for black piece, "1" was the value assigned for white piece and "0" represented an empty square.

The first hidden layer is the spatial preprocessing layer -- designed for processing, analyzing and emphasizing the most important areas on board. The sampling pattern is the key change relative to the models used for Checkers [2] and Othello [3]. In the previous works, the spatial processing method was sub-squares sampling. This pattern is related to the rules of checker but not to rules of Othello. That is, for checkers, the key special information required is the status of adjacent diagonal squares as jumps are made diagonally. In other words, any move changes the status of these squares. In contrast to checkers, in Othello, the rules indicate that placing a piece on any given square affects the status of squares in the horizontal, vertical and diagonal directions -- a total of eight directions. So, just as sub-squares is a natural pattern for Checkers, the queen attack positions (in International Chess, henceforth called Q-position patterns) is a natural pattern for Othello.

The sampling procedure of the Q-position pattern is illustrated in Fig. 3 for four different squares. For example, in Fig. 3a the central point of the base Q-position pattern is mapped to the top-left corner of the input board. The occupancy of 22 squares thus forms the input to this node. Similarly, in Fig. 3b, the pattern is mapped to the second square in the top row. Again the occupancy of 22 squares forms the input. This is repeated for all 64 squares of the Othello board. In addition, the full board pattern, which serves the function of a weighted piece counter (WPC), is sampled. As shown in Fig. 1, there are total 65 patterns in this spatial processing layer (hidden layer one) having a total of 1584 inputs. The second and third layers consist of 40 and 10 nodes, respectively. The output of every first layer node is connected to the input of each second layer node, the output of each second layer node is connected to the input of each third layer node. Finally the output of each third layer node forms the input of a single output node which provides a scalar assessment of the value of a the inputted board configuration.. Note, that in contrast to previous work [3], piece differential information is not given to the output node, forcing the neural network to learn to defeat the piece maximum strategy naturally.

In the above fully connected architecture, the total number of weights and biases (denoted $w_i(j)$): [(1649 in hidden layer one)] + [(40 nodes in hidden two) × (65 inputs + 1 bias)] +[(10 nodes in hidden layer 3) × (40 inputs + 1 bias)] + [(1 node in output layer) × (10inputs + 1 bias)] = 4710. For each hidden node and for the output node, the hyperbolic tangent (bounded by +/-1) function is used with a variable bias. The initial population was created by random generation of the weights and biases by sampling from a uniform distribution over [-0.2, 0.2]. We note that the total number of weight and bias is quite small compared to previous architecture (5900 weights and biases) saving calculation time [3].

B. Tournament Selection

In order for evolution to occur, in each generation, a selection mechanism must be employed to determine each neural network's overall fitness relative to the whole population. This allows parents to be selected for the next generation.

In the current simulation, a fair tournament selection model is used [6]. The population at every generation consists of 100 neural networks (strategies): fifty parents and

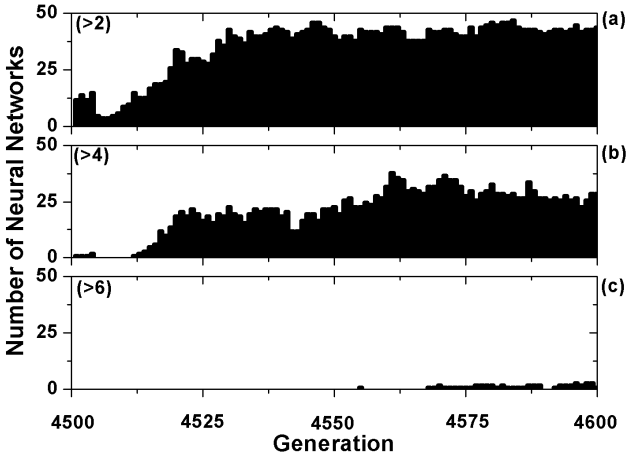

Fig. 6. Evolution in playing ability of the networks within the range 4500 to 5600 generations. Networks were evaluated at every generation by playing against the positional strategy player, with the ply-depth being increased in steps of two until the network lost. Number of networks that can defeat the fixed strategy player playing at ply-depth of (a) two (b) four and (c) six.

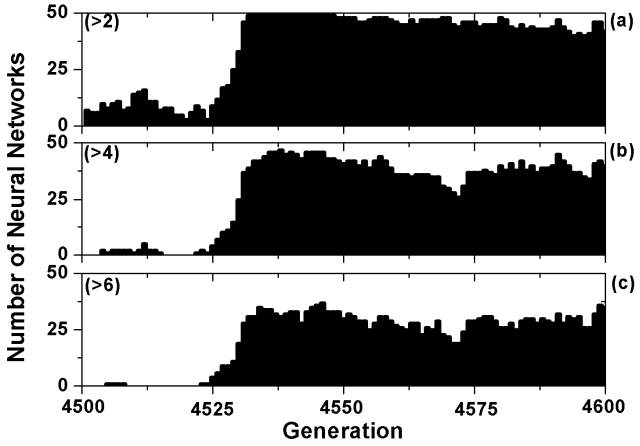

Fig. 7. Evolution in playing ability of the networks within the range 4500 to 5600 generations. Networks were evaluated at every generation by playing against the piece differential strategy player, with the ply-depth being increased in steps of two until the network lost. Number of networks that can defeat the fixed strategy player playing at ply-depth of (a) two (b) four and (c) six.

fifty offspring. An additional ten offspring are generated that are used for competition only. All 100 networks play against a total of 10 games against the test offspring -- 5 as white and 5 as black. As each network plays against the same 10 neural networks, the selection method is completely fair. Strategies are assigned points not on the basis of win/loss/draw but rather by the number of pieces by which each game is won or lost. The top scoring 50 neural networks are chosen as the next generation parents.

We note that this is in contrast to the unfair tournament selection used in the work of Chong [3] where, for every neural network in each generation, not only are opponents randomly chosen, but also the number of games in which a strategy competes is random.

C. Mutation

In the results presented here, the neural networks are not learning in the traditional fashion. Rather the evolution of the neural networks is accomplished through co-evolution. In other words, offspring are generated from parent neural networks by Gaussian mutation. To aid in this each network is assigned its own self-adaptive parameter vector ($s_i(j)$), which is initially set to 0.05 for consistency with the range of initial weights and biases.

Offspring (consisting of weights and bias terms, as well as self-adaptive parameter vector) were generated through self-adaptive Gaussian mutation from each parent, respectively, according to the following equations:

$$s_i'(j) = s_i(j) \times \exp(\tau \times R_j(0,1))$$

$$w_i'(j) = w_i(j) + s_i'(j) \times R_j(0,1)$$

where i=1...50 denotes the neural network being evolved, j=1...4710 spanning across the total number of weights and biases needed for each network being involved. R_j is a standard Gaussian random variable re-sampled for every j and

$$\tau^{-1} = \sqrt{2\sqrt{R_j}} \rightarrow \tau = 0.08535$$

D. External Observers

During simulation, neural networks (after tournament selection) were compared with computer players (external observers) through Othello game competitions to provide an independent monitor of the "fitness" of the population as evolution progressed. The computer player employed the Fail-Soft Alpha-Beta (FAB) Algorithm [5] at variable ply-depths, along with a deterministic evaluation function or strategy. Comparison started with each neural network competing with the computer player playing at ply-depth of zero (the same ply-depth used by neural network). For every network win, the relative level of play of the external observer was raised by increasing its ply-depth by two. It should be emphasized that the results of this competition are not available to the neural networks themselves.

Four distinct fixed strategies were used to monitor the

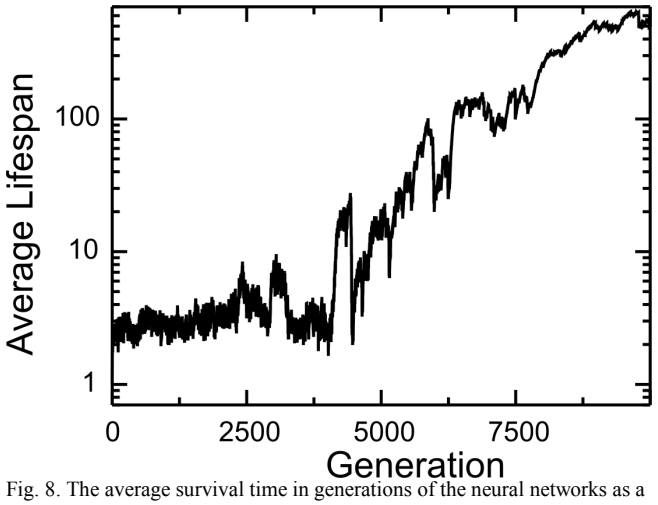

Fig. 8. The average survival time in generations of the neural networks as a function of generation

evolution of the neural networks: a positional strategy (evaluation function based on the positional component of Iago's strategy of Iago [7]), a piece differential player who for a given board position sought to maximize the number of pieces the board, a mobility player who seeks to maximize the number of available moves and the complete expert strategy of Iagno version 22.3 (June 30, 2008) [8] During the fixed strategy comparison, all the neural networks played black (generally considered to be a handicap compared to the white) and used ply-depth of zero.

II. RESULTS

Fig. 4 compares the performance of the neural networks with that of a computer player using the positional strategy to evaluate the quality of a move, while Fig. 5 compares the performance of the neural network with that of a computer player using piece differential as the sole means of evaluating the quality of a move.

Let us consider first the performance of the neural networks relative to the positional player. After only 100 generations, over half of neural networks have learned to defeat the positional strategy playing at a ply-depth of two despite the fact that they are playing black. While a majority of networks quickly learn to defeat the ply-depth=2, it takes another 6000 generations for essentially the whole population to consistently maintain this level of performance. Correlated to this is the fact that only after 4000 generations, do we see performance improving such that the networks that a shift from defeating ply-depth of two to ply-depth of four takes place. Around generation 4500 there sudden improvement with some neural networks learning to defeat the positional player with a ply-depth=6. As can be seen in the figure the lifespan of these exceptional neural networks is quite short with the play of the best networks dropping back quickly. As will be discussed later, we believe that this is due to the networks becoming overly specialized at this point -- developing a specialized strategy to defeat the positional player that is not applicable for other playing strategies. Over the next 3500 generations an increasing number of networks develop the ability to defeat the ply-depth=4 positional player. By generation 7000 essentially all networks have obtained this ability. It is significant to note that this ability is not lost over time. However, despite one attempt at around generation 7000 it is not possible for the networks to defeat the ply-depth=6 player.

Fig. 5 shows that these same neural networks evolved at the end to outperform the piece differential as well. By the 100^{th} generation, over 25 neural networks can defeat ply-depth two piece differential strategy. However, in contrast to their performance against the positional player, they are unable to maintain this level with the level of play appearing to be sporadic., By generation 3000, the neural networks appear to loose all ability to defeat the piece differential player. It is only in the range between generation 4000 to 5000 does the play relative to the piece differential player improve drastically. In contrast to play with the positional player, in which networks first learned to defeat a ply-depth=2 player before learning to defeat the ply-depth=4 player, there is a sudden shift being unable to defeat the ply-depth=2 player to being able to defeat a ply-depth=6 player.

In order to investigate the change in playing competence around generation 4500, we have increased the sampling rate from every 50 to every generation in this region. so as to be able to see how the strategy improvement occurred. Figures 5 and 6 compare the performance of the neural networks with that of a computer player using the positional strategy and piece differential strategies, respectively. Considering first the positional strategy. Against ply-depth of two positional strategy (Fig.6b), the key change of the neural networks' ability happened over 40 generations starting from generation 4525. This improvement is quickly followed, with a delay of about 10 generations, by a considerable portion of the networks learning to defeat the positional player playing at ply-depth=4. The improvement in play relative to the piece-differential player is not, surprisingly, coincident with the improvement seen with respect to play against the positional player but rather delayed by about 15 to 20 generations. Perhaps more amazing is the fact that the jump is from not being able to defeat a ply-depth=2 strategy to being able to defeat a ply-depth=6 strategy and that this ability is gained by about 50% of the population in only about 11 generations. We note that the networks have gained this ability without explicitly being given any piece differential information as was the case in previous simulations [3].

Next we compared the ability of the neural networks with the simple mobility strategy. At the beginning, the networks slowly learn to defeat pure mobility strategies with considerable improvement being seen in the range from generation 4000 to generation 5000. However, after this generation, despite improved play against the positional and piece differential players, the networks seem to have lost all ability to defeat the mobility strategy operating at a ply-depth greater than themselves. In addition, they never regain the ability. Finally, in competition with the general expert strategy Iagno [8], the evolved neural networks exhibited good performance when playing at equal ply-depths.

In Fig. 8 we plot the average survival time of the neural networks as a function of generation. Before the generation 4000, the average survival time is less than 10 generations. After the improvement in play occurs (~generation 5000) the survival time starts to slowly increase reaching over 600 generations by generation 10000 when the simulation was stopped. At the 10000^{th} generation, the average survived times is already reach 600 generation. It would seem that the evolutionary algorithm has converged to a solution that can defeat the positional strategy playing at ply-depth=4, the piece differential strategy playing at ply-depth=6 and the mobility strategy playing at ply-depth=0. In repeating Chong's simulation [3], we found no increase in network survival time over the simulation window.

III. DISCUSSION

In this paper we have compared the performance of our neural networks with four different strategies. One question that arises with regard to these strategies is whether there is one best strategy. Unfortunately this is a difficult question. Playing at a ply-depth=0, a White positional player can defeat the piece differential strategy but loses to the mobility strategy. The White piece differential player can defeat both the positional and mobility strategies, while the White mobility strategy can defeat the positional and piece differential strategies. In addition, the positional player can, on occasion, defeat the combined expert strategy used in Iagno (Iagno incorporates a series of 8 random moves in its opening). However, at a ply-depth of 2, the White positional player loses to the piece differential player. This strongly suggests that there is not one unique strategy that can defeat all comers.

The current simulation has shown that while a given neural network strategy cannot maintain a high level of play against all opposing fixed strategies, networks can be evolved to play reasonable games against all strategies while specializing in defeating some strategies. (Note that neural networks control the weaker black player when competing with the fixed strategies.) The fact that the evolved neural networks perform well against the positional player is not surprising as they have been given spatial information and are allowed to evolve their own weighted piece counter. On the one hand, it is not unsurprising that they had difficulties when playing against a mobility player using a high ply-depth of look-ahead as the networks were not given any mobility information. On the other hand, it is quite interesting that without piece differential information the neural networks can evolve to crush the piece-differential player.

In order to compare with the results for the spatial processing layer of Fogel [2] and Chong [3] it is necessary to speak in terms of the difference in ply-depth used by the strategies. In other words, if we define the ply-depth being played by the neural networks as n, the neural networks in Chong's work learned to play at a level of n+4 against both piece-differential and positional players in a period of 1000 generations. As can be seen in their work, there is considerable noise even until the end of the simulation. In contrast, in the current simulation, using a different spatial layer and neural network having 50% less nodes, and in which comparison was made with a stronger positional player, networks performed at a level of n+4 and n+6 against the positional and piece-differential players respectively. In addition, the more natural sampling pattern coupled with a fair tournament selection reduced noise allowing the network strategies to converge as indicated by the increase network survival time.

REFERENCES

[1] A.E.Leouski, P. E. Utgoff, What a Neural Network Can Learn about Othello, Technical Report, Dept of Computer Science, University of Massachusetts, Amherst, MA, January 20, 1996

[2] D.B. Fogel, *Blondie24: Playing at the Edge of AI*, San Francisco,CA: Morgan Kaufmann, 2002.

[3] S.Y. Chong, M.K. Tan, J. D. White, Observing the Evolution of Neural Networks Learning to Play the Game of Othello, Evolutionary Computation, IEEE Transactions on, 9, 240-251 2005.

[4] Lucas, S. M. Learning to Play Othello with N-Tuple Systems, Australian Journal of Intelligent Information Processing Systems, Special Issue on Game Technology, Vol 9, No. 4 pp 01--20, 2007.

[5] H. Kaindle, "Tree searching algorithms," *Computers, Chess, and Cognition,* Marsland TA and Schaeffer J (eds.), Springer, New York, pp. 133-168, 1990.

[6] A. Sokolov, D. Whitley, Andre' da Motta Salles Barreto, "A note on the variance of rank-based selection strategies for genetic algorithms and genetic programming", Genetic Programming and Evolvable Machines,Vol 8, No.3, pp.221-237, 2007

[7] P. S. Rosenbloom, "A world-championship-level Othello program, Artif. Intell., vol. 19, pp. 279–320, 1982.

[8] Iagno version 22.3 (Released June 30, 2008) Downloaded from http://live.gnome.org/Iagno