Observing the Evolution of Neural Networks Learning to Play the Game of Othello

Siang Y. Chong, *Member, IEEE*, Mei K. Tan, and Jonathon D. White

***Abstract*—A study was conducted to find out how game-playing strategies for Othello (also known as reversi) can be learned without expert knowledge. The approach used the coevolution of a fixed-architecture neural-network-based evaluation function combined with a standard minimax search algorithm. Comparisons between evolving neural networks and computer players that used deterministic strategies allowed evolution to be observed in real-time. Neural networks evolved to outperform the computer players playing at higher ply-depths, despite being handicapped by playing black and using minimax at ply-depth of two. In addition, the playing ability of the population progressed from novice, to intermediate, and then to master's level. Individual neural networks discovered various game-playing strategies, starting with positional and later mobility. These results show that neural networks can be evolved as evaluation functions, despite the general difficulties associated with this approach. Success in this case was due to a simple spatial preprocessing layer in the neural network that captured spatial information, self-adaptation of every weight and bias of the neural network, and a selection method that allowed a diverse population of neural networks to be carried forward from one generation to the next.**

***Index Terms*—Artificial intelligence, coevolution, evolutionary computation, neural networks, Othello.**

I. Introduction

GAMES HAVE always been an important domain for studies into the behavior of artificial intelligence (AI) systems. Board games like Othello, checkers, and chess, provide a simple yet interesting testbed to study both the decision-making and the learning aspects of AI systems. First, these games have specific set of rules that constrain the possible behaviors of the players (i.e., legal moves), thereby simplifying the problem at hand. Second, these games have a definite goal (e.g., to win the game) for the players to achieve. Rewards are given to players that best exhibit certain behaviors (game-playing strategies) under the constraints of finite resources (game pieces) that allow them to achieve the goal. Third, these games have enough subtleties that allow a wide range of complex behaviors represented by the diverse environment of players.

Manuscript received January 23, 2003; revised June 10, 2003 and June 25, 2004.

S. Y. Chong is with the School of Computer Science, University of Birmingham, Edgbaston, Birmingham B15 2TT, U.K. (e-mail: s.y.chong@cs.bham.ac.uk).

M. K. Tan is with the Faculty of Engineering and Technology, Multimedia University, Melaka 75450, Malaysia (e-mail: iem_nat@yahoo.com).

J. D. White is with the Department of Electrical Engineering, Yuan Ze University, Taoyuan 320, Taiwan (e-mail: whitejd@xiaotu.com).

Digital Object Identifier 10.1109/TEVC.2005.843750

Many of the previous approaches to develop strong AI systems to play these games focused on incorporating expert knowledge, whether from domain experts, studies, or experimentations. Despite enjoying measured success, most of them required substantial human knowledge that needed to be preprogrammed into the system. Unlike most human players that learned to play games from experiences, these AI systems simply lacked the capability to learn to play the games on their own.

However, some researchers have developed systems that can learn to play games, using design cues from natural evolution [1]–[6]. These evolutionary systems are a result of more widespread research efforts on evolutionary computation [7]–[10], as well as its application in neural networks' training [11]–[15]. Evolutionary computation and neural networks have been used either individually or as hybrids to solve many other interesting real-world problems [16]–[18]. In particular for applications in games, these evolutionary systems have shown great success. In his recent book, "*Blondie24: Playing at the Edge of AI*" [2], Fogel showed that neural networks, trained using evolutionary computation techniques, can be evolved to play checkers at a high level without requiring any preprogrammed expert knowledge. When pitted against human players on the Internet, the best-evolved neural network obtained an expert ranking (rating of 2045).

In this paper, we will focus on the game of Othello. Like checkers, Othello is an interesting board game for studies. Othello is a deterministic, perfect information, zero-sum game of two players. It has a simple set of rules, an average of 60 moves to complete, and a small average branching factor of seven [19]. However, despite its simplicity for players to learn, it is a challenging game to master. One reason is that unlike games like chess, every legal move in Othello adds an extra piece to the board that cannot be removed (only flipped to another color). As the game progresses, possible moves are gradually limited. Another reason is that every legal move involves flipping pieces to another color. This makes it difficult to keep track of the piece topology because it can change drastically from one move to another.

As for games in general, most programs developed for strong Othello play depend on expert-knowledge-based evaluation functions and advanced search algorithms that can search to deep plies in a relatively short time. A typical example is Rosenbloom's Iago, the first master-level program [20]. No learning mechanism was incorporated. Later, Lee and Mahajan introduced Bayesian learning (by combining features in the evaluation function) into Bill, although advanced searching and timing techniques were still required [21]. Finally, Buro

continued with the development of a stronger program by improving on and using more sophisticated machine learning techniques [22]. The resulting program called Logistello became the first Othello program to defeat a human world champion [23], [24].

An interesting alternative is to employ a hybrid system of evolutionary computation and neural networks to enable the system to learn to play Othello without expert knowledge. Within this general framework, three different approaches are possible: 1) apply evolutionary algorithms (EAs) to the search algorithm to allow for more effective pruning, while keeping the evaluation function fixed; 2) apply EAs to the evaluation function, while keeping the search algorithm fixed; or 3) apply EAs to both the search algorithm and the evaluation function.

Moriarty and Miikkulainen chose the first approach, employing simple genetic algorithms (GAs) to evolve focus networks for more effective pruning and searching through the game tree by directing minimax searches away from poor information [5]. In comparison to a full-width, fixed-depth alpha-beta, the selective search by this focus network resulted in better play at reduced search times. Its success was due to the ability of evolved focus networks to produce a ranking for the moves. Moriarty and Miikkulainen also evolved neural networks (at the level of architecture and connectivity) that could make moves directly and that exhibited strong Othello game-play [6].

In this paper, we take the second approach—keeping the search algorithm fixed and evolving the evaluation function. We show that through coevolution, neural networks can be evolved to learn to play Othello. In Section II, we describe the model and the implementation of the simulation. In Section III, computer players implementing deterministic strategies were used as external monitors of the evolutionary process. We show that the population of neural networks coevolved in level of play from novice, to intermediate, and finally, to master level. In Section IV, individual games were analyzed. Such analysis shows that the evolved neural networks (ENNs) discovered interesting strategies, starting with positional strategy and, in the later generations, the more sophisticated mobility strategy. In addition, since the approach used here is considered generally to be difficult as it requires comparisons of absolute values produced from different neural networks, as well as activations from the same neural network (we note previous attempts in Othello were not encouraging [5]), we introduce three important design criteria that led to success in this case. Finally, Section V concludes this paper with some recommendations for future work.

II. Game Implementation

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 [Fig. 1 (top-left)], 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 [Fig. 1 (top-right)] 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 [Fig. 1 (bottom-left)]. The move is completed when the surrounded pieces are removed and replaced with pieces of the player's own color [Fig. 1 (bottom-right)]. 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, each player is awarded a draw.

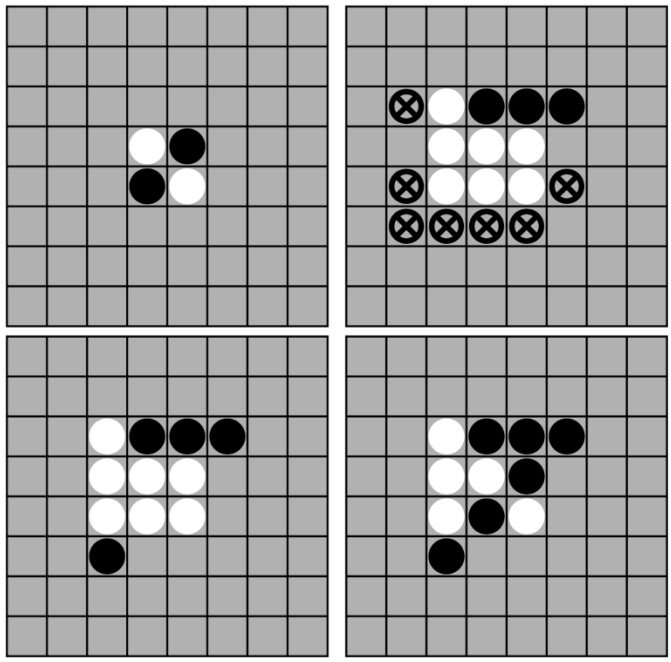

Fig. 1. Illustration of basic allowed Othello moves. Players alternatively place black and white pieces on the board. (Top-left) The initial board position. (Top-right) Legal black moves at a point later in the game are shown with crossed circles. Once the player (black) has selected a valid move (bottom-left), the surrounded enemy (white) pieces are replaced with the player's own pieces (bottom-right).

With respect to the implementation of the simulation, four modules [1) search algorithm; 2) neural-network-based evaluation function; 3) mutation module that is applied to the neural networks; and 4) tournament-based selection for the next generation] are interfaced together to simulate evolution of a population of neural networks (game-playing strategies). In addition, a fifth module is used to monitor the evolution process without participating. The simulation itself was written in ANSI-C and compiled and run on a 1 GHz Intel processor-based PC. Each module is discussed below.

A. Search Algorithm

As the emphasis of this study is not on optimizing the search algorithm, this module employs the minimax algorithm described in [25]. The basic principle of the algorithm is that the best possible move for every turn is the one that minimizes the maximum damage the opponent can inflict. It assumes that the heuristics used to determine the value of the moves when searching through the game tree are accurate. Here, this well-known approach to solving zero-sum games (i.e., chess, checkers, and Othello) is used to search for the best possible

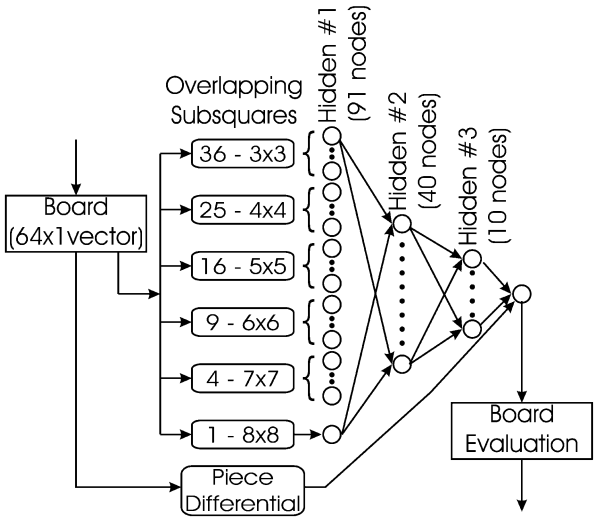

Fig. 2. Model of the neural-network-based evaluation function. The model is based on a feed-forward MLP used in [1]. Given any board pattern as a vector of 64 inputs, the first hidden layer (spatial preprocessing layer) assigned a node to every possible square subset of the board. The outputs of these nodes were then passed to two additional hidden layers, consisting of 40 and 10 nodes, respectively. The final output node, which included the piece differential as an additional input, was scaled between $[-1, 1]$. This output represents the evaluation of the board position by the neural network.

move based on game heuristics (evaluations) provided by the neural networks. Thus, the overall strength of play of the neural networks will be based largely on the reliability of game heuristics provided by the neural networks (input–output response) when searching through the tree. During game simulation, the ply-depth used for searching was restricted to two for the neural networks. A variable ply-depth mechanism is not implemented as the focus of this study is on evolving the evaluation function and not making improvements to the basic minimax algorithm.

B. Neural-Network-Based Evaluation Function

The neural network provides strategies required to play Othello effectively through its evaluation of the board positions presented to it during minimax search. It depends on both piece differential information and the use of spatial information gained through pattern recognition for decision-making, as well as the ability to generalize input–output mappings to effective Othello strategies. It should be noted that unlike other approaches, no Othello expert knowledge such as lookup tables for opening, middle and end sequences, or expert heuristics of board position values are used in the simulation. Neural networks are initialized randomly and have to learn to evaluate board positions through coevolution following the methods of [1] and [4].

Fig. 2 illustrates the neural network model that is used for the evaluation function. It is based on the multilayer perceptron (MLP) model implemented in Chellapilla and Fogel's experiments on evolving neural networks to learn to play checkers [1]. The neural network consists of an input layer, three hidden layers, and an output layer.

The input layer is designed to represent the Othello board. It is a 64×1 vector (array) consisting of 64 components to represent the 64 Othello board positions. The components can take black pieces, white pieces, or empty squares that are represented in a computable manner $\{-1, 0, +1\}$, where 0 represents an empty square, while $+1$ and -1 represents a black piece and white piece, respectively.

The first hidden layer acts as a spatial preprocessing layer to allow the neural network to process, analyze and most important of all, to generate features of the possible spatial information contained in the Othello board. As shown in Fig. 2, this layer consists of a total of 91 nodes of seven different types—each responsible for sampling a different board subsection. For example, 36 nodes sample all possible 3×3 board subsections, while another 25 nodes sample all possible 4×4 subsections of the 8×8 board. While other methods which take into account the symmetry of the board are possible [19], this method ensures that no spatial features characteristic of the Othello board are input *a priori*, while still allowing the neural network to capture spatial information.

The second and third layers consist of 40 and 10 nodes, respectively. Each second layer node is connected to the output of every first layer node, while each third layer node is connected to the output of every second layer node. A single output node, in addition to being connected to the output of every third layer node, also receives piece differential information. The output of this node (a real number between -1 and $+1$) is the evaluation of the current board position. This output is returned to the search algorithm module. A positive value indicates a board position that the neural network considers to be favorable to it, while a negative value indicates a board position adverse to the neural network. As for the design of the nodes, at each node a hyperbolic tangent (bounded by ± 1) function is applied to the sum of the weighted inputs and a bias to provide nonlinearity.

Three hidden layers were chosen as a similar architecture was successfully applied by Chellapilla and Fogel to the problem of learning to play checkers [1]. For Othello, a game of moderate complexity (harder than checkers [19], yet simpler than chess), it is likely that the number of hidden layers are more than that required by the problem. However, any hand optimization to the number of hidden layers and nodes of the fixed-architecture neural networks would represent a *defacto* use of *a priori* knowledge, and as such would defeat the purpose of this simulation—the avoidance of using *a priori* knowledge. In addition, such an optimization would offer no significant advantage since the main cost in computing time is the minimax algorithm, not the neural-network-based evaluation function. We note that while there are other sophisticated approaches that can evolve both neural network weights and structure at the same time [6], [14], [15], the advantage of the approach used here in evolving the weights of a fixed-architecture neural network is its conceptual and implementation simplicity.

C. Mutation

Coevolution is used to evolve the neural networks. The advantage of this approach is that it precludes the need to design a strong computer player to act as opponent and fitness evaluator for the neural networks. Designing such an opponent still requires expert knowledge of the game. In coevolution, neural networks compete with their peers (other neural networks of the same generation) for survival. The aim is that by having neural networks compete with their peers that are evolving at the same

time and hopefully improving, an escalating arms race will result and neural networks will discover Othello game-playing strategies. A beneficial effect of a careful use of coevolution is that it can produce more general game-playing strategies compared to traditional evolutionary approach that uses deterministic opponents. Coevolution had been used in many game-related problems such as checkers [1], Backgammon [26], and Iterated Prisoner's Dilemma [27], [28].

As in other real-valued parameters optimization problems, the coevolutionary approach used requires an efficient mutation scheme that generates appropriate changes to move object vectors (in this case, the neural networks) to optimal regions by a right amount and at the right time [29]. For example, Ankit and Fogel [30] showed that when Gaussian mutation was used to optimize a simple neural network to solve the XOR problem, the optimal mutation step size is different at different time (generation) during the evolutionary process. Here, instead of experimenting with heuristic schedules to vary mutation step sizes throughout the evolutionary process (which can be difficult), self-adaptation is used to allow the evolutionary process itself to search and vary the step sizes accordingly [7], [10], [31]. Self-adaptation has been used in most EAs for optimizing real-valued representations such as in evolution strategies (ES) and evolutionary programming (EP) [8], [9], [32]. Self-adaptive mutation operators have been investigated in a number of benchmark problems [33]–[35]. Furthermore, a self-adaptive mutation operator has been successfully used to evolve neural networks to solve real-world problems such as in breast cancer detection [18].

In this paper, the primary variation operator used is a self-adaptive Gaussian mutation, extended to include individual control of mutation step size for each weight or bias of the neural network [1]. This increase in the degree of freedom in mutating the neural networks is important, given the complexity of the problem and the architecture of neural network being used. The procedure for generating offspring is described as follows. The population at every generation consists of 20 neural networks (strategies): ten parents and ten offspring. Each neural network has its own weights and biases (denoted $[w_i(j)]$), as well as a self-adaptive parameter vector (denoted $[\sigma_i(j)]$) that controls the step size of the mutation of the weights and biases when offspring are generated. In each generation, the self-adaptive Gaussian mutation is used to generate offspring.

The initial population was created by randomly generating (by sampling from a uniform distribution over $[-0.2, 0.2]$) the weights and biases $[w_i(j)]$ of ten parent neural networks (PNNs). Each component of the self-adaptive parameter vector $[\sigma_i(j)]$ of the PNNs was set to 0.05 for consistency with the range of initial weights and biases.

Ten offspring (consisting of weights and bias terms $[w'_i(j)]$, as well as self-adaptive parameter vector $[\sigma'_i(j)]$) were generated through self-adaptive Gaussian mutation from each of the ten parents, respectively, according to the following equations:

$$\sigma'_i(j) = \sigma_i(j) * \exp(\tau * N_j(0,1))$$
$$w'_i(j) = w_i(j) + \sigma'_i(j) * N_j(0,1)$$

where $i = 1, \ldots, 10$ denotes the neural network being evolved, $j = 1 \ldots N_w$ where N_w is the total number of weights and biases needed for each neural network, $\tau = (2(N_w)^{0.5})^{-0.5} = 0.08068$ and $N_j(0,1)$ is a standard Gaussian random variable resampled for every j. In the above architecture: $N_w =$(total number of nodes and biases in layer 1)+(40 nodes in layer 2)×(91 inputs+1 bias)+(10 nodes in layer 3)×(40 inputs+1 bias)+(1 output node)×(10 inputs+1 bias)= 5900.

D. Tournament Selection

For each generation, a selection mechanism is employed to determine a neural network's overall fitness relative to the whole population in order to select the parents for the next generation. While it is possible to use a fitness function, such an approach would change the simulation into a strictly traditional optimization exercise where knowledge is important. In this experiment, tournament selection (EP selection mechanism [36]) is used in the coevolutionary setting. Tournament selection has been used successfully in a number of experiments that involve coevolving neural networks to play games [1]–[4]. In the present work, each neural network competes (as black) against five randomly chosen peers. For each game, the winning and losing neural networks received five and zero points, respectively. In the case of a draw, both neural networks received one point. This point setting was chosen to encourage neural networks to play for wins rather than draws. Two neural networks were not prohibited from competing twice with each other if each game was played as a different color. Games were ended when ten move skips occurred (i.e., generally when neither player is able to move). A key feature of the tournament selection used here is that the number of times a neural network was selected to play white were not fixed. This design adds further to the nondeterministic nature of the selection method in order to further increase population diversity in the next generation by occasionally allowing weaker neural networks to survive.

At the end of the tournament, the ten neural networks with the most points were selected as the parents of the following generation. Control was then passed to the mutation module. Together, iterating the process of modules 3 (mutation) and 4 (tournament selection) allows for coevolution of the neural-network-based evaluation functions.

E. External Observer

During simulation, the parents for the next generation (after 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 same search algorithm but rather than using a neural network to provide the board evaluation, a deterministic evaluation function (such as one based on piece differentials or positional information) was employed. Comparison started with each PNN competing with the computer player playing at ply-depth of two (the same ply-depth used by neural network). For every PNN win, the relative level of play of the computer player was raised by increasing its ply-depth by two. The PNN competed always as black. It

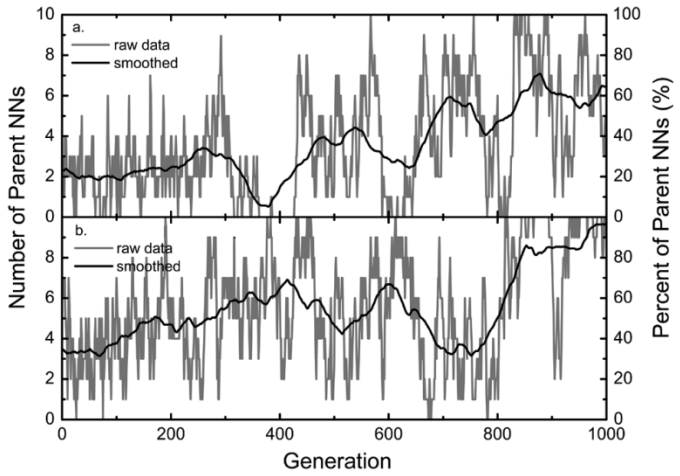

Fig. 3. Number and percent of PNNs playing at ply-depth of two capable of defeating computer players using fixed evaluation function and playing at ply-depths greater than the PNNs as a function of generation. The raw data is indicated by gray lines. The black line is the result of adjacent averaging over 100 generations and shows improvement as evolution continues although there are setbacks. In (a), the opponent is using a piece-differential-based evaluation function, while in (b), the opponent is a positional player (evaluation function is based on the positional strategy of Iago [20]).

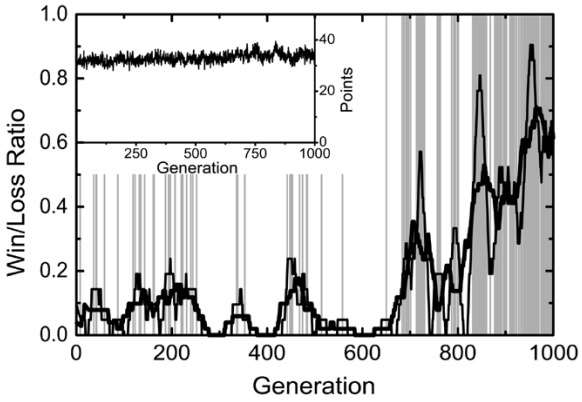

Fig. 4. Comparison between the top-ranked PNN of each generation with the piece differential player playing at ply-depth of six. Vertical gray lines indicate wins. The black lines represent the win/loss ratio of the top-seated PNN averaged over 20 (thin line) and 50 (thick line) generations. The inset illustrates the average tournament points received by the population of PNNs.

should be emphasized that the results of this competition are not available to the neural networks themselves.

III. Results

The simulation was run for 1000 generations and the results were summarized in Figs. 3–5. Fig. 3 shows the number of PNNs that outperformed computer players using fixed evaluation functions as a function of generation. The PNN played black and used a ply-depth of two, while the computer players were allowed to play at ply-depth of four. In Fig. 3(a), the computer player used a piece-differential-based evaluation function. The raw data (gray lines) indicates that even in early generations, there were PNNs that defeated the piece differential player. Looking at the entire evolutionary process, improvements were noted across 1000 generations although substantial fluctuations could be seen. In later generations, a consistently greater percentage of PNNs outperformed the

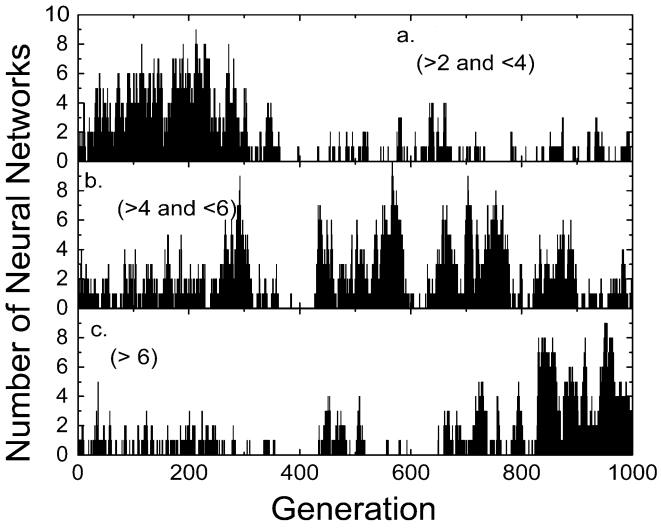

Fig. 5. Evolution in playing ability of the PNNs with generation. The PNNs at each generation were evaluated by playing against the piece differential player, with the ply-depth being increased in steps of two until the PNN lost. (a) Number of PNNs that can defeat the piece differential player playing at ply-depth of two but lost when it is playing at ply-depth of four. (b) Number of PNNs that can defeat the piece differential player playing at ply-depth of two and four but lost when it is playing at ply-depth of six. (c) Number of PNNs that can defeat the piece differential player playing at ply-depth of two, four, and 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.

piece differential player. At the end of the simulation, about nine out of ten PNNs achieved this particular level of play. This evolution can be seen more clearly by adjacent averaging the numbers over 100 generations (black lines), which indicates, despite fluctuations, gradual improvements in the PNNs' performance, starting from approximately 20% win-rate to a 70% win-rate over the generations. A least-squares fit of a second-order polynomial (third-, and fourth-order coefficients are essentially zero) indicates that the rate of improvement of the play of the PNNs increased as evolution progressed.

Fig. 3(b) provides a similar comparison between PNNs and a computer player that used a position-based evaluation function (positional strategy of Iago [20]). Looking at the evolutionary process as a whole, the results in Fig. 3(b) are somewhat similar to Fig. 3(a), with improvements of the PNNs' competitiveness against the positional player as evolution continued. As in Fig. 3(a), fluctuations from generation to generation, and "performance dips" can be observed. However, the exact locations of the dips differ [compare the black lines in Fig. 3(a) and (b)]. For example, around the 600th generation, very few PNNs employed strategies that can defeat the piece differential player but the majority of PNNs defeated the positional player, indicating that performance dips are not the result of a complete loss of playability among the PNNs. The significance of these dips will be discussed later in the text.

Fig. 4 (inset) shows the change in the average tournament points of PNNs over the generations (maximum point is 50). While slight fluctuations exist between generations, there is little increase in the average point of the ensemble of PNNs over time. The average remains at seven wins with very few draws. Individually, each PNN also achieved between six to eight wins, indicating that neural networks are coevolving without

any "superstars" appearing. The vertical lines (gray) in Fig. 4 (main) are the results of comparisons between the PNN ranked first based on the tournament selection in each generation with the piece differential player playing at ply-depth of six. Vertical lines occurred whenever the top PNN defeated the piece differential player. For a given generation, the win/loss ratio was either one (win) or zero (loss). While isolated victories occurred earlier in the young population, it was only in the later generations that top PNNs could defeat consistently the piece differential player. This can be seen more clearly with adjacent averaging of top PNN's performance over 20 (thin black line) and 50 generations (heavy black line). For the young population, the win/loss ratio remained below 0.2 dropping to 0.0 at about the 600th generation. However, from the 600th generation onwards, particularly from the 800th to 1000th generations, most of the top PNN during this period outperformed the piece differential player playing at a ply-depth of six. This indicates that coevolution had found higher level of play, with top PNNs not only outperforming the piece differential player, but also their peers in their respective generation that were similarly competitive with the same piece differential player.

Fig. 5 illustrates the evolution of the neural network population in relation to the level of play expected by a human player. Fig. 5(a) plots the number of PNNs playing at novice level (able to defeat the piece differential player when it is playing at ply-depth of two but lost when it is playing at ply-depth of four). Fig. 5(b) plots the number of PNNs that are playing at an intermediate level (able to defeat the piece differential player when it is playing at ply-depth of four but lost when it is playing at ply-depth of six). Fig. 5(c) plots the number of PNNs that are playing at a master level (able to defeat the piece differential player when it is playing at ply-depth of six). The majority of the PNNs that were playing at the novice level occurred in the earlier generations of the evolution [Fig. 5(a)]. The number of PNNs playing at the novice level decreased sharply around the 400th generation as most of the PNNs improved their level of play. From the 400th generation to 800th generation, the majority of the PNNs played at intermediate level [Fig. 5(b)]. However, the shift from novice to intermediate level was not particularly specific, with indications of intermediate level of play surfacing as early as the 250th generation for a short period. Sustained intermediate level of play among the PNNs started around the 425th generation, with occasional dips at intervals between the 600th and 625th, as well as at the 800th to 820th generations. The number of PNNs playing at an intermediate level slowly diminished from the 800th generation onwards being replaced with PNNs playing at master level of play. PNNs started to play at a master level around 700th generation [Fig. 5(c)] with the shift from intermediate to master level of play occurring around the 800th generation. (We were unable to run the computer player at ply-depths higher than six, as the computation time taken by the computer player was prohibitive on the available personal computers).

IV. Discussion

A. Evolution of the Neural Networks

As mentioned earlier, the overall fitness of the population of neural networks was monitored with the help of an external computer player through game competitions. For effective monitoring, it is important that the computer player's level of play is neither substantially stronger nor substantially weaker than the evolving neural networks. A computer player that either defeats all or loses to all the neural networks is not a good monitor. Furthermore, the computer player's level of play should be adaptable in a controlled manner. This was accomplished by allowing the computer player to use the same minimax search algorithm and to vary the ply-depth. Given the advantage that the neural networks had from their added computational power, the neural networks were handicapped. First, neural networks were limited to using a ply-depth of two. Second, the neural networks played black as the computer players are better playing white (when two computer players using the same evaluation functions competed with each other, the black player needed to play at ply-depth of six to defeat the white player playing at ply-depth of two). Such an appropriate choice of the computer player allowed us to observe the changing level of play of the population of neural networks.

Consider first the performance of the neural networks in terms of overcoming the handicap of playing black and using a ply-depth of two for the minimax. Results in Fig. 3(a) show increasing numbers of the PNNs outperformed the piece differential player playing at ply-depth of four as evolution continued. At the end of the evolution, the majority of the PNNs outperformed the computer player. From these two observations, there is a strong indication that the neural networks had evolved to overcome the handicap of playing black and using a shallow minimax search depth of two. This is an interesting result, considering that no information that could help distinguish the performance among the strategies (represented by neural networks) were fed back to the evolutionary system. The only information available was the points obtained from the neural network's competitions, and this information did not indicate the specific games that the neural network won, lost, or drew.

Fig. 5 shows that the level of play of the ensemble of PNNs improved gradually as evolution continued. Before the 400th generation, the majority of PNNs played at novice level. As evolution continued, the majority of PNNs improved in their level of play to intermediate, and finally to master level. From about the 800th generation onwards, the majority of the PNNs achieved master level. Fig. 6 shows that these same PNNs evolved at the end to outperform the positional player as well. In addition, the top PNNs at each respective generation also achieved a similar level of play (Fig. 4). This observation indicates that the top performing neural networks had evolved to a high level of play, given that they can defeat both piece differential and positional players playing at ply-depth of six, as well as most of their peers that were similarly competitive with the computer players.

Having observed from the game statistics the gradual evolution of the neural networks to a high level of play, we now turn to the analysis of individual games between the neural networks and the computer player to look for changes in the strategies employed by the networks. Starting early on around 250th generations, analysis of individual games showed that the PNNs had ascertained the importance of the corner points. Although individual PNNs started the game differently and appeared to be playing different strategies, during the middle game, most

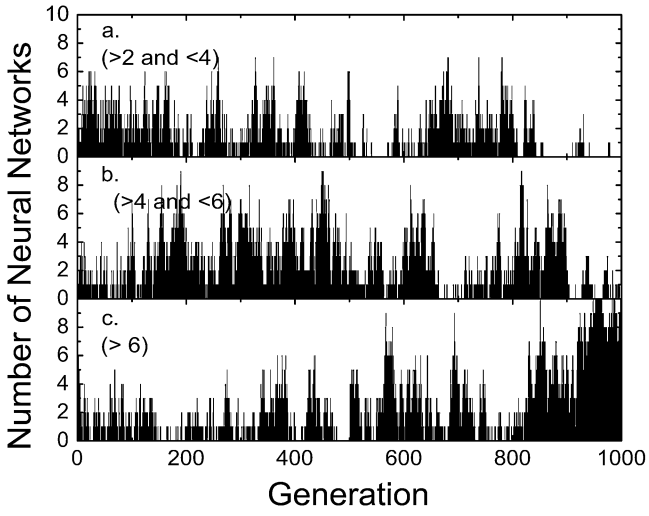

Fig. 6. Evolution in playing ability of the PNNs with generation. In this figure, the external observer is a positional player. (a) Number of PNNs that can defeat the positional player playing at ply-depth of two but lost when it is playing at ply-depth of four. (b) Number of PNNs that can defeat the positional player playing at ply-depth of two and four but lost when it is playing at ply-depth of six. (c) Number of PNNs that can defeat the positional player playing at ply-depth of two, four, and six.

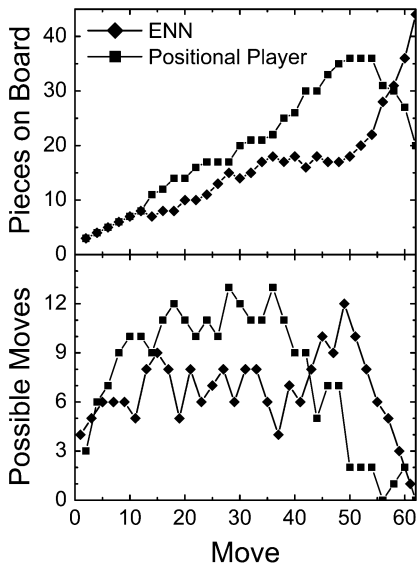

Fig. 7. (Top) Total number of pieces on the board and (bottom) total number of possible moves for an PNN (generation 643) playing against a positional player as an Othello game progresses. The PNN is playing at a ply-depth of two and the positional player is playing at a ply-depth of four.

PNNs captured more corner points than the computer player. After that, PNNs took advantage of the corner points to win the game. It appears that neural networks, through the use of the spatial preprocessing layer, had discovered and incorporated interesting positional strategies that placed importance to, and then captured the strategic corner points.

As the neural networks continued to evolve, their style of play shifted from positional and started to show hints of employing mobility strategy [37]. Fig. 7 shows the example of one game between a PNN at generation 643 that used a mobility strategy against the positional player playing at ply-depth of four. From

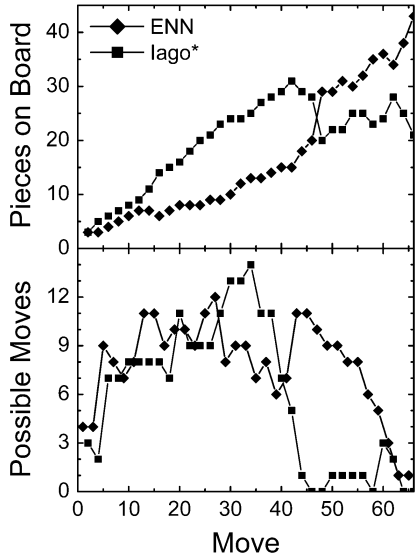

Fig. 8. (Top) Total number of pieces on the board and (bottom) total number of possible moves for an PNN (generation 892) playing against a computer player as an Othello game progresses. The computer player is playing at a ply-depth of two and employing both the positional and mobility strategies of Iago. The PNN is playing at a ply-depth of four.

the start to the middle of the game, the neural network maintained a low piece count allowing the positional player to accrue more pieces. However, as the game progressed from the middle to the end, the neural network maintained much greater mobility (i.e., at move 50, the neural network had 12 possible choices, while the positional player had only two). Starting from move 50, the better mobility of the PNN (the positional player only had around one to two choices from this point onwards) allowed it to move-by-move reduce the piece differential and finally win by a comfortable margin of 44:20.

Toward the end of the evolution, the majority of the PNNs did well against both positional and piece differential players, indicating more feasible and general use of strategies that included mobility play. As a further comparison, PNNs in the later generations were pitted against a player incorporating both the positional strategy and mobility strategy of Iago. PNNs playing at ply-depth of two were competitive with the Iago-like player, slightly losing out at around four pieces. However, when PNNs' ply-depth was increased to four, they outperformed the Iago-like player. For example, in one game, the PNN exhibited a feasible mobility strategy against the Iago-like player (Fig. 8). However, as reported in [6], it will be difficult for PNNs to compete against players with strong positional and mobility strategies such as Bill [21] or Logistello [22] even while playing at higher ply-depths.

B. Noise in, and Reliability of, the Evolutionary Process

As seen in Fig. 3 (data), there are strong fluctuations in the performance of the PNNs between generations, as well as some longer performance dips at the 400th, 600th, and 800th generations when comparisons were made with the piece differential player. This strong noise is due mainly to using a small population (20 neural networks) and was confirmed by increasing the population size by 50% and rerunning the simulation. Fig. 9 illustrates the result, showing a smoother evolutionary process

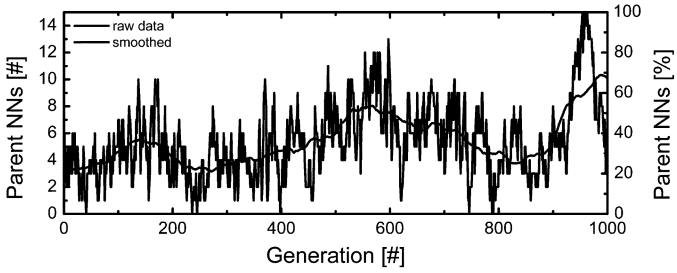

Fig. 9. Number and percent of PNNs playing at ply-depth of two capable of defeating piece differential player playing at ply-depths greater than the PNNs as a function of generation for a population size of 30. The black line is the result of adjacent averaging over 100 generations.

with less pronounced fluctuations. Performance dips, while still present, do not occur for substantial periods of time (generation). This effect of population size on noise can be understood by noting that each neural network represents a search point in the Othello strategy search space. A small population has a tendency to become homogeneous because it lacks the information capacity to accurately sample the search space [38]. Indeed, the original experiment (Fig. 3) showed that it was difficult to evolve neural networks to be effective against both piece differential and positional players at the same rate. As seen in the 400th, 600th, and 800th generations, the majority of the neural networks were only effective against a particular deterministic strategy. Although the evolution of neural networks against both players showed an increasing trend in the long run, it was only toward the end that the neural networks evolved to be simultaneously competitive against both players. Increasing the population size can help reduce the noise and subsequently increase the reliability of the evolutionary process, since increasing the population size (i.e., search points) makes it easier for the evolutionary process to balance exploring other regions for potential strategies and maintaining current feasible strategies. However, consideration also needs to be given to the additional computation time per generation incurred from using a larger population size. In this work, we have shown that, despite using a small population size, good performance could still be achieved with very low computation costs.

C. On A Priori Knowledge

In this paper, we have argued that no use has been made of any preprogrammed expert knowledge. In response to the argument that the knowledge of the piece differential provided to the neural network is a form of expert knowledge, it should be noted that any beginner knows that the player with the most pieces at the end of the game, wins. Furthermore, piece difference is an obvious and important factor for consideration at every move that can be computed easily by even novice human players. In response to the argument that the inclusion of the spatial preprocessing layer in the neural network is a form of expert knowledge, it should be noted that any human player could see and discern, with ease, the spatial arrangement of pieces on the Othello board. The spatial preprocessing layer should not be considered as a preprogrammed mechanism to give the neural network advantage as it has only provided a basic ability for the neural network to sample the Othello board for spatial information. It is the spatial feature extraction and the corresponding information processing that should be considered as *a priori* knowledge, and in this experiment, this knowledge was evolved, not preprogrammed. Finally, in response to the argument that the use of a minimax search algorithm is a form of expert knowledge, it may be legitimate if a deep search is used. In the current work, however, the neural network is limited to using only a ply-depth of two. In this case, the use of minimax offers no big advantage to the neural network's performance. It only provides the neural network with a mechanism to compare board situations from certain move choices. The move selected as a result of the minimax depends on the accuracy of the neural network in evaluating the board situations. In turn, the accuracy of the neural network's evaluations depends on its representation of strategies, which is evolved, not preprogrammed.

D. Design Criteria Important for Success

In this paper, we have shown that through coevolution, fixed-architecture neural networks used as an evaluation function can be evolved to learn to play Othello. Such an approach is considered generally to be difficult because it requires comparisons of absolute values produced from different neural networks as well as activations from the same neural network. Indeed, previous attempts in Othello were not encouraging [5]. This raises the question of why this approach was used here successfully. In the following paragraphs, we discuss three important factors that led to the successful coevolution.

First, neural networks were provided with the ability to sample spatial information from the board. Earlier, it was argued that one of the requirements for the neural network in learning to play Othello is the ability to identify spatial characteristics of the board. The spatial preprocessing layer provided such a capability to the neural network. Two experiments were conducted to look at the utility of the spatial preprocessing layer, especially in allowing the neural networks to achieve master level of play (Fig. 5). In the first, the spatial preprocessing layer was removed to give a flat neural network: two hidden layers at 40 and 10 nodes, respectively. In the second, the spatial preprocessing layer was the only hidden layer. Without the spatial preprocessing layer, it was very difficult to evolve neural networks to play Othello (Fig. 10). With only the spatial preprocessing layer, the evolutionary process did meet with some success (Fig. 11). Clearly, the spatial preprocessing layer is crucial in giving the neural networks the capacity to evolve to higher levels of play.

Second, the self-adaptation process was extended to every weight and bias of the neural network. As seen in Fig. 12, when the vector $[\sigma_i(j)]$ was replaced with a scalar $[\sigma_i]$ (a single component self-adaptive parameter used to control mutation step sizes for all weights and biases) for each neural network, evolution did not occur. Similar results were seen when the vector $[\sigma_i(j)]$ was replaced with a fixed scalar $[\sigma_i]$ of arbitrary values (essentially changing the operator to a simple Gaussian mutation). These results suggest that although for simple optimization problems, simple Gaussian mutation can be used [30], for more complicated problems, such as coevolving complicated neural networks to learn to play Othello, it becomes crucial to extend the degree of freedom of the self-adaptation process as

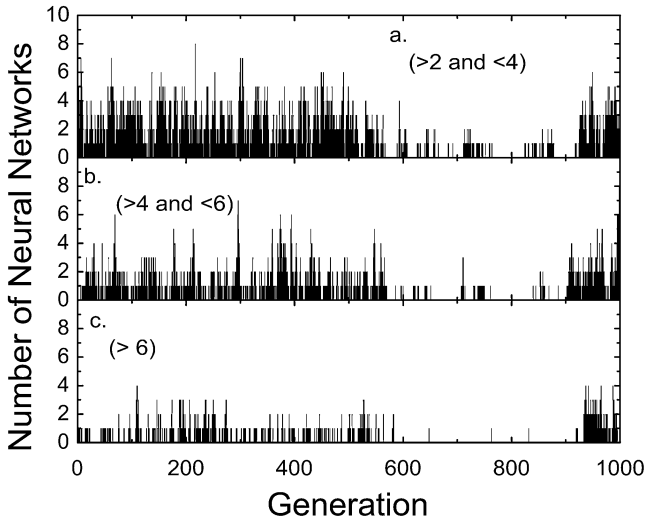

Fig. 10. Evolution in playing ability of the PNNs with generation with the spatial preprocessing layer removed from the neural network. (a) Number of PNNs that can defeat the piece differential player playing at ply-depth of two but lost when it is playing at ply-depth of four. (b) Number of PNNs that can defeat the piece differential player playing at ply-depth of two and four but lost when it is playing at ply-depth of six. (c) Number of PNNs that can defeat the piece differential player playing at ply-depth of two, four, and six.

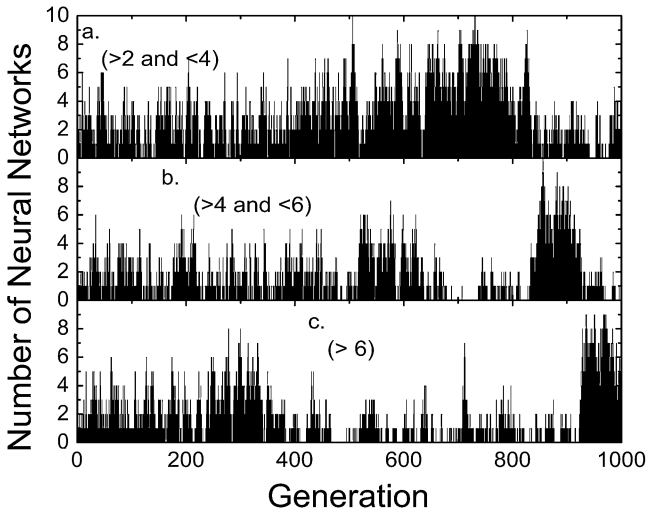

Fig. 11. Evolution in playing ability of the PNNs with generation with the spatial preprocessing layer as the only hidden layer. (a) Number of PNNs that can defeat the piece differential player playing at ply-depth of two but lost when it is playing at ply-depth of four. (b) Number of PNNs that can defeat the piece differential player playing at ply-depth of two and four but lost when it is playing at ply-depth of six. (c) Number of PNNs that can defeat the piece differential player playing at ply-depth of two, four, and six.

the optimal mutation step sizes may vary between individual weights and biases. (We note that [33] showed that Cauchy mutation or mixing of Gaussian and Cauchy mutation further improves EAs that use self-adaptation, while Fogel [39] showed that having multiple self-adaptive parameter vectors can improve the performance of the self-adaptation process).

Third, in order to sustain the arms race to search for more innovative strategies in the coevolutionary process, it is important to maintain diversity in the population. This is a crucial consideration as coevolution of a small and finite population can result in overspecialization to specific strategies [40], [41].

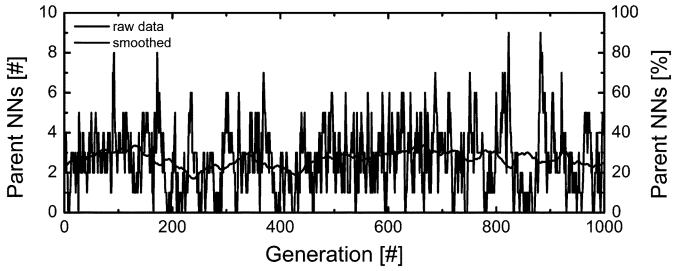

Fig. 12. Number and percent of PNNs playing at ply-depth of two capable of defeating piece differential player playing at ply-depths greater than the PNNs as a function of generation when the self-adaptive vector $[\sigma_i(j)]$ was replaced with a self-adaptive scalar $[\sigma_i]$. The black line is the result of adjacent averaging over 100 generations.

To encourage a longer arms race, a number of methods have been developed [40]–[43], some of which are more specific such as the Hall-of-Fame [44], and speciation [45], [46]. However, the important thing to note is that coevolution as a competitive learning process requires some nondeterminism to allow for a variety of different strategies to compete with one another [42]. In this experiment, this is accomplished through a tournament selection that allows a series of random competitions among the neural networks during evaluations. Occasionally, a relatively weak neural network (compared to its peers) may be selected for the next generation because it competed successfully with weak opponents while a stronger neural network competed only with strong opponents. Rather than killing off "weak" neural networks at the current generation, it may be more beneficial in the long run to retain these neural networks for the evolutionary process to fine-tune, resulting in other unique strategies. In contrast, a round-robin tournament that selects only the currently best performing neural networks (relative to their peers) may ultimately limit the potential for evolution in the long run. We have investigated this by modifying slightly the tournament selection module to keep the best neural network of the population for the next generation (two copies if the neural network had already been selected from the tournament competitions). The results are shown in Fig. 13. Compared with the original simulation (Fig. 5), it is obvious that in the modified simulation that evolution produced mostly PNNs at novice level. In another experiment, increasing the number of interactions from five in the original simulation to 15 (making the selection mechanism closer to a round-robin) also resulted in very few PNNs evolved past the intermediate level (Fig. 14), suggesting that a too high selection pressure that inhibits diversity in the population is not preferable in a coevolutionary process.

E. Comparison With the Work of Moriarty et al. in Evolving Neural Networks as Minimax Filters

Finally, we would like to compare the approach used here (evolving neural networks as evaluation functions) and that used by Moriarty and Miikkulainen [5] (evolving neural networks to function as minimax filters). In terms of performance difference, a rough comparison can be made through the use of a computer player based on the positional strategy of Iago [20]. In both approaches, the final neural networks learned mobility strategy and outperformed the positional player. Moriarty and Miikkulainen's neural networks [5] playing at a ply-depth of

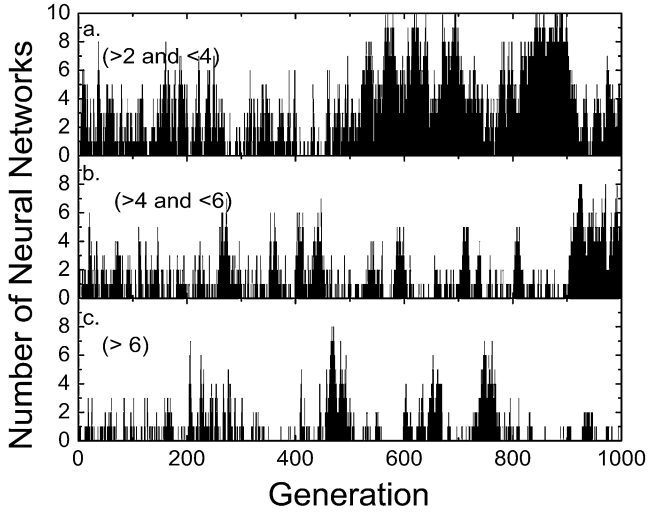

Fig. 13. Evolution in playing ability of the PNNs with generation when the best neural network player (based on comparisons with every neural network in the population) was carried forward to the next generation. Two copies of the same neural network may be carried forward if it was already selected by the tournament. (a) Number of PNNs that can defeat the piece differential player playing at ply-depth of two but lost when it is playing at ply-depth of four. (b) Number of PNNs that can defeat the piece differential player playing at ply-depth of two and four but lost when it is playing at ply-depth of six. (c) Number of PNNs that can defeat the piece differential player playing at ply-depth of two, four, and six.

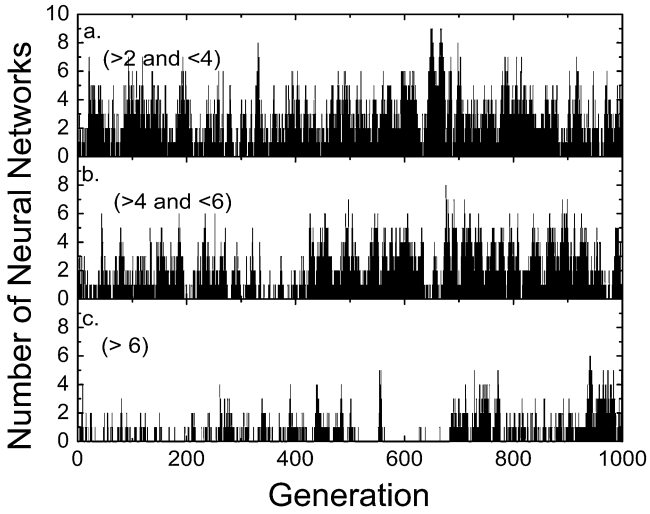

Fig. 14. Evolution in playing ability of the PNNs with generation when the number of tournament games was increased from five to 15. (a) Number of PNNs that can defeat the piece differential player playing at ply-depth of two but lost when it is playing at ply-depth of four. (b) Number of PNNs that can defeat the piece differential player playing at ply-depth of two and four but lost when it is playing at ply-depth of six. (c) Number of PNNs that can defeat the piece differential player playing at ply-depth of two, four, and six.

two were competitive against the positional player playing at ply-depth of four. In this paper, neural networks playing at ply-depth of two outperformed the positional player playing at ply-depth of six (Fig. 6).

While the above comparison is interesting, a more important concern is whether the approach used here can be applied to evolve neural networks to function as minimax filters. We investigated this problem by making suitable modifications on the neural network. The spatial preprocessing layer was retained, and fully interconnected to the output layer of 64 nodes. This kept the complicated neural network manageable at 7687 weights and biases. As in [5], the search window for the minimax was fixed to three.

Result of the simulation showed that the coevolutionary approach could not evolve the neural networks. Only when the population size was increased from 20 to 50 (population sizes of 30 and 40 were also unsuccessful), was it possible to evolve neural networks to the novice level. Coevolution could not produce consistently neural networks that played at intermediate and master levels.

Modifications were made to the evolutionary process to investigate the cause behind the failure of the coevolutionary approach. No improvement was seen when the Gaussian distribution was replaced with a Cauchy, or Gaussian–Cauchy hybrid approach in [33] and [34]. This suggests a self-adaptive mutation operator that uses a fixed distribution is ineffective in evolving the weights and biases of a fixed-architecture neural network to function as minimax filters. An alternative is to replace the self-adaptive mutation operator with differential evolution (DE) mutation operator [29]. This method has been studied and applied in neural networks training [47]. In DE mutation, any change to a particular weight or bias is made by adding a scaled difference of the particular weight or bias of two parents chosen randomly. This eliminates the need to use mutation based on some fixed distribution. This change allowed the coevolutionary approach with DE mutation to evolve neural networks to novice level with a population size of 20. However, it was still difficult to evolve neural networks to higher level of play.

V. Conclusion and Future Directions

Neural networks were evolved successfully to learn to play Othello without relying on expert knowledge. Coevolution improved the level of play of the population of neural networks gradually from novice to intermediate and lastly to master level over 1000 generations. By the end of the simulation, the majority of the neural networks had evolved to outperform players employing both piece differential and positional (based on Iago) evaluation functions playing at ply-depth of six. More importantly, individual neural networks managed to discover different game-playing strategies through coevolution. Some of the neural networks in the later part of evolution even discovered mobility strategies, and could compete with a computer player that uses both the positional and mobility strategies of Iago.

The success met in coevolving fixed-architecture neural networks to represent Othello game-playing strategies through board evaluations during fixed minimax searches was shown to be due to three factors.

1) A spatial preprocessing layer made it possible for neural networks to capture spatial information directly from the board.
2) The use of self-adaptive Gaussian mutation, with individual vector components used to control the mutation step size for every weight and bias independently, allowed the EA great flexibility in adapting mutation step sizes.

3) Diversity introduced into the population by means of a tournament selection process (where neural networks competed with each other in series of random encounters) allowed for new and innovative strategies to develop.

On the one hand, while it is possible to coevolve neural networks as evaluation functions, the same approach does not appear feasible when applied to coevolve neural networks to function as minimax filters. On the other hand, a previous approach that was successful in evolving focus networks to function as minimax filters did not fare well when evolved as evaluation functions. Thus, rather than arguing which one of the approaches is better, it would be more advantageous to develop a hybrid system that combines the strengths of both approaches. For example, such a system would benefit from a modular approach that used one neural network to provide strong evaluation function, while at the same time, utilizing another neural network to filter the minimax so that the hybrid can search the tree deeper and faster without losing accuracy. The main challenge will be to develop an evolutionary approach that can evolve two different neural networks for two seemingly different modular tasks to arrive at a stronger play.

ACKNOWLEDGMENT

The authors gratefully acknowledge the guidance and encouragement of Dr. D. B. Fogel in the early stages of this work and the anonymous referees for their constructive comments. The authors thank the Centre of Excellence for Research in Computational Intelligence and Applications (CERCIA), School of Computer Science, University of Birmingham, Birmingham, U.K., for providing computing resources for some of the experiments. Finally, they would also like to acknowledge and thank the Faculty of Engineering and Technology, Multimedia University, Melaka, Malaysia, for support in earlier work.

REFERENCES

[1] K. Chellapilla and D. B. Fogel, "Evolution, neural networks, games, and intelligence," *Proc. IEEE*, vol. 87, no. 9, pp. 1471–1496, Sep. 1999.

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

[3] K. Chellapilla and D. B. Fogel, "Anaconda defeats Hoyle 6-0: A case study competing an evolved checkers program against commercially available software," in *Proc. 2000 Congr. Evol. Comput.*, 2000, pp. 857–863.

[4] ——, "Evolving neural networks to play checkers without expert knowledge," *IEEE Trans. Neural Netw.*, vol. 10, no. 6, pp. 1382–1391, 1999.

[5] D. E. Moriarty and R. Miikkulainen, "Improving game-tree search with evolutionary neural networks," in *Proc. 1st IEEE Conf. Evol. Comput., IEEE World Congr. Comput. Intell.,* , 1994, pp. 496–501.

[6] ——, "Discovering complex Othello strategies through evolutionary neural networks," *Connection Sci.*, vol. 7, no. 3, pp. 195–209, 1995.

[7] D. B. Fogel, "An introduction to simulated evolutionary optimization," *IEEE Trans. Neural Netw.*, vol. 5, no. 1, pp. 3–14, Jan. 1994.

[8] ——, *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. Piscataway, NJ: IEEE Press, 1995.

[9] T. Back, *Evolutionary Algorithms in Theory and Practice*. New York: Oxford Univ. Press, 1996.

[10] T. Back, U. Hammel, and H. P. Schwefel, "Evolutionary computation: Comments on the history and current state," *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 3–17, Apr. 1997.

[11] D. B. Fogel, "Using evolutionary programming to construct neural networks that are capable of playing tic-tac-toe," in *Proc. 1995 IEEE Int. Conf. Neural Netw.*, 1995, pp. 875–880.

[12] B. Fullmer and R. Miikkulainen, "Using marker-based genetic encoding of neural networks to evolve finite-state behavior," in *Proc. 1st Eur. Conf. Arti. Life, Toward a Practice of Autonomous Syst.*, F. J. Varela and P. Bourgine, Eds., 1992, pp. 255–262.

[13] G. Weiss, "Neural networks and evolutionary computation part I: Hybrid approaches in artificial intelligence," in *Proc. 1st IEEE Conf. Evol. Comput., IEEE World Congr. Comput. Intell.*, 1994, pp. 268–272.

[14] X. Yao, "Evolving artificial neural networks," *Proc. IEEE*, vol. 87, no. 9, pp. 1423–1447, Sep. 1999.

[15] X. Yao and Y. Liu, "A new evolutionary system for evolving artificial neural networks," *IEEE Trans. Neural Netw.*, vol. 8, no. 3, pp. 694–713, May 1997.

[16] K. Chellapilla and D. B. Fogel, "Exploring self-adaptive methods to improve the efficiency of generating approximate solutions to traveling salesman problems using evolutionary programming," in *Evolutionary Programming VI*, P. J. Angeline, R. G. Reynolds, J. R. McDonnell, and R. Eberhart, Eds. Berlin, Germany: Springer-Verlag, 1997, pp. 361–371.

[17] V. W. Porto, M. Hardt, D. B. Fogel, K. Kreutz-Delgado, and L. J. Fogel, "Evolving tactics in computer-generated forces," in *Proc. AeroSense'99 Symp. Enabling Technol. Simulation Sci. III*, A. Sisti, Ed., 1999, pp. 75–80.

[18] D. B. Fogel, P. J. Angeline, V. W. Porto, E. C. Wasson, and E. M. Boughton, "Using evolutionary computation to learn about detecting breast cancer," in *Proc. 1999 Cong. Evol. Comput.*, 1999, pp. 1749–1754.

[19] A. V. Leouski and P. E. Utgoff, "What a Neural Network can Learn About Othello," University of Massachusetts, Amherst, MA, Tech. Rep. 96-10, 1996.

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

[21] K. F. Lee and S. Mahajan, "The development of a world class Othello program," *Artif. Intell.*, vol. 43, pp. 21–36, 1990.

[22] M. Buro, "Improving heuristic mini-max search by supervised learning," *Artif. Intell.*, vol. 134, pp. 85–99, 2002.

[23] J. Schaeffer and H. J. van den Herik, "Games, computers, and artificial intelligence," *Artif. Intell.*, vol. 134, pp. 1–7, 2002.

[24] H. J. van den Herik, J. W. H. M. Uiterwijk, and J. van Rijswijck, "Games solved: Now and in the future," *Artif. Intell.*, vol. 134, pp. 277–311, 2002.

[25] H. Kaindl, "Tree searching algorithms," in *Computers, Chess, and Cognition*, T. A. Marsland and J. Schaeffer, Eds. New York: Springer-Verlag, 1990, pp. 133–168.

[26] J. B. Pollack and A. D. Blair, "Coevolution in the successful learning of backgammon strategy," *Mach. Learn.*, vol. 32, no. 3, pp. 225–240, 1995.

[27] D. B. Fogel and P. G. Harrald, "Evolving continuous behaviors in the iterated prisoner's dilemma," in *Proc. 3rd Annu. Conf. Evol. Program.*, A. V. Sebald and L. J. Fogel, Eds., 1994, pp. 119–130.

[28] P. Darwen and X. Yao, "Coevolution in iterated prisoner's dilemma with intermediate levels of cooperation: Application to missile defense," *Int. J. Comput. Intell. Applicat.*, vol. 2, no. 1, pp. 83–107, 2002.

[29] K. V. Price, "An introduction to differential evolution," in *New Ideas in Optimization*, D. Corne, M. Dorigo, and F. Glover, Eds. New York: McGraw-Hill, 1999, pp. 79–108.

[30] A. Jain and D. B. Fogel, "Case studies in applying fitness distribution in evolutionary algorithms II. Comparing the improvements from crossover and Gaussian mutation on simple neural networks," in *Proc. IEEE 1st Symp. Combin. Evol. Comput. Neural Netw.*, X. Yao and D. B. Fogel, Eds., 2000, pp. 91–97.

[31] A. E. Eiben, R. Hinterding, and Z. Michalewicz, "Parameter control in evolutionary algorithms," *IEEE Trans. Evol. Comput.*, vol. 3, no. 2, pp. 124–141, Jul. 1999.

[32] H. P. Schwefel, *Numerical Optimization of Computer Models*. New York: Wiley, 1981.

[33] X. Yao, Y. Liu, and G. Lin, "Evolutionary programming made faster," *IEEE Trans. Evol. Comput.*, vol. 3, no. 2, pp. 82–102, Jul. 1999.

[34] K. Chellapilla, "Combining mutation operators in evolutionary programming," *IEEE Trans. Evol. Comput.*, vol. 2, no. 3, pp. 91–96, Sep. 1998.

[35] K. Chellapilla and D. B. Fogel, "Fitness distributions in evolutionary computation: Analysis of local extrema in the continuous domain," in *Proc. 1999 Congr. Evol. Comput.*, 1999, pp. 1885–1892.

[36] T. Bäck, "Selective pressure in evolutionary algorithms: A characterization of selection mechanism," in *Proc. 1st IEEE Conf. Evol. Comput., IEEE World Congr. Comput. Intell.*, 1994, pp. 57–62.

[37] D. Billman and D. Shaman, "Strategy knowledge and strategy change in skilled performance: A study of the game Othello," *Amer. J. Psychol.*, vol. 103, no. 2, pp. 145–166, 1990.

[38] K. A. De Jong and W. M. Spears, "An analysis of the interacting roles of population size and crossover in genetic algorithms," in *Proc. Int. Workshop Parallel Problem Solving from Nature (PPSN)*, 1990, pp. 38–47.
[39] D. B. Fogel, G. B. Fogel, and K. Ohkura, "Multiple-vector self-adaptation in evolutionary algorithms," *BioSystems*, vol. 61, no. 2–3, pp. 155–162, 2001.
[40] P. Darwen and X. Yao, "On evolving robust strategies for iterated prisoner's dilemma," in *Progress in Evolutionary Computation*, ser. Lecture Notes in Artificial Intelligence, 1995, vol. 956, pp. 276–292.
[41] P. Darwen and X. Yao, "Every niching method has its niche: Fitness sharing and implicit sharing compared," in *Lecture Notes in Computer Science*, 1996, vol. 1141, Proc. Parallel Problem Solving from Nature (PPSN IV), pp. 398–407.
[42] P. J. Angeline and J. B. Pollack, "Competitive environments evolve better solutions for complex tasks," in *Proc. 5th Int. Conf. Genetic Algorithms*, S. Forest, Ed., 1993, pp. 264–270.
[43] X. Yao and P. Darwen, "An experimental study of n-person iterated prisoner's dilemma games," *Informatica*, vol. 18, no. 4, pp. 435–450, 1994.
[44] C. Rosin and R. Belew, "New methods for competitive coevolution," *Evol. Comput.*, vol. 5, no. 1, pp. 1–29, 1997.
[45] X. Yao, Y. Liu, and P. Darwen, "How to make best use of evolutionary learning," in *Complex Systems—From Local Interactions to Global Phenomena*, R. Stocker, H. Jelinck, B. Burnota, and T. B. Maier, Eds. Amsterdam, The Netherlands: IOS Press, 1996, pp. 229–242.
[46] P. Darwen and X. Yao, "Speciation as automatic categorical modularization," *IEEE Trans. Evol. Comput.*, vol. 1, no. 2, pp. 101–108, Jul. 1997.
[47] J. Ilonen, J. I.-K. Kamarainen, and J. Lampinen, "Differential evolution training algorithm for feed-forward neural networks," *Neural Process. Lett.*, vol. 17, pp. 93–105, 2003.

Siang Y. Chong (M'99) received the B.Eng. (honors) and M.Eng.Sc. degrees in electronics engineering from Multimedia University (MMU), Melaka, Malaysia, in 2002 and 2004, respectively. He is currently working towards the Ph.D. degree in computer science at the University of Birmingham, Birmingham, U.K.

He is a former Tutor for the Faculty of Engineering and Technology, MMU. He is currently a Member of the Natural Computation Research Group, School of Computer Science, University of Birmingham. His research interests include evolutionary computation, neural networks, and game theory.

Mr. Chong is the recipient of the Student Travel Grant for the 2003 Congress on Evolutionary Computation (CEC'03). He is a Member of the Institution of Engineers, Malaysia (IEM).

Mei K. Tan received the Advanced Diploma degree in computer studies from the Informatics Institute, Melaka, Malaysia, in 1995.

She is a Microsoft Certified Systems Engineer (MCSE) NT4. She was previously with the Faculty of Engineering and Technology, Multimedia University, Melaka, as a Technician overlooking computer systems and communication networks of several research laboratories of the faculty.

Jonathon D. White was born in Oakville, ON, Canada, on May 10, 1961. He received the B.Eng. degree in engineering physics from McMaster University, Hamilton, ON, Canada, in 1984, the M.Div. degree from Trinity International University, Deerfield, IL, in 1991, and the Ph.D. degree in engineering physics from McMaster University in 1991.

After completing the Ph.D. degree, he studied Chinese at Beijing Teachers College, Beijing, China, and then worked for two years at Fuji Electric, Yokosuka, Japan, on the development of high power Nd:YAG lasers. This was followed by a year at Ocean University of China, Qingdao, China, a two-year STA Fellowship at Kanagawa Academy of Science and Technology, Kawasaki, Japan, concentrating on nearfield optics, and another year at Xi'an Institute of Optics and Precision Mechanics, Xi'an, China, before becoming a Member of the Faculty of Engineering at Multimedia University, Melaka, Malaysia. In 2004, he joined the faculty of Yuan Ze University, Taoyuan, Taiwan. He is coauthor of the undergraduate textbook *Structuring Data and Building Algorithms* (Malaysia: McGraw-Hill, 2004). He currently collaborates with researchers in China, Malaysia, and Taiwan in the areas of evolutionary computation, neural networks, image processing, computerized analysis of the Chinese language, and the use of high-resolution optical techniques (such as singular molecular detection) to study conjugated polymers and microbiological systems.