An Investigation on Piece Differential Information in Co-Evolution on Games Using *Kalah*

Wee-Chong Oon
National University of Singapore
3 Science Drive 2
Singapore 117543
oonwc@comp.nus.edu.sg

Yew-Jin Lim
National University of Singapore
3 Science Drive 2
Singapore 117543
limyewji@comp.nus.edu.sg

Abstract- This paper decribes a series of experiments using co-evolution of artificial neural networks on a game called *Kalah*. The technique employed closely follows the one used by Chellapilla and Fogel to evolve the successful checkers program Anaconda. The experiments aim to provide insight on the effect of including piece differential information, a basic yet crucial piece of expert knowledge, into the neural network inputs.

1 Introduction

The domain of intellectual games is an excellent testbed for evolutionary machine learning techniques. The nature of games is one of a "micro world", where a small set of rules determines all possible states, thereby avoiding the noisiness of real-world problems. Furthermore, traditional games like chess and checkers are acknowledged to require intelligence to play, and the successfulness of the technique is measured simply by the program's playing strength.

Co-evolutionary techniques have been applied to several games in the past, including chess (Kendall & Whitwell 2001), go (Richards et al 1998, Lubberts & Miikkulainen 2001), and othello (Moriarty & Miikkulainen 1995). However, possibly the most successful demonstration of the machine learning capability of evolutionary computation in the creating of a game-playing program was performed by Kumar Chellapilla and David Fogel in the creation of their checkers program Anaconda (Chellapilla & Fogel 1999, Chellapilla & Fogel 2001, Fogel 2002). This program, created using co-evolution of artificial neural networks without expert knowledge, is able to play checkers at Master level, and is a definite success story for evolutionary computing.

However, the authors in their creation of Anaconda included two pieces of basic checkers information in their neural network designs. The first was piece differential information, i.e. the difference in the total value of all of the player's pieces versus the opponent's. The second was spatial information, i.e. the geographical location of the squares on the board, which is not immediately apparent to the neural networks as the board is represented as a 32-element vector. Both of these pieces of information were regarded by the authors as basic information that even beginning checkers players would know, and hence their inclusion does not constitute "expert knowledge".

This research looks into the effects of the inclusion of the first of these pieces of knowledge, piece differential information. We do so by reproducing Chellapilla and Fogel's work on a simpler game, namely the seed-sowing game of *Kalah*. By using a simpler game with several similarities to checkers, experiments can be conducted in a much shorter time frame and yet are still relevant to the previous work.

In Section 2, we will provide an overview of the techniques used to create Anaconda, and also discuss why we feel the issue of piece differential information is important. We will introduce the game of *Kalah* in Section 3, and explain our reasons for choosing this game for this research. Section 4 describes our experimental setup. Section 5 presents our experimental results and analyses. We will conclude our findings in Section 6, and suggest some possible future research avenues in Section 7.

2 Anaconda: Co-Evolution on Checkers

In 1999, Kumar Chellapilla and David Fogel used the co-evolutionary technique to evolve a program that plays checkers (Chellapilla & Fogel 1999). Without using any form of expert checkers knowledge other than the rules of the game and basic piece differential information, they managed to generate using the co-evolution of artificial neural networks a checkers program that played at Class 'A' level. The neural networks would take as input a board position, and output a value that expresses the desirability of the position for the player to move. These neural networks formed the evaluation function of a basic alpha-beta search algorithm.

Chellapilla and Fogel followed up their work in 2001 by expanding their neural networks to include spatial information, and was able to create a program that played at Master level (Chellapilla & Fogel 2001). They named their program Anaconda, after its ability to restrict the mobility of its opponents in games.

2.1 The Evolution of Anaconda

The Anaconda program is nothing more than the basic alpha-beta algorithm that can be found in most basic AI

textbooks. The key to its playing strength, however, lies in the accuracy of its evaluation function, and Anaconda's evaluation function was generated using the co-evolution of artificial neural networks.

The initial population of the evolutionary process consisted of 15 fully connected feed-forward neural networks, labeled P_i,for i = 1,…,15. Each neural network had 32 input nodes, denoting a particular board position. Each node takes a value from {*-K, -1, 0, 1, K*}, where *K* is the value assigned for a king (the value of *K* differs between neural networks), *1* represents a regular checker, and *0* denotes an empty square. A positive value shows that the piece belongs to the player to move, and a negative value shows that it belongs to the opponent.

These input nodes were fully connected to a hidden layer with 40 nodes, which were in turn fully connected to a second hidden layer with 10 nodes. These final 10 nodes were connected to a single output node that returns a value between –1.0 and 1.0, which is the neural network's evaluation of the input position; a value closer to 1.0 denotes a better position for the player to move, and correspondingly a value closer to –1.0 denotes a worse position. The nonlinear function at each node was the hyperbolic tangent (*tanh* bounded by ±1). Each of the original 32 input nodes were also directly connected to the output node. In addition, piece differential information in the form of the total of all the input nodes was also fed directly to the output node. This gives 1741 connections. See figure 1.

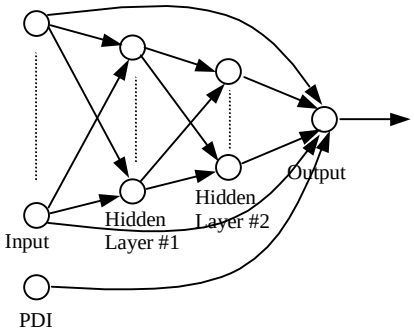

Figure 1: Neural Network Architecture for Class 'A' Version of Anaconda (*Direct PDI*)

In the initial 15 neural networks, the N_w = 1741 connection weights were generated randomly from a uniform distribution over [-0.2, 0.2], with the initial value of *K* set to 2.0. Each connection also had an associated bias $\sigma_i(j)$, for j = 1,…,N_w, which were all set initially to 0.05. Reproduction is achieved solely via mutation (no crossover operation is used). Specifically, for each parent P_i, an offspring P'_i was created by

$$\sigma'_i(j) = \sigma_i(j) \exp(\tau N(0, 1))$$
$$w'_i(j) = w_i(j) + \sigma'_i(j) N_j(0, 1)$$
$$K'_i = K_i \exp((1/\mathrm{sqrt}(2)) N(0, 1))$$

where $\tau = 1/\mathrm{sqrt}(2\ \mathrm{sqrt}(N_w)) = 0.1095$, and $N_j(0, 1)$ is a standard Gaussian random variable resampled for every j. Furthermore, the value of K'_i was constrained to be within [1.0, 3.0], such that the value is reset to the limit if it is exceeded.

In the first generation, each of the 15 initial neural networks produces a single offspring using this mutation method, resulting in a population of 30. These neural networks form the evaluation function for a depth-4 alpha-beta search algorithm with the "quiescent search" refinement of increasing the search depth by 2 if the position contains forced captures.

Each of these 30 players was tested in turn by randomly selecting 5 opponents from the rest of the population; the player plays first, and the opponent plays second. The players are awarded 1 point for a win, 0 points for a draw and –2 points for a loss, the game being adjudged a draw after 100 moves. There were therefore 150 games per generation, and at the end of all the games the 15 players with the highest scores are retained as parents for the next generation, with the remaining 15 discarded. After 250 generations, the authors tested the best-evolved network against human opposition at the free gaming website www.zone.com, and found that it could play at Class 'A' level.

In 2001, Chellapilla and Fogel extended their work by including spatial information into their neural network inputs. Human players, who play the game on a 2D checkerboard, can see the geographical locations of each square. Since the original neural network design expressed the positions as 32-element vectors, the neural networks do not have this information. The authors added this information into the design using an additional hidden pre-processing layer of 91 nodes that covered *n*x*n* square overlapping subsections of the board. Each of the 36 3x3 square subsections of the board were given as inputs to 36 nodes in the pre-processing layer, as well as the 25 4x4 subsections, and so on. These 91 nodes were fully connected to the original 40-node first hidden layer, resulting in 5046 connections. The rest of the experimental setup remained the same.

Due to the increased size of the neural networks, the evolution was run for 840 generations. The best-evolved network (Anaconda) was once again tested at www.zone.com, and it was found to play at Master level. Additionally, Anaconda was able to beat the online version of Chinook, the World Man-Machine Checkers champion and probably the strongest checkers playing entity in the world, at Novice level.

2.2 The Issue of Piece Differential Information

It is important to note that the aim of Chellapilla and Fogel's work was not to create the strongest possible checkers player. Rather, their intention was to show that their evolutionary computing approach would be able to automatically generate a player that can develop the complex strategies required to play a game like checkers without the need for expert knowledge. To that effect, they deliberately refused to include any of the features used in other manually tuned evaluation functions (e.g.

runaway or trapped checkers) other than basic spatial and piece differential information. Viewed in this light, the fact that their evolved program could play at the Master level is impressive.

One question that might arise regarding Anaconda is how great a part the piece differential information played in the playing strength of the program. According to the authors, piece differential information (PDI) is a basic piece of checkers knowledge that any novice checkers player would understand, and therefore it was included as a direct input to the output node of the neural networks. However, in the technical sense, PDI can still be considered "expert knowledge", since it is clear from previous experience that PDI is one of the, if not the most important piece of knowledge required for the playing of checkers. If we take as a baseline a player that simply maximizes piece differential (henceforth referred to as the Piece Differential Player PDP), how long would it take the evolutionary process to evolve a player that surpasses the PDP if PDI is not included in the design?

Another issue concerning the use of PDI is the authors' decision to connect it directly to the output node. Essentially, this precludes any computation on PDI other than a linear relation to the output node. Is this necessarily the best use of PDI? The authors' reasoning no doubt was that since each input had a direct connection to the output node as well, the evolutionary process via these connections could eventually derive PDI. After all, PDI is nothing more than the total of all inputs. Is this assumption correct?

These are some of the questions that this research aims to answer.

3 Kalah

Mancala games refer to a class of sowing games (Erickson 1994) widely played in Africa and Asia, of which there are over 1000 variants known. These games are played using a number of pits (either carved into a board or simply dug out from the ground) and several tokens. In a typical Mancala-type game, a move consists in part of removing all seeds from a pit, and then placing ("*sowing*") them one at a time into successive adjacent pits. The variants of the game are primarily due to different turn-end rules, capture rules and winning conditions.

Kalah is one of the variants of Mancala, and arguably the most popular. The rest of this section presents the rules of Kalah, along with the reasons why we chose Kalah as the basis of our experiments.

3.1 Rules of Kalah

Kalah is played on a board with 2 rows of 6 pits, with a larger scoring pit (or *kalaha*) at either end of the rows. The initial position has 6 tokens (or *seeds*) placed in each of the 12 smaller pits. The two players sit on the north and south sides of the board. The scoring pit to the right of each player belongs to that player. For the purposes of this study, we assume that south moves first. See figure 2.

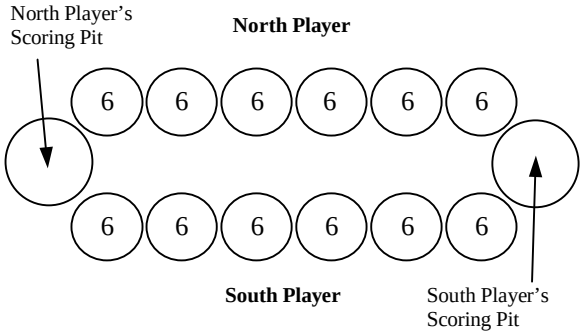

Figure 2: Starting Position for Kalah

The players move alternately. A move consists of removing all seeds from one of the 6 pits closest to the player (the *source pit*), and then placing ("*sowing*") these seeds one at a time into successive pits in an anti-clockwise direction, including his own scoring pit but excluding his opponent's scoring pit. Should the source pit contain enough seeds to go around the board (i.e. it has 13 seeds or more), the source pit itself is skipped. The last pit to receive a seed is called the *destination pit*.

If the destination pit is the scoring pit, the moving player immediately gets another move. This is known as a *move-again*.

If the destination pit is an empty pit on the player's side of the board, and the pit directly opposite the destination pit is non-empty, then all seeds from both said pits are removed and placed into the moving player's scoring pit. This is known as a *capture*.

If the player to move has no moves (i.e. all of his pits are empty), then all remaining seeds are placed into his opponent's scoring pit The game ends when one player has 37 or more seeds in his scoring pit, whereupon that player is the winner. If both players have exactly 36 seeds in their scoring pits, the game is a tie.

3.2 Why Kalah?

Ideally, we would have liked to use checkers for our experiments since this was the domain chosen by Chellapilla and Fogel, whose work we wish to examine. However, checkers can be a long game that lasts over 100 moves, and as a result it takes a long time to evolve a checkers player (e.g. the Master version of Anaconda took 6 months to evolve). Therefore, we needed to select a shorter game, so that our experiments could be completed in a reasonable amount of time.

To that end, we decided on Kalah due to its many similarities to checkers. The branching factor of checkers has been estimated to be 2.84; Kalah has a similar branching factor of about 4. Like checkers, piece differential is possibly the single most important piece of knowledge for playing of Kalah, but other strategies exist. However, unlike checkers, Kalah is a monotonically diminishing game, in that a position can never be repeated

in a particular game (contrast this with Awari, another Mancala variant, where repeat positions are possible), and the numbers of seeds in play always decreases. As a result, games of Kalah seldom last more than 30 moves.

It is almost certain that Kalah is a solvable game, given the fact that Awari, a more complicated Mancala variant, was recently solved (Romein & Bal 2002). Furthermore, a slightly different version of Kalah was solved for all initial board configurations with 1 to 6 pits on the board and 1 to 6 seeds in each pit, excluding the maximum case of 6 pits and 6 seeds (Irving et al 2000). However, this does not matter to our investigation, as Kalah remains an excellent game with which to analyze the effects of the co-evolutionary process.

4 Experimental Setup

The aim of our experiments is to judge the effect of PDI in the co-evolutionary process on evolution time and playing strength. The setup of our experiments follow those used by Chellapilla and Fogel as closely as possible, with the necessary alterations for the game of Kalah (however, we exclude spatial information since it is not immediately clear how such information pertains to Kalah). Therefore, the architectures of each neural network is the same as theirs, except for the following changes: (1) The inputs have 14 nodes representing the current board position, where each input is an integer equal to the number of seeds in the corresponding pit; and (2) The hidden layer architecture is 20/10. This is used because Kalah has less than half the number of inputs as checkers, and for the sake of faster evolutionary time, we halved the number of nodes in the first hidden layer.

The evolutions were performed on 3 different configurations with differing treatments of PDI. *Direct PDI* is the configuration used in Anaconda, where PDI is connected directly to the output node, as shown previously in figure 1. *Indirect PDI* treats the PDI as another input, and connects the PDI to the hidden layer. This configuration is shown in figure 3. *NoPDI* removes the PDI, as shown in figure 4. For Kalah, PDI is the difference between the values of the two scoring pits.

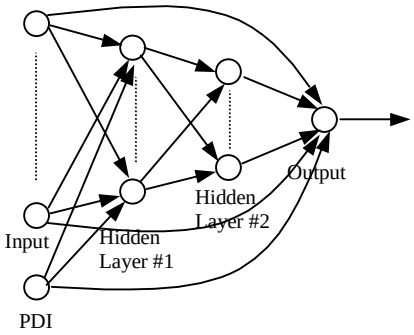

Figure 3: Neural Network Architecture with PDI Treated as Standard Input (*Indirect PDI*)

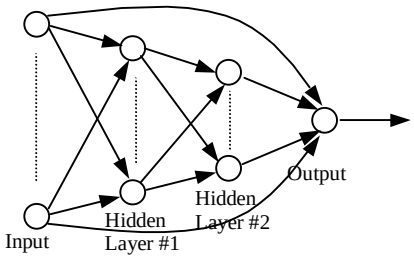

Figure 4: Neural Network Architecture with no PDI (*No PDI*)

During the evolutionary process, where each player plays against 5 random opponents, the players were awarded 2 points for a win, 1 point for a draw and 0 points for a loss. We record the total score of the top 15 players after every generation. When or if this total score reaches a consistent 150 points, this represents a point of stability for the process.

As a baseline comparison, we make use of the piece differential player (PDP) with a depth-4 search. Every generation, we take the top 5 evolved neural networks and play them against PDP as both the first and second player at a depth 4 search. This is done as a measure of the evolved networks' playing strength as the co-evolution process goes on.

In any case, we terminate our experiments after 1000 generations.

5 Results and Analyses

After 1000 generations, we competed the top neural network for each setup against each other as both first and second players. The results showed an obvious hierarchy in playing strength. *DirectPDI* won all its games against both *IndirectPDI* and *NoPDI*. Similarly, *IndirectPDI* won both its games against *NoPDI*. Hence, the order of playing strength of the three setups is (1) *DirectPDI*, (2) *IndirectPDI* and (3) *NoPDI*.

For the rest of this section, we will analyze each setup more closely in turn, and try to explain these results.

5.1 No PDI

NoPDI is the most basic setup, which contains no PDI in the design. Chellapilla and Fogel's reasoning for including PDI is that it is information that any beginner checkers player would know, and therefore there is no reason why the neural networks should have to discover this information on its own. After all, this information exists in the basic board position.

In our experiment, *NoPDI* arrives at one stabilization point after about 90 generations, where the top 15 players in the evolution all achieve maximum score. This stable period continues until about generation 470, when it manages to climb out of this local optimal. The evolution

progresses until about generation 700, when the second and final stable period is reached. See figure 5.

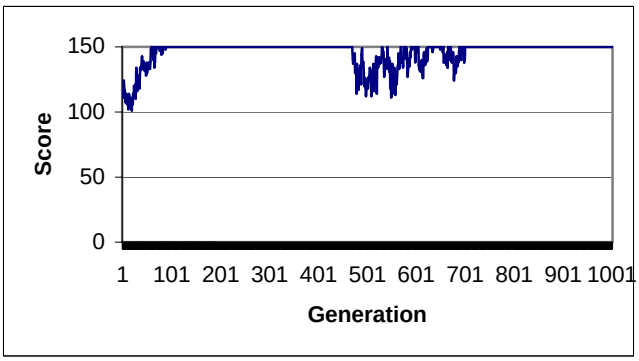

Figure 5: Combined Scores of Top 15 Players During Evolution for *NoPDI*

The playing strength of the evolved players in the first stabilization period is exceedingly poor. In fact, it loses every game as both first and second player against PDP during this period. Therefore, without PDI, the evolved players are unable to reach the level of a PDP at this point. In the second stable period, *NoPDI* is able to win most of its games as first player against PDP, but none as second player. This suggests that while *NoPDI* is able to evolve into a good Kalah player if it moves first, it is unable to develop any strategy as second player. The results of *NoPDI* against PDP are given in figure 6, which is a stacked bar graph. Wins against PDP are awarded 2 points, draws 1 and losses 0. The first player results are above the second player results.

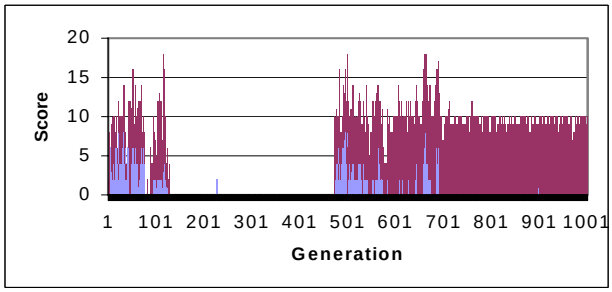

Figure 6: Scores vs. PDP for *NoPDI*

NoPDI eventually manages to evolve a satisfactory Kalah player that wins most of its games playing first against PDP. While it is able to fit the criteria given in the evolutionary process (i.e. it is able to win as first player), it is unable to do much more than that. It is reasonable to say that *NoPDI* is not much stronger than a piece differential player.

5.2 Direct PDI

The *DirectPDI* configuration is the one used for Anaconda, where the PDI is a direct input to the output node. In this case, the only difference with the *NoPDI* setup is a single input node with a single connection to the output node.

It took about 650 generations for *DirectPDI* to stabilize. From then on, until our limit of 1000 generations, the top 15 players continued to achieve the maximum score during evolution. See figure 7.

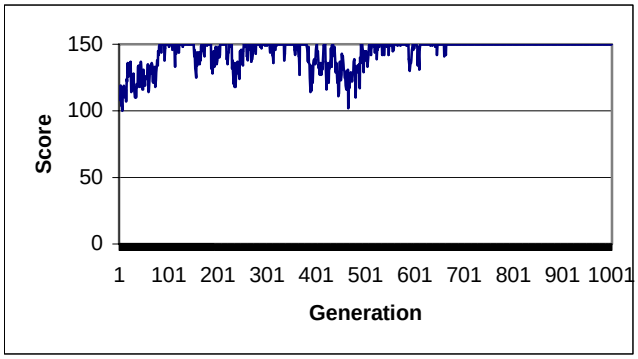

Figure 7: Combined Scores of Top 15 Players During Evolution for *DirectPDI*

From the stabilization point of around generation 650 onwards, the top 5 players in *DirectPDI* beat PDP in practically every game as both first and second player. This shows that it was able to achieve a playing strength above the criteria of the evolution, which only required the neural networks to do well as first player. This is shown graphically in figure 8, where the maximum score is reached after generation 650.

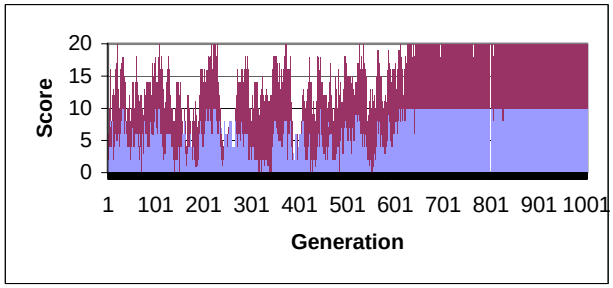

Figure 8: Scores vs. PDP for DirectPDI

One possible explanation of how *DirectPDI* is able to play well as second player is the way it treats PDI. Since PDI is a direct connection to the output node, *DirectPDI* can essential simulate PDP using simply this node. The rest of the neural network setup can then be used to seek out strategies beyond PDI, which is what is required to play well as second player.

5.3 Indirect PDI

The *IndirectPDI* configuration treats PDI as simply another input node, and feeds this information into the hidden layers (as well as directly to the output node).

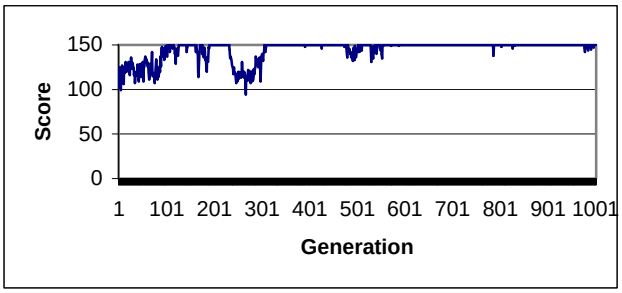

Figure 9: Combined Scores of Top 15 Players During Evolution for *IndirectPDI*

IndirectPDI does not appear to fully stabilize, even after 1000 generations. While there are detectable periods where the top 15 evolved players achieve the maximum score, there are still times up until the very last generation where a significant change in playing strength is being discovered. See figure 9.

Because it has not yet reached a final stabilization period, its results against PDP are still wildly fluctuating. However, it is clear that *IndirectPDI* has better success playing second than *NoPDI*, and is therefore a better player overall. However, compared to *DirectPDI*, *IndirectPDI* after 1000 generations is obviously weaker. See figure 10.

Even though *IndirectPDI* has not stabilized, it is already obviously stronger than having no PDI whatsoever. However, note that the only difference between the architecture of *IndirectPDI* and *NoPDI* is an additional input node that contains nothing more than the difference between two other nodes (representing the scoring pits). Therefore, simply adding a very simple preprocessing node significantly increases the playing strength. It is clear that PDI, although "basic" knowledge that any beginning Kalah player would know, must be considered "expert" knowledge to the evolutionary process.

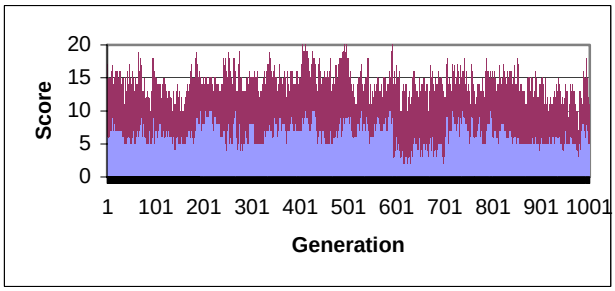

Figure 10: Scores vs. PDP for *IndirectPDI*

Further, note that *IndirectPDI* is the same as *DirectPDI*, except that it has 20 additional connections between the PDI input node and the first hidden layer. Therefore, conceptually, *IndirectPDI* must be computationally more powerful than *DirectPDI*. However, *DirectPDI* evolved to be a much stronger player in a faster timeframe. This shows that the appropriate neural network architecture can aid the co-evolutionary process in arriving at a good solution. It is possible that the additional connections in fact interfered with the usage of PDI in the playing strategy.

6 Conclusions

This research looked into the effect of different treatments of PDI in the co-evolutionary process, using Kalah as the testbed. Three configurations of neural networks were used, namely *NoPDI*, *IndirectPDI* and *DirectPDI*. After the experiments of 1000 generations, it was found that the ranking in ascending order of playing strength for the three approaches were as listed above.

The setup with no PDI information was unable to achieve a playing strength that was much better than a piece differential player. It could beat PDP as first player, but loses all games as second player.

By simply adding an additional node containing PDI and connecting it directly to the output node, the playing strength increased dramatically. The final players could beat PDP in all games, as both first and second player. It is obvious that PDI is a significant piece of information, and there are doubts to whether the co-evolutionary process could ever arrive at a strategy equivalent to *DirectPDI* with the *NoPDI* setup. Therefore, PDI must be considered expert knowledge.

IndirectPDI treated the PDI as an additional input to the hidden layers. This configuration is exactly the same as *DirectPDI*, but with additional connections. However, these additional connections extended the time required for stabilization to beyond 1000 generations, and therefore was detrimental to the effort to create a strong player in a reasonable amount of time. Theoretically, the *IndirectPDI* setup would achieve a playing strength at least equal to *DirectPDI*, but this may take many additional generations.

7 Future Work

The effect of PDI is just one of several aspects of the co-evolutionary process that should be examined. When Chellapilla and Fogel designed their experiments while creating Anaconda, most of the parameters were decided either arbitrarily or heuristically. For example, the 40/10 hidden layer architecture was used because a similar design proved successful in Fogel's previous tic-tac-toe experiments (which used a 20/10 architecture). While their design successfully created a Master level checkers player, there is no evidence that it is anywhere close to optimal. The obvious question then would be, how can the design be improved to create an even stronger program?

More research is needed to understand the relative importance of the parameters. These include the neural

network architecture; size of the neural networks; size of populations; mutation method; scoring method; search depth; and additional expert knowledge. Such experiments can be conducted using Kalah or a similar game.

Games like checkers and Kalah tend to confer a significant advantage to the first player. In such cases, programs using a single evaluation function may be unable to play well as both first and second player. This suggests that competitive co-evolution (Rosin and Belew, 1996, Lubberts & Miikkulainen 2001), where two populations simultaneously evolve strategies to overcome the opposing population, may be a fruitful avenue. Using this method, two separate neural networks, one for the first player and the other for the second player, can then be evolved.

Since Anaconda's performance has shown that the process is capable of producing a strong game playing program, another possible research direction is to use evolutionary computing to produce the strongest game playing program possible. With such an aim, we no longer have to restrict ourselves to the barest minimum of expert information and techniques, like Chellapilla and Fogel did. Therefore, the multitude of game-playing techniques like opening books and endgame databases can be included, both during the evolutionary process and in the final program.

Acknowledgments

We would like to thank Dr. Martin Henz for all his help and advice in this research.

Bibliography

Chellapilla K. and Fogel D. (1999) "Evolving Neural Networks to Play Checkers without Expert Knowledge", *IEEE Trans. on Neural Networks*, vol. 10, no. 6, pp. 1382-1391.

Chellapilla K. and Fogel D. (2001) "Evolving an Expert Checkers Playing Program without using Human Expertise", *IEEE Trans. on Evolutionary Computation*, vol. 5, no. 5, pp. 422-428.

Erickson J, (1994) "Sowing Games", in *Games of No Chance*, Nowakowski R. (ed.), pp. 198-298, University Press.

Fogel D. (2002) "Blondie24: Playing at the Edge of AI", Academic Press, London, UK.

Irving G., Donkers J., and Uiterwijk J. (2000), "Solving Kalah", *International Computer Games Association* (*ICGA) Journal*, Vol. 23, No. 3, pp. 139-147.

Kendall G. and Whitwell G. (2001) "An Evolutionary Approach for the Tuning of a Chess Evaluation Function using Population Dynamics", *2001 IEEE Congress on Evolutionary Computation (CEC 2001)*, pp. 995-1002.

Lubberts A. and Miikkulainen R. (2001) "Co-Evolving a Go-Playing Neural Network", in *Coevolution: Turning Adaptive Algorithms upon Themselves*, Belew R. and Juille H (eds.), pp. 14-19.

Moriarty D. and Miikulainen R. "Discovering Complex Othello Strategies Through Evolutionary Neural Networks", *Connection Science*, vol. 7, no. 3-4, pp. 195-209.

Richards N.; Moriarty D.; McQuesten P.; and Miikkulainen R. (1998) "Evolving Neural Networks to Play Go", *7th International Conference on Genetic Algorithms*.

Romein J. and Bal H. (2002) "Awari is Solved" (note), *International Computer Games Association* (*ICGA) Journal*, vol. 25, no. 3, pp. 162-165.

Rosin C. And Belew R. (1996), "A Competitive Approach to Game Learning", *9th Annual ACM Conference on Computational Learning Theory (COLT-96)*, pp. 292-302.