An Investigation of an Evolutionary Approach to the Opening of Go

Graham Kendall, Razali Yaakob
School of Computer Science and IT
University of Nottingham, UK
{gxk|rby}@cs.nott.ac.uk

Philip Hingston
School of Computer and Information Science
Edith Cowan University, WA, Australia
p.hingston@ecu.edu.au

Abstract – **The game of Go can be divided into three stages; the opening, the middle, and the end game. In this paper, evolutionary neural networks, evolved via an evolutionary strategy, are used to develop opening game playing strategies for the game. Go is typically played on one of three different board sizes, i.e., 9x9, 13x13 and 19x19. A 19x19 board is the standard size for tournament play but 9x9 and 13x13 boards are usually used by less-experienced players or for faster games. This paper focuses on the opening, using a 13x13 board. A feed forward neural network player is played against a static player (Gondo), for the first 30 moves. Then Gondo takes the part of both players to play out the remainder of the game. Two experiments are presented which indicate that learning is taking place.**

I. Introduction

Go is an ancient game, which originated in China at least 2000 years ago and was known originally as Weiqi [1]. The name Go came from the shortened Japanese word Igo. Automated Go programs began appearing in the 1970s [2] and 1980s [3, 4, 5, 6]. Since that time, Go has been the subject of research areas such as machine learning [7, 8, 9], cognitive science [10], and evolutionary computing [11, 12], but automated Go programs are still far from the standard of human experts [13].

The games of Go, chess, and checkers are similar, in that they are finite, perfect information games, played between two players. One of the best-known chess programs, Deep Blue [14], achieved world champion status when, in 1997, it beat Garry Kasparov by a score of 3.5-2.5. Utilising custom-built hardware [15], Deep Blue executed sophisticated search algorithms to analyse up to 200 million positions per second.

Chinook, a checkers program, developed by Jonathan Schaeffer's team at The University of Alberta won the world checkers title in 1994 [16, 17]. Chinook used an opening, and endgame database and an extensive checkers knowledge base.

Chellapilla and Fogel [18], also considered checkers, developing Blondie24 - a checkers player, which learned its strategy using a co-evolutionary approach. The current board position in Blondie24 is evaluated using an artificial neural network with the weights of the network being evolved via an evolutionary strategy. The output of the network is used in a minimax search. Blondie24 was not provided with any domain knowledge. It received points based on whether it won, lost, or drew games and was not even told how it had performed in each individual game. The conclusion of Fogel and Chellapilla is that, without any expert knowledge, a program can learn to play a game at an expert level using a co-evolutionary approach. Additional information about this work can be found in [19, 20, 21, 22].

Go is considered more complex than chess or checkers. One measure of this complexity is the number of positions that can be reached from the starting position which Bouzy and Cazenave estimate to be 10^{160} [23]. Chess and checkers are about 10^{50} and 10^{17}, respectively [23].

The size of the search space is the primary reason why a good automated Go player is difficult to produce. To reduce the search space, the board can be scaled down either by reducing the board size or by dividing the board into sub-boards (decomposition). These techniques can reduce the computational effort required for playing the game, which is one of the main constraints in producing an effective automated Go player. The early work for small boards can be found in [24, 25] and decomposition approaches are reported in [26].

The game of Go can be divided into three general phases: the opening game, the middle game, and the endgame [27]. Researchers have applied a variety of techniques to the middle game, which is considered the most difficult – yet important – phase [27]. However, Ishigure [28] states that the opening game is the most important phase. In order to make territory, the best place to move is in a corner and the least valuable moves are those in the centre of the board. The number of plays in the opening is not predetermined [28]. Aggressive player may start to attack the opponent (thus entering the middle game) just a few turns after the start of the game, whilst other openings may last longer.

Playing just the opening is the approach we adopt here. In this paper, we are not attempting to reduce the size of the search space or incorporate any Go knowledge. The objective of this research is to investigate a population of evolutionary neural networks, which play the opening of Go and compete to survive to the next generation. A feedforward neural network player is played against a static player, Gondo (written by Jeffrey Greenberg [29]). We play against Gondo for the first 30 turns (15 turns for each player). At that time Gondo takes the part of both players and plays out the remainder of the game to determine the outcome. An evolutionary strategy is used to evolve the networks. We use a 13x13 board.

Previous authors have investigated evolving neural networks to play Go. Richard et al. evolved neural networks using the SANE[1] (Symbiotic, Adaptive Neuro-Evolution [30]) method [27]. Board sizes of 5x5, 7x7, and 9x9 were used in evolving networks to play on small boards against a simple computer opponent called Wally[2]. Each intersection on the board has two input units and one output unit. Any positive output indicates a good move, the larger the value, the better the move. If none of its outputs is positive, the network will pass its turn. SANE required 20 generations to defeat Wally on a 5x5 board, 50 generations on a 7x7 board, and on a 9x9 board, 260 generations were needed. Donelly et al. also evolved neural networks via self-play [31].

Other researchers have used neural networks to learn to play Go using machine learning methods such as supervised learning (training on moves from professional games) [32] and reinforcement learning [33, 34].

Yen et al. took a different approach to learning to play Go openings with Jimmy 4.0, which has been ranked 4 *kyu*[3] when they adopted 100 questions from Go books to test the rank of their opening game [35]. Jimmy plays the opening stage by trying to occupy the corners (called joseki - a series of standard moves in corners or sometimes on edges), extending the edges, and dealing with Moyo (large potential territories). Pattern matching techniques are used to occupy the corners and generate the Moyo moves. In extending the edges, two techniques are used. Pattern matching generates moves near the existing stones on the board. A "split" method places a stone equidistant between two stones, which are near opposite corners.

For an excellent review of the state of the art in computer Go programming, we recommend Müller [36] and as a starting point for further reading, Enzenberger's online bibliography [37]. Other reviews focussing on AI approaches may be found in [38, 39].

II. Rules of Go

The aim of Go is to gain as much territory as possible. Players will try to capture an opponent's stones and also define areas as their own territory. At the end of the game, the winner will be identified by counting territories using one of two scoring methods; territory scoring or area scoring. Japanese rules use territory scoring. Surrounded territory is counted, plus the number of captured opponent stones. In area scoring, used when playing under Chinese rules, surrounded territory will be counted plus the live stones on the board. We follow the Japanese scoring method.

Differences in playing strength can be balanced by a handicap system, which allows the Black player to place several stones on the board before the start of the game.

Go is played between two players, Black and White. The game starts with an empty board. Stones are placed on any empty intersection on the board with players making alternate moves. Black plays first. Go players can pass their turn at any time but consecutive passes ends the game.

A stone on an intersection has a set of liberties (empty intersections next to a stone). Figure 1a shows a white stone with four liberties. When the opponent places a stone on any neighbouring intersection, white will lose one liberty. In Figure 1b, black stones surround the white stone, which means white has no liberties and is captured and removed from the board (Figure 1c). This territory now belongs to black.

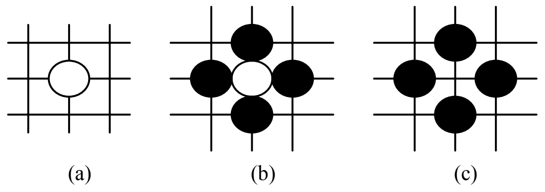

Figure 1: (a) White stone has 4 liberties, (b) Black stones are surrounding a white stone, (c) White stone has been removed and the black player gets 1 point of territory.

Placing a stone that does not make a capture, and has no liberties, is illegal. For example, the white player cannot place a stone in the centre of black territory in Figure 1c. Placing a stone that results in a repeating position seen after the same player's previous play is also illegal. This is called the *ko-rule*.

Figure 2a shows a marked white stone that has only one liberty. Black captures the marked white stone by playing at position *k*. After the marked white stone has been removed from the board, the white player cannot place a stone on *r,* Figure 2b, before the board position has changed (i.e. white must play on another intersection before being able to place a stone on *r*).

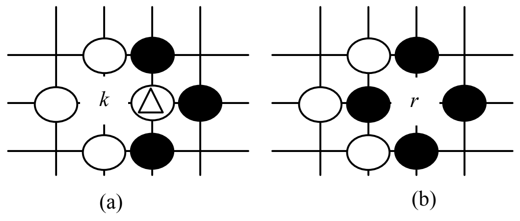

Figure 2: Repeating position.

Figure 3a shows a string, which is when a number of stones of the same colour are connected along edges or a set of connected stones. White's string in Figure 3a, excluding a stone marked with 1, has eight liberties. To

1 For more details about SANE, please refer to [30]. The source code is available from www.cs.utexas.edu/users/nn/.

2 A simple Go program written by Bill Newman.

3 Human Go players are ranked by *kyu* (student) and *dan* (master). Players start at high *kyu* grades of 20 or more, and progress towards 1 *kyu* and then progress onto the lowest master grade of 1 *dan* and work upwards.

capture the string, black stones must surround it until the string has no liberties (Figure 3b). Figure 3c shows the position once the white stones have been removed from the board. The black player now has four points of territory.

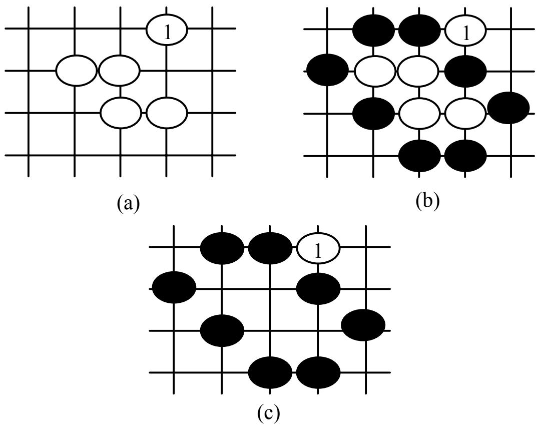

Figure 3: (a) String of white stones excluding stone 1, (b) String had been captured by opponent stones, (c) String of white stone have been removed from the board.

When a string surrounds one or more empty intersections, it will form *eyes*. Figure 4a shows an eye, *E,* and Figure 4b shows two eyes, *E* and *F* for a white string. To be captured, the string must first be surrounded by the opponent and then the eye filled. Single eyes can be captured but two eyes cannot be captured because black player cannot place a stone either on *E* or *F*. If black tries to place a stone in *E*, it is forbidden because each white stone still has one liberty, i.e., *F*, and vice versa. Any string is considered to be dead if there is no way to stop it from being captured or considered alive if the opponent cannot capture it. The white stones in Figure 4b are alive.

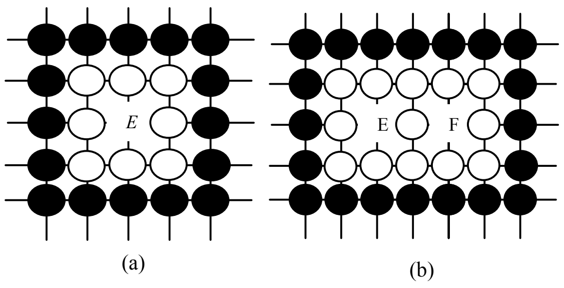

Figure 4: Eyes in Go.

Of course, the game of Go has many strategic elements and good introductions can be found in [40, 41].

III. METHOD

We instantiate a population of neural networks and each one plays against a static player (Gondo) for the first 15 turns (30 turns in total). After this time, the current position for the neural network player is copied to another version of Gondo and it plays against itself until end of the game. Figure 5 shows the structure for our evolving system.

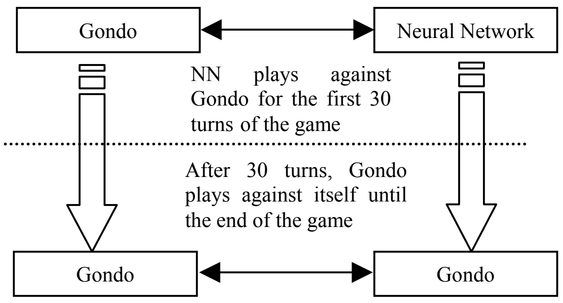
Gondo
Neural Network
NN plays against Gondo for the first 30 turns of the game
After 30 turns, Gondo plays against itself until the end of the game
Gondo
Gondo

Figure 5: Go, evolutionary model.

Each intersection on the board is represented by −1, 0, or +1 which indicates a black stone, an empty intersection or a white stone respectively. An evaluation function is structured as a feed forward neural network, which is used to evaluate the current board position. A hyperbolic tangent (tanh, bounded by ±1) function is chosen as a non-linear function for each hidden and output node.

At the end of each game, the difference in score between the neural network player and Gondo will be used as a fitness for that neural network player. At the end of each generation, players will be sorted in descending order according to their fitness. Half the players, those with the highest fitness, are selected and survive to the next generation. Copies of these players are mutated to create the new population.

Two experiments are presented. Firstly, a Go program that uses a minimax search tree as a look ahead to predict possible moves, and secondly, a Go program that does not use minimax as a look ahead. The methods used in this research are inspired from [18, 19, 20, 21, 22], with some minor modifications.

A. Neural Network with Minimax

In the first experiment, a feed-forward neural network is used to evaluate the current board position and a minimax search tree is used to look ahead two-ply. The neural network consists of 169 input nodes (i.e the intersections on a 13x13 board), 55 hidden nodes, and an output node. The number of hidden nodes (55) was chosen to be about one third of the number of input nodes. This appears to be a good rule of thumb in the absence of any other data. Figure 6 shows this architecture.

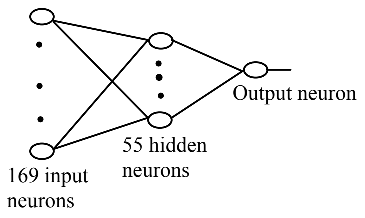

Figure 6: A structure of the feed-forward neural network.

The algorithm for this first experiment is as follows:

1. Build a search tree to two-ply, starting from the current board position.
2. Assign a value to the leaf nodes using the output of the neural network.
3. Propagate the values to the root of the tree (using minimax and alpha-beta pruning)
4. Choose the best move.

B. Spatial Neural Network without Minimax

In a second experiment, we do not use a minimax search tree. Three-layer, feed-forward neural networks are used. The networks have 169 nodes in the input layer, 286 and 95 nodes in the first and second hidden layer respectively. The networks have a single output node. The first and second hidden layers, and the output node are fully connected.

The first hidden layer is structured to cover *m* x *m* overlapping subsections of the board, i.e. 3x3, 5x5, 7x7, 9x9, 11x11, and 13x13 (full board). The 3x3 subsections provides input to the first 121 neurons in the first hidden layer; it is followed with 81 neurons from the 5x5 subsections, and so on to one subsection of the 13x13 board. These resulted a total node for the first hidden layer is 286 and the total node for the second hidden layer is about one third of the first hidden layer.

Figure 7 shows an example of overlapping 3x3 and 5x5 subsections. This idea of using spatial information is taken from Fogel's work on the co-evolution of a checkers player [22].

Figure 8 shows a spatial neural network with a part of subsection from Figure 7. Node *p* in the first hidden layer will receive inputs from intersections indexed 0a, 0b, 0c, 1a, 1b, 1c, 2a, 2b, and 2c (9 intersections from one 3x3 subsection), and their associated weights. Node *q* then will receive inputs from intersections indexed 0b, 0c, 0d, 1b, 1c, 1d, 2b, 2c and 2d, and their associated weights.

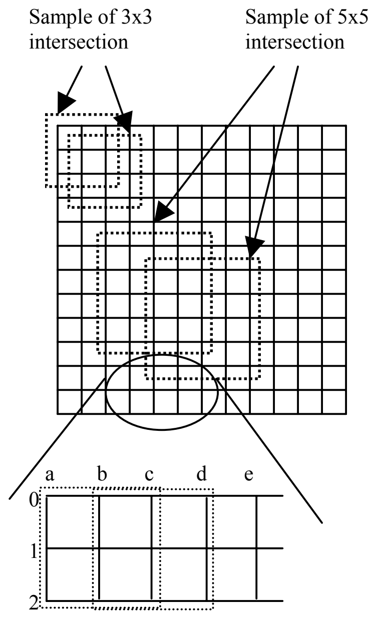

Figure 7: Example of 3x3 and 5x5 subsection intersection.

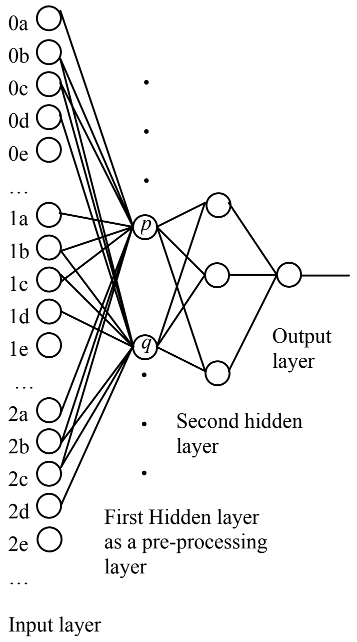

Figure 8: The architecture of the "spatial" neural network with example connections for a 3x3 subsection intersection.

The algorithm for the second experiment is given below:

1. Place a stone on any available and legal intersection.
2. The current board state is input to the neural network, which outputs a value.

3. Remove the stone placed in step 1.
4. Repeat step 1 to 3 until no more legal intersections are available.
5. Select the intersection with the highest output value from the neural network. Select randomly from equal values.

All weights in the neural network for both experiments are mutated at each generation using an evolutionary strategy. Each weight has an associated self-adaptive parameter, σ, which serves to control the step size of the search for the mutated parameters of the neural network [22].

Initially, a Gaussian random generator is used to generate all weights. The self-adaptive parameter and the weights are then adapted by

$$\sigma'_j = \sigma_j \,.\, exp\,(\tau \,.\, N_j(0,1)), \qquad (1)$$

$$w'_j = w_j + \sigma'_j . N_j(0,1)\ , \qquad (2)$$
$$j = 1, \ldots, N_w$$

Where:

N_w is a number of weights, w_j, in the neural network

$N_j(0,1)$ is a standard Gaussian random variable resampled anew for every j

τ is a learning rate, which is $\tau = (2N_w^{0.5})^{-0.5}$

In fact, formula (1) is taken from [19, 20, 21, 22].

IV. Experiments And Result

Below we present the results from the above two experiments. Both were run on a Pentium 4, 512 MB RAM.

A. *Experiment on Evolved Neural Network with Minimax Search Tree*

In the first experiment, a population of 10 networks was evolved, with the best 5 (copies and mutations) from one generation surviving to the next. The experiment ran for 20,000 generations with an elapsed time between two generations being about 228 seconds. This experiment ran for about two months. Every 1000th generation, the best player was kept and these were played against each other at the end of the experiment. This resulted in 21 "best" players (0th to 20,000th generation).

Table I shows the results, with x being the generation for each 1000th generation, and y is the total points after the best player from that generation has played against the best player from all other generations.

TABLE I. Total Points for each Best Player with Minimax

x (x10^3)	y	x (x10^3)	y
0	14	10	8
1	17	11	44
2	32	12	44
3	-1	13	23
4	15	14	8
5	26	15	26
6	26	16	14
7	14	17	29
8	5	18	23
9	35	19	20
		20	26

Based on Table I, we can plot a graph (Figure 9). Each point in the graph represents the total points for each best-evolved player after playing against each of the other players when played as black and white, i.e. each player played 40 games in total. The total points are based on +2 points for win, -1 for a loss and 1 for a draw. This points mechanism is only used to calculate the winning, losing and drawing points for each player. It is never used for parent selection. We did try other scoring mechanisms but the results were similar.

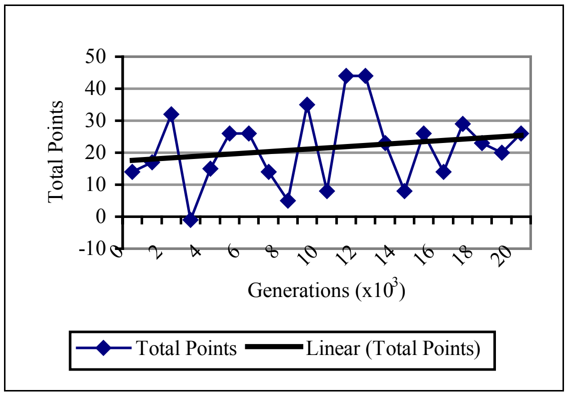

Figure 9: Total points every 1000th generation against each other

The graph, although looking chaotic, does indicate a gradual improvement, as shown by the trend line.

We noted that some players could not beat players from previous generations. For example, a player from the 10,000th generation won against a player from the 9,000th generation but lost when playing against the 11,000th generation. This is what we would expect (at least what we would hope for). However, the 10,000th player lost against the 2,000th player when playing as black. This, we believe, is due to the fact that, due to the complexities of Go, we are only at the very early stages of the learning process and there is not yet a significant difference between players just a few thousand generations apart.

B. *Experiment on Evolved Spatial Neural Network without Minimax*

In the early stages of the second experiment, we did try to use the spatial neural network with the minimax search tree (with a population size of 10). The elapsed time between two generations was approximately about 1548 seconds. Due to this slower processing time, we decided to remove the minimax search tree from the program. The objective was now to investigate whether learning will occur without any look ahead.

In this second experiment, the size of the population is 20 and we ran for 60,000 generations, where the elapsed time between two generations was approximately about 117 seconds. We can increase the population size and the number of generations as there is no look ahead (i.e. minimax search). This experiment ran for about three months.

At each generation, copies of the best 10 players were mutated and competed in the next generation. Every 10,000th generation, the best-evolved player was kept and these were played against each other at the end of the experiment. This resulted in seven best players (0th to 60,000th generations). Table II shows the results based on the total number of points at each generations (using the same scoring mechanism as the first experiment).

TABLE II. TOTAL POINTS FOR EACH BEST PLAYER WITHOUT MINIMAX

$x(\text{x}10^4)$	y
0	-7
1	3
2	12
3	15
4	1
5	10
6	11

Based on Table II, we can plot a graph as Figure 10 and a linear regression line.

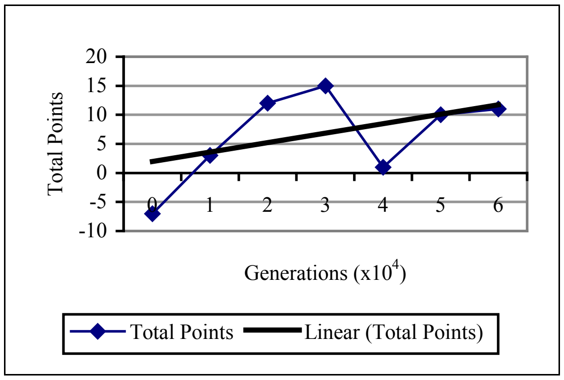

Figure 10: Total points every 10,000th generation against each other.

The trend line indicates a process of learning is taking place. Without any look ahead, the program is totally dependent on the evaluation function. There is no significant difference between players just a few thousand generations apart and because of that, in the second experiment, we only kept best player at every 10,000th generation.

V. CONCLUSION AND FUTURE WORK

Based on the results from above, some learning has occurred even though no Go knowledge or information was given to the program. A positive linear regression gradient in both experiments indicates learning has taken place, even though the learning appears erratic. This, we believe, is due to the fact that we are only at the initial stages of the learning process.

We only collected data at every 1,000 games (first experiment) and 10,000 games (second experiment). The data, as the graphs show (figures 9 and 10) appears chaotic, although there is an underlying upward trend. However, due to the small number of data points collected, we cannot calculate if these results are statistically significant. For our future work we will collect data at more frequent intervals so that valid statistical tests can be conducted. Unfortunately, we cannot collect data for this paper due to excessive run times involved.

Evolving neural networks has the potential as a technique for evolving automated Go players. However, more generations are needed to determine whether the program can become a good player. We are continuing to run our experiments and we are also seeking access to faster machines.

Of course, a search depth of two is unlikely to be enough to predict good moves, but deeper searches come at the cost of increased computational time. One of the ways to reduce the running time for each generation is by taking out the minimax search from the program, as we did in the second experiment. The program can still learn even without look ahead, but the program probably needs more generations to do so.

This research also shows that the program can learn through experience via playing against a static player. The program might learn more efficiently if we had access to more than one static player or adopted a co-evolutionary approach by allowing the players to play against each other at some point in the learning process.

In addition to addressing the questions that we have raised as part of this work (e.g. more computation time (i.e. generations), deeper searches, whether we require look-ahead, co-evolution, playing against other static players, network architecture etc.), we would also like to investigate the following:

Passing mechanism: If the automated player makes an illegal move we currently consider that a *pass*. We would like to give the automated player another chance to choose a move.

Play the middle game and end game: If we can make further progress on evolving good opening game strategies then we would like to apply similar techniques to the middle and end game. The middle game would be particularly challenging, as there is a lot of strategic play in this part of the game. For example, can the player learn how to attack an opponent or how to save/protect its stones?

Play with handicap: We would like to experiment with starting with an automated player that starts with a high handicap, which is gradually reduced as it improves its play.

Determine strength differences among the players: The results we have presented are only based on each player playing against each other as black and white (i.e. two games against each other player). We would like to carry out more work to determine if there is a statistical difference between the best players at each generation.

In summary, we have provided some evidence that an evolutionary approach can produce learning for an automated Go player. Considering that Deep Blue [15] and Chinook [17] utilised custom-built hardware to achieve their landmark results, and given that Go, is considered more complex, this can be seen as an achievement. At the present time, no computer program can play Go at human levels and perhaps an evolutionary approach may be one way to achieve this goal.

REFERENCES

[1] Nationmaster.com. Encyclopedia: Go (board game). Available: http://www.nationmaster.com/encyclopedia/Go-(board-game)

[2] A. Zobrist, "A Model of Visual Organization for the Game of Go," in Proceedings of AFIPS Spring Joint Computer Conference, Boston, AFIPS Press, Montvale, New Jersey, 1969, pp. 103-111.

[3] B. Wilcox, "Computer Go," originally published in: American Go Journal, vol. 13, no. 4, 5, 6; vol. 14, no. 1; vol. 14, no. 5, 6; vol. 19 (1978, 1979 and 1984).

[4] W. Reitman and B. Wilcox, "Perception and Representation of Spatial Relations in a Program for Playing Go," originally published in: Proceedings of the ACM Annual Conference, 1975, pp. 37-41.

[5] W. Reitman and B. Wilcox, "Pattern Recognition and Pattern-Directed Inference in a Program for Playing Go," originally published in: Pattern-Directed Inference Systems (D. Waterman and F. Hayes-Roth, Eds.), pp. 503-523, 1978.

[6] W. Reitman and B. Wilcox, "The Structure and Performance of the INTERIM.2 Go Program," originally published in: Proceedings of the International Joint Conference on Artificial Intelligence, 1979, pp. 711-719.

[7] H. Remus, "Simulation of a Learning Machine for Playing Go," Originally published in: Proceedings of IFIP Congress, 1962, pp. 428-432.

[8] B. Mc Quade, "Machine Learning and the Game of Go," Master Thesis, Middlebury College, 2001.

[9] D. Stoutamire, "Machine Learning, Game Play and Go," Ph.D. Thesis, Case Western Reserve University, 1991.

[10] J.M. Burmeister, "Studies in Human and Computer Go: Assessing the Game of Go as a Research Domain for Cognitive Science," Ph.D. Thesis, The University of Queensland, Australia, 2000.

[11] N. Richards, D. Moriarty, P. Mc Questen and R. Miikkulainen, "Evolving Neural Networks to Play Go," Applied Intelligence, vol. 8, pp. 85-96, 1998.

[12] A. Lubberts and R. Miikkulainen, "Co-evolving a Go-Playing Neural Network," in Coevolution: Turning Adaptive Algorithms upon Themselves, Birds-of-a-Feather Workshop, Genetic and Evolutionary Computation Conference (Gecco-2001, San Francisco), 2001.

[13] D. Fotland. The 1999 FOST (Fusion of Science and Technology) Cup World Open Computer Go Championship, Tokyo, 1999. Available: http://www.britgo.org/results/computer/fost99.htm

[14] T.S. Anantharaman, M. Campbell, F.H. Hsu, "Singular Extensions: adding selectivity to brute force searching," Artificial Intelligence, vol. 43, no. 1, pp. 99-109, 1989.

[15] F. Hsu, "IBM's Deep Blue Chess Grandmaster Chips," IEEE Micro, (March-April): pp. 70-81, 1999.

[16] J. Schaeffer, R. Lake and P. Lu, "Chinook the World Man-Machine Checkers Champion," AI Magazine, 17 (1), pp. 21-30, 1996.

[17] J. Schaeffer, "One Jump Ahead - Challenging Human Supremacy in Checkers," Springer Verlag, 1997.

[18] Fogel D.B., 2002, "Blondie24: Playing at the Edge of AI". Morgan Kaufmann, SF, CA.

[19] K. Chellapilla, D.B. Fogel, "Evolving an Expert Checkers Playing Program Without Using Human Expertise," IEEE Transactions on Evolutionary Computation, vol. 5, no. 4, pp. 422-428, August 2001.

[20] K. Chellapilla and D.B. Fogel, "Co-Evolving Checkers Playing Programs using only Win, Lose or Draw," in AeroSense99, Symposium on Applications and Science of Computational Intelligence II, vol. 3722, 1999, pp. 303-312.

[21] K. Chellapilla and D.B. Fogel, "Evolving Neural networks to Play Checkers Without Relying on Expert Knowledge," IEEE Transactions on Neural Networks, vol. 10, no. 6, pp. 1382-1391, November 1999.

[22] K. Chellapilla and D.B. Fogel, "Anaconda Defeats Hoyle 6-0: A Case Study Competing an Evolved Checkers Program against Commercially Available Software," in Proceedings of Congress on Evolutionary Computation, La Jolla Marriot Hotel, La Jolla, California, USA, July 16-19 2000, pp. 857-863.

[23] B. Bouzy and T. Cazenave, "Computer Go: an AI Oriented Survey," Artificial Intelligence Journal, vol. 132, pp. 39-103, 2001.

[24] E. Thorp and W. Walden, "A Partial Analysis of Go," originally published in: The Computer Journal, vol. 7, no. 3, pp. 203-207, Heyden and Son Ltd., 1964. Reprinted

[25] E.O. Thorp and W.E. Walden, "A Computer-Assisted Study of Go on *mxn* Boards," originally published in: Information Sciences, vol. 4, pp. 1-33, America Elsevier Publishing Company, Inc., 1972.

[26] M. Müller, "Decomposition Search: A Combinatorial Games Approach to Game Tree Search, with Applications to Solving Go Endgames," in IJCAI-99, vol. 1, pp. 578-583, 1999.

[27] N. Richards, D.E. Moriarty and R. Miikulainen, "Evolving Neural Networks to Play Go," Applied Intelligence, Kluwer Academic Publishers, Boston, vol. 8, pp. 85-96, 1998.

[28] I. Ishigure, “In The Beginning: The Opening in the Game of Go,” The Ishi Press, Japan, 1973.

[29] J. Greenberg, “OpenGo,” hosted by Inventivity.com LLC’s Software Research, 1995-2001. Available: http://www.inventivity.com/OpenGo/

[30] D. Moriarty and R. Miikulainen, “Forming Neural Networks Through Efficient and Adaptive Co-Evolution,” Evolutionary Computation, vol. 5, no. 4, pp. 373-399, 1997.

[31] P. Donelly, P. Corr and D. Crookes, “Evolving Go Playing Strategy in Neural Networks,” AISB Workshop on Evolutionary Computing, Leeds, England, 1994.

[32] H.D. Enderton, “The Golem Go Program,” Technical Report CMU-CS-92-101, Carnegie Mellon University, 1991.

[33] M. Enzenberger, “The Integration of a Priori Knowledge into a Go Playing Neural Network,” 1996. Available bye internet: http://www.markus-enzenberger.de/neurogo.ps.gz

[34] N. Schraudolph, P. Dayan and T. Sejnowski, “Temporal Difference Learning pf Position Evaluation in the Game of Go,” Advances in Neural Information Processing 6, Morgan Kaufmann, 1994.

[35] S-J Yen, W-J Chen and S-C Hsu, “Design and Implementation of a Heuristic beginning Game System for Computer Go,” in Joint Conference on Information Sciences (JCIS) 1998, pp. 381-384. Association for Intelligent Machinery, 1998.

[36] M. Müller, “Computer Go,” Artificial Intelligence, Elsevier, vol. 134, pp. 145-179, 2002.

[37] M. Enzenberger, “Computer Go Bibliography,” Available in internet: http://www.markus-enzenberger.de/compgo_biblio/compgo_biblio.html

[38] B. Bouzy and T. Cazenave, “Computer Go: An AI Oriented Survey,” Artificial Intelligence, vol. 132, no. 1, pp. 39-103, 2001.

[39] J. Burmeister and J. Wiles, “AI Techniques Used in Computer Go,” Fourth Conference of the Australasian Cognitive Science Society, Newcastle, 1997.

[40] E. Lasker, “Go and Go-moku: The Oriental Board Games,” Dover Publications, Inc., New York, 1960.

[41] C. Cho, “Go: A Complete Introduction to the Game,” Kiseido Publishing Company, Korea, 1997.