Introducing a Round Robin Tournament into Blondie24

Belal Al-Khateeb and Graham Kendall, Member, IEEE

Abstract—Evolving self-learning players has attracted a lot of research attention in recent years. Fogel's Blondie24 represents one of the successes in this field and a strong motivating factor for other scientists. In this paper evolutionary neural networks, evolved via an evolution strategy, are utilised to evolve game playing strategies for the game of checkers by introducing a league structure into the learning phase of a system based on Blondie24. We believe that this helps eliminate some of the randomness in the evolution. Thirty feed forward neural network players are played against each other, using a round robin tournament structure, for 150 generations and the best player obtained is tested against a reimplementation of Blondie24. We also test the best player against an online program, as well as two other strong programs. The results obtained are promising.

I. INTRODUCTION

Designing automated computer game playing programs has been of interest since the 1950's [1,2], and is still of interest today, with successes such as Deep Blue in 1997 [3,4], which defeated Garry Kasparov, considered the best ever chess player. Game playing involves many important aspects of interest to artificial intelligence such as knowledge representation, search and machine learning. Traditional computer games programs use a knowledge based approach, where human knowledge about the game is encoded by hand into the computer by means of an evaluation function and a database of opening and end game sequences.

The experiments described here represent an extension to Fogel's work on checkers where we suggest that the introduction of a round robin league structure will produce a better player. The designed system show a better performance than the old one by playing many games between them and with two other computer programs.

The rest of the paper organised as follows: In section II related work is presented, section III discusses Blondie24. Our re-implementation of Blondie24 is presented in section IV, while section V shows our experimental setup for this work. Section VII presents our results. Section VII shows our conclusions along with suggested future research directions.

Belal Al-Khateeb and Graham Kendall are with The School of Computer Science, University of Nottingham, UK (T:+44 (0) 115 951 4251, F:+44 (0) 115 951 4254; email: gxk@ics.nott.ac.uk).

II. BACKGROUND

In 1954, Arthur Samuel started work on evolving a checkers player in an attempt to demonstrate that a computer program could improve by playing against itself. Samuel's program evolved weights for 39 features [2,5]. A reinforcement learning method [6-9] was used to evolve these features during the game instead of tuning them by hand. The most important feature that Samuel discovered was piece difference and the remaining 38 features (including capacity for advancement, control of the centre of the board, threat of fork, etc.) have various importance rankings. Due to the limitation of memory Samuel only used 16 features, out of the 38, in his evaluation function, swapping between features to include the remaining 22, which he called term replacement [10]. Two evaluation functions were used to determine the weights for the features. He called these Alpha and Beta. At the start, Alpha and Beta have the same weight for every feature. Alpha weights were modified during the run of the algorithm and Beta did not change. The process gave an appropriate weight to each parameter and summed them together. Each leaf node in the game tree was evaluated using this evaluation function. This process represents one of the first attempts to use heuristic search methods in searching for the best next move in a game tree. Samuel used minimax with three ply and a procedure called *rote learning* [2] was included in the program. This procedure is responsible for storing the evaluation of different board positions in a look-up table for fast retrieval (Look-Ahead and memorization technique). Samuel also incorporated alpha-beta pruning that also included a supervised learning technique to allow the program to learn how to select the best parameters to calculate the evaluation function [5].

One of the criticisms of traditional knowledge based approaches for developing game-playing machine intelligence is the large amount of pre-injected human expertise that is required for the computer program, together with the lack of learning capabilities of these programs [10,11]. The evaluation function and opening and end game databases are typically provided by program experts. In this sense, a computer game's *intelligence* is not gained by actually playing a game, but rather comes from the pre-designed evaluation function and a look up of database moves. Moreover, this intelligence is not adaptive. It could be argued that humans read books and watch other people playing a game before they actually start playing themselves. Humans also improve their skill through trial-and-error. New features and strategies for playing a game can be discovered by new players rather than grand masters, while old features could be viewed as worthless and old strategies are discarded. Humans also adapt their strategies when they meet different types of players, under different conditions, in order to accommodate their special characteristics. We do not see such adaptations and characteristics in the knowledge based computer game programs. Fogel commented on this phenomenon in computer game-playing [10]: "*… To date, artificial intelligence has focused mainly on creating machines that emulate us. We capture what we already know and inscribe that knowledge in a computer program. We program computers to do things – and they do those things, such as play chess, but they only do what they are programmed to do. They are inherently "brittle". … We'll need computer programs that can teach themselves how to solve problems, perhaps without our help. …*"

In 1998 Richards et al. produced a self-learning program [12] that was capable of playing the game of Go on small boards (9x9), without any injection of prior knowledge. This program used the SANE (Symbiotic Adaptive Neuro-Evolution) method [13,14] to evolve neural networks to be able to play Go on small boards. The design of the neural network consisted of a three (one input, one hidden and one output) layer feed-forward network with evolvable connection weights. The input units were used to indicate whether the black or white stones were present, while the output unit indicated whether a move is good or not (a positive value reflects a good move, while a negative or zero value indicates a bad move). The evaluation function of SANE used Chinese scoring by counting all the stones of the same colour, together with all locations completely surrounded by stones of that colour and the difference in the scores between SANE and its opponent is summed over N games and used as a fitness level for the networks. SANE was tested by playing against Wally (written by Bill Newman), Wally was chosed because it was strong enough to play with an unskilled network [14]. On a 5x5 board SANE needed only 20 generations to defeat Wally, while it needed 50 generations on a 7x7 board. On a 9x9 board, SANE needed 260 generations.

TD-Gammon [15,16] represents a first attempt to produce a self-learning computer program that is able to play a game of backgammon to the level that is competitive with human experts. TD-Gammon is a neural network based computer program that is able to teach itself how to play the game of backgammon by playing against itself starting from completely random initial play. TD-Gammon used a multilayer perceptron, which takes a sequence of board positions from the start, until the end (one side succeeds in removing all their pieces) and produces an output that represented the network's estimation as to how good is that board position. No features were encoded in the neural network during training and the network was used to select the best move for both sides (learning from the results of playing against itself). TD-Gammon contained 160 hidden nodes and performed a three-ply search. It was trained on over six million self play games [17,18]. TD-Gammon has been tested against many human players during its different versions, with different modifications, and was shown to be very successful. TD-Gammon was also shown to be able to play better against human experts than Neurogammon [16].

Blondie25 [19] was an attempt to produce a self-learning evolutionary chess program that can learn how to play the game of chess by playing against itself, injecting as little expert knowledge as possible. Blondie25's implementation worked as follows: The chessboard was represented as a vector of length 64, with each element representing an available board position. Components in the vector take values from {-K, -Q, -R, -B, -N, -P, 0, +P, +N, +B, +R, +Q, +K} where 0 represented an empty square and the variables P, N, B, R, Q, and K represented material values for pawns, knights, bishops, rooks, the queen and king, respectively. The sign of the value indicated whether or not the piece belonged to the player (positive) or to the opponent (negative). Three fully connected artificial feedforward neural networks were used, each one with 16 inputs, 10 hidden nodes, and a single output. The three neural networks focused on the first two rows, the back two rows and the centre of the chess board. To start the evolutionary process, 20 computer players were initialized with the values 1,3,3,5,9 and 10,000 for P, N, B, R, Q and K respectively [20]. Each player played 10 games (five as white and five as black) against 10 randomly selected players from the same population and according to their scores (+1 for win, 0 for draw and -1 for lose) the 10 players which scored more points were selected and the others were killed off. The selected players were mutated to produce 10 offspring. The best player from the last generation was selected to be Blondie25. Games were played using an alpha beta search with a depth of four ply. Blondie25 was tested against many popular chess programs [21] and showed success as a first attempt to produce an evolved chess player, which had limited expertise in the experimintal setup. Blondie25 was the first machine learning program to defeat a human master at chess. The performance rating for Blondie25 against Fritz8.0 (rated at 2752(±20) on SSDF[1]) is at about 2640.

[1] Swedish Chess Computer Association (Swedish: *Svenska schackdatorföreningen*, SSDF)

III. BLONDIE24

Blondie24 [10,22-26], represents an attempt to design a computer checkers program without injecting any human knowledge (except the basic rules). Evolutionary neural networks were used, which were very successful as a self-learning computer program. The evaluation function, for a given board position, was provided by the neural network that was being used for a particular colour for a given game. Initially, these networks acted randomly (as their weights were initialized randomly) but over time they gradually improved due to evolutionary pressure. The final network was able to beat the majority (>99%) of human players registered on www.zone.com at that time. Blondie24 does not play at the level of Chinook [27], but it does represent a significant contribution to the literature, particularly in machine learning and artificial intelligence. In comparison, the major difference between Blondie24 and other traditional game-playing programs is in the employment of the evaluation function [22,23]. In traditional game-playing programs, evaluation functions usually comprise important features derived from expert human techniques for generating good moves. Hand tuning is used to alter the weighting of these features. In Blondie24, the evaluation function is an artificial neural network that only knows the number of pieces on the board, the type of each piece and their positions: No other inputs, such as human experience about the techniques of the game, are pre-programmed into the neural network.

Blondie24 demonstrated that it is possible for a computer to learn rather than just being pre-programmed with human knowledge. The following algorithm is used in Blondie24 [24-26]:

1- Initialise a random population of 30 neural networks (strategies), P_i=1,…,30, sampled uniformly [-0.2,0.2] for the weights and biases.
2- Each strategy has an associated self-adaptive parameter vector, s_i=1,…,30 initialised to 0.05.
3- Each neural network plays against five other neural networks selected randomly from the population.
4- For each game, each competing player receives a score of +1 for a win, 0 for draw and -2 for a loss.
5- Games were played until either one side won, or until one hundred moves were made by both sides, in which case a draw was declared.
6- After completing all games, the 15 strategies that have the highest scores are selected as parents and retained for the next generation. Those parents are then mutated to create another 15 offspring using the following equations:

$$s_i(j) = s_i(j)exp(tN_j (0,1)), j = 1, ..., N_w$$
$$w_i(j) = w_i(j) + s_i(j)N_j(0,1), j = 1, ..., N_w$$

where N_w is the number of weights and biases in the neural network (here this is 5046), $t = \frac{1}{\sqrt{2\times\sqrt{N_w}}}$ = 0.0839, and $N_j(0,1)$ is a standard Gaussian random variable resembled for every *j*.
7- Repeat steps 3 to 6 for 840 generations (this number was an arbitrary choice by Fogel in his implementation of Blondie24).

Blondie24 represents a milestone in evolutionary learning but the construction of the evolution did not allow for the end product of the evolutionary process to learn as well (the learning was exercised only in the evolution phase and no learning took place in the playing phase). This makes Blondie24 incapable of adapting itself when interacting with that environment. After learning, Blondie24 is an *end product,* that is incapable of further improvement. Harley mentions this in his book review and comments [28]:
"… *An interesting point is that the end product which looks intelligent is Blondie, yet she is not in fact the intelligence. Like the individual wasp, Blondie is fixed in her responses. If she played a million games, she would not be iota smarter. In this sense, she is like Deep Blue. ... Perhaps a better example of intelligence would be ... a human, who can adapt her behavior to any number of new challenges…*"

To be more accurate, the creation of Blondie24 is to be considered as a learning process (achieving Samuel's challenge [10]) but Blondie24 itself is unable to learn from its environment [29]. One other thing that can be noticed from step 3 in the algorithm is that the strategies do not all play the same number of games because, by chance, some would be selected as opponents more often than others. Our research will investigate if this is a limiting factor in order to eliminate the randomness in choosing opponents.

IV. BLONDIE24 RE-IMPLEMENTATION

In order to investigate our proposed extensions to Fogel's work we firstly re-implemented Blondie24 (which we refer to as Blondie24-R) in order to provide a firm foundation for our research. Our re-implementation has the same structure and architecture that Fogel utilised in Blondie24, with the exception that the value of the King is fixed to 2.

Intuitively, the King is more valuable than an ordinary piece, and this is a well known, even to novice players. So putting the value of the King as two (or any other value that is greater than an ordinary piece value) will not be considered as knowledge injection to the program.

V. EXPERIMENTAL SETUP

In order to eliminate the randomness in the evolutionary phase of Blondie24-R and hence produce a better player, a league competition between all the 30 neural networks is suggested. A league competition could be operated by making all the neural networks play against each other. This means any selected network would play, as a red player, against the other 29 players instead of only playing against five randomly chosen players. The total number of matches per generation in this model will be 870 (30*29) rather than 150 (30*5), as in the original implementation of Blondie24. This increase in the number of matches will decrease the number of generations (145 verses 840) that can be played in a given time, in order to provide a meaningful comparison against the original work, as Blondie24-R has a total of 126,000 games (30*5*840) so Blondie24-RR needs 145 generations to play almost the same number of games (29*30*145=126,150).

Therefore, the only difference between this algorithm and the one above is in step 3, where every network competes against every other. We refer to this player as Blondie24-RR.

VI. RESULTS

To guage the effect of introducing a round robin tournament we play Blondie24-R against Blondie24-RR. Bearing in mind the fact that both players are end products, a win result for our modified player would be seen as a success. Also we play several matches against an online program in addition to playing several matches against two strong checkers programs. Table 1 shows the results obtained.

Table1: Results of Playing Against Selected Programs.

	Blondie 24-R	Blondie 24-RR	Online[1]	Win Check 3D[1]	SX checkers[1]
Blondie 24-R[1]	-	Draw	Win	Lose[1]	Lose[1]
Blondie 24-RR[5]	Win	-	Win	Lose[1]	Lose[1]

VII. CONCLUSIONS AND FUTURE WORK

Analyzing the results in table 1, Blondie24-RR (after 140 generations) plays two matches (one as red and one as white) against Blondie24-R. Blondie24-RR wins as red (starts first) against Blondie24-R, the result is draw when Blondie24-RR moves second. This clearly reflects a success for our hypothesis based on the fact that both players are *end products.* It should be noted that both players will always play with the same strategy due to their deterministic nature. For this reason it is only sensible to play a single game. It would be possible to play several matches between Blondie24-R and Blondie24-RR by starting with all possible first moves and playing out each scenario but we have not conducted this experiment in this work.

The other results obtained show that both Blondie24-R and Blondie24-RR win against an online program which can be considered as another success. We decided to play against two programs, which are considered strong players[3,4]. For the first one Blondie24-RR lost with a two piece difference, while Blondie24-R lost with a seven piece difference. Playing against the second program shows that Blondie24-RR lost with a four piece difference, while Blondie24-R lost with an eight piece difference. We played several matches with those programs[2,3,4] in order to investigate whether they are deterministic or not. The results were the same, indicating that the players always respond with the same moves. These results show that Blondie24-RR is performing better than Blondie24-R. Losing by four checkers is actually the same as losing by eight checkers (it iss still a loss), but in this experiment we want to compare the performance of Blondie24-RR with Blondie24-R and not with those computer programs, bearing in mind that all of them are end products.

Based on these results it would seem appropriate to use the league structure, instead of only choosing five random opponents to play against during the evolutionary phase.

Now that we have shown that enhancements are possible to the Blondie24 framework our future work will investigate if other changes are possible. For example, we will investigate using individual and social learning methods [29] in order to enhance the ability of Blondie24-RR to overcome the problem of being an *end product*.

VIII. REFERENCES

[1] Turing, A. M., Computing machinery and intelligence, Mind, Vol.59, 1950, 433-460.
[2] Samuel, A. L., Some studies in machine learning using the game of checkers, IBM Journal on Research and Development, 1959, 210-229. Reprinted in: E. A. Feigenbaum and J. Feldman, eds., Computer and Thought, NY: McGraw-Hill, 1963. Reprinted in: IBM Journal on Research and Development, 2000, 207-226.
[3] Newborn M., Kasparov vs. Deep Blue, Computer Chess Comes of Age. New York: Springer-Verlag, 1997.
[4] Campbell M., Hoane A.J., and Hsu F.H., "Deep Blue," Artificial Intelligence, Vol. 134, 2002, 57-83.
[5] Samuel, A. L., Some studies in machine learning using the game of checkers II – recent progress, IBM Journal on

Research and Development, 1967, 601-617. Reprinted in: D. L. Levy, ed., Computer games, NY: Springer-Verlag, 1988, 366-400.

[6] Kaelbling, L. P., Littman M. L. and Moore A.W., Reinforcement Learning: A Survey, Journal of Artificial Intelligence Research, Vol. 4, 1996, 237-285.

[7] Mitchell, Tom M., Machine learning, McGraw-Hill, 1997.

[8] Sutton, R. S. and Barto, A. G., Reinforcement learning, MA: MIT Press, 1998.

[9] Vrakas D., Vlahavas I. PL., Artificial intelligence for advanced problem solving techniques, Hershey, New York, 2008.

[10] Fogel D. B., Blondie24 Playing at the Edge of AI, United States of America: Academic Press, 2002.

[11] Fogel, D. B., Evolutionary Computation: Toward a new philosophy of machine intelligence (Second edition). NJ: IEEE Press, 2000.

[12] Richards N., Moriarty D. E. and Miikkulainen R., Evolving Neural Networks to Play Go, Applied Intelligence, Vol. 8, 1998, 85-96.

[13] Moriarty D. E. and Miikkulainen R., Forming neural networks through efficient and adaptive co-evolution, Evolutionary Computation 1998, Vol. 5, 373-399.

[14] Lubberts A. and Miikkulainen R., Co-Evolving a Go-Playing neural network, In Coevolution: Turning adaptive algorithms upon themselves, Birds-of-a-Feather Workshop, Genetic and Evolutionary Computation Conference 2001.

[15] Robertie B., Carbon versus silicon: Matching wits with TD-Gammon, Inside Backgammon, Vol. 2, 1992, 14–22.

[16] Tesauro G., Programming backgammon using self-teaching neural nets, Artificial Intelligence, Vol. 134, 2002, 181–199.

[17] Tesauro G., Practical issues in temporal difference learning, Machine Learning, Vol.8, 1992, 257–277.

[18] Tesauro G., Temporal difference learning and TD-Gammon, Comm. ACM, Vol. 38, 1995, 58–68.

[19] Fogel D. B., Hays T. J., Hahn S. L. and Quon J., A Self-Learning evolutionary chess program, in Proceeding of IEEE, IEEE Press, Vol. 92, 2004, 1947- 1954.

[20] Fogel D. B., Hays T. J., Hahn S. L. and Quon J., Further evolution of a self-learning chess program, Computational Intelligence and Games, IEEE Press, 2005, 73-77.

[21] Fogel D. B., Hays T. J., Hahn S. L., Quon J.: The Blondie25 Chess Program Competes Against Fritz 8.0 and a Human Chess Master. Computational Intelligence and Games 2006, IEEE Press, 2006, 230-235.

[22] Chellapilla K. and Fogel, D. B., Anaconda defeats hoyle 6-0: A case study competing an evolved checkers program against commercially available software. Congress on Evolutionary Computation, La Jolla Marriot Hotel, La Jolla, California, USA, 2000, 857-863.

[23] Fogel D. B. and Chellapilla K., Verifying anaconda's expert rating by competing against Chinook: experiments in co-evolving a neural checkers player. Neurocomputing, Vol. 42, 2002, 69-86.

[24] Chellapilla K. and Fogel D.B., Evolution, Neural Networks, Games, and Intelligence," Proceedings of the IEEE, Vol. 87, 1999, 1471-1496.

[25] Chellapilla K. and Fogel D. B., Evolving an expert checkers playing program without using human expertise, IEEE Transactions on Evolutionary Computation, Vol. 5, 2001, 422-428.

[26] Chellapilla K. and Fogel D. B., Evolving neural networks to play checkers without relying on expert knowledge. IEEE Transactions on Neural Networks, Vol. 10, 1999, 1382-1391.

[27] Schaeffer, J., One jump ahead: challenging human supremacy in checkers. New York: Springer-Verlag, 1997.

[28] Harley, E., Book Review: Blondie 24, playing at the edge of AI, The IEEE Computational Intelligence Bulletin, 2002, 25-27.

[29] Kendall G. and Su Y., Imperfect Evolutionary Systems, IEEE Transactions on Evolutionary Computation, 2007, Vol. 11, 294-307.