Using Coevolution and Gradient-based learning for the Virus Game

Munir H Naveed
Fatima Jinnah Women University
Department of Computer Science, ITC Building, The Mall, Rawalpindi, Pakistan.
+92 (051) 9271167
M.H.Naveed@bradford.ac.uk

Peter I Cowling
University of Bradford
MOSAIC Research Centre, Horton Building, Department of Computing, Bradford, UK.
+44 (01274) 234005
P.I.Cowling@bradford.ac.uk

ABSTRACT
This paper presents a novel coevolutionary model which is used to create strong game (The Virus Game) playing strategies. We use two approaches to coevolve Artificial Neural Networks (ANN) which evaluate board positions of a two player zero-sum game (The Virus Game). The first approach uses the coevolution with initial population of random ANN and second approach is a novel coevolutionary model with initial population of ANN which are trained using gradient based adaptive learning methods (Backpropagation, RPROP and iRPROP). In our case, the results of coevolutionary experiments show that pre training of the population in coevolution is highly effective in creating stronger game playing strategies than coevolution with random population.

Categories and Subject Descriptors
I.1.2 [**Artificial Intelligence**]: Learning, Connectionism and neural nets, Knowledge acquisition.

I.1.5[**Pattern Recognition**]: Models, Neural nets.

General Terms
Algorithms, Performance, Design, and Experimentation.

Keywords
Coevolution, Gradient-based Learning, Artificial Neural Networks, Virus Game.

1. INTRODUCTION
In biology, coevolution is the mutual evolutionary influence between two species that become dependent on each other. Each species in a coevolutionary relationship exerts selective pressures on the other species. Coevolution occurs if the traits of one species *A* have evolved due to the presence of a second species *B* and vice versa. This natural phenomenon has motivated AI researchers to apply coevolution in solving different types of problems where two or more entities are interacting with each other. Coevolution is an unsupervised learning method that requires only relative measurement of phenotype performance, well-suited to the game-playing domain.

The gradient-based learning methods: Backpropagation [11], Resilient backpropagation (RPROP) [10] and improved resilient backpropagation (iRPROP) [7] are supervised learning methods. They use a delta rule and can be applied to the problem of learning neural network weights to give a network which produces the desired outputs with minimised error.

Games continue to be important domains for investigating problem solving techniques [9]. Games offer tremendous complexity in a computer manageable form and need sophisticated AI methods to play at expert level. Board games like Chess [3], checkers [6], Othello [15] and backgammon [14] have been used to explore new ideas in AI. We will survey this work late in this section. In this paper, we used the "Virus Game" as a testbed to explore coevolutionary ideas.

The "Virus Game" [4][5] is a two-person perfect information board game of skill. The game is played on a square board. The player who always starts the game is the *Black Player* and the other player is the *White Player*.

In the Virus Game, there are two kinds of moves available for each turn. The first kind of move is *grow move* or *one step move*. In this kind of move, a player moves a piece of his colour to an empty position adjacent to its current position. The positions are adjacent if their borders or corners are adjacent. The result of this move reproduces the moving piece and occupies both positions, the new position and the old position. The second kind of move is *jump move* or *two step-move* where a player moves a piece to an empty position which is two squares away from its current position via an empty square. The piece leaves the old position empty and occupies the new position. In either case, all opposing pieces adjacent to the new player's piece change colour. Players alternate, moving only one piece per turn. The game ends when neither player can move. The player with the greatest number of pieces is the winner. The game is declared a draw if both players have the same number of pieces at game end.

Competitive learning was initially explored by Samuel [13] to adjust the parameters of a deterministic evaluation function in a checkers playing computer program. Tesauro [14] used the temporal-difference learning approach to evolve a backgammon-playing neural network. Tesauro's TD-Gammon yields a computer playing backgammon program of world-champion strength. Coevolutionary competitive learning is explored for the Repeated Prisoner's Dilemma (RPD) by Axelrod [2] and Miller [8]. Axelrod evolve RPD playing strategies using a fixed environment (i.e. using eight fixed opponents) while Miller coevolved the RPD strategies by playing each strategy against every other strategy and itself in a population. According to the results shown by Miller, the best evolved RPD playing strategies

in his work performed well against strong strategies (like Tit-for-Tat) taken from Axelrod's work. Axelrod and Miller used Genetic Algorithms to evolve RPD playing strategies. Angeline and Pollack [1] used competitive coevolution with Tic Tac Toe as a testbed. They introduced a tournament competitive fitness function. Smith and Gray [12] introduced a competitive function where there are n/2 competitions per generation for a population of size n. Smith and Gray applied coevolution to Othello. The weights of a deterministic evaluation function are evolved using a co-adapted GA with explicit fitness sharing. The coevolved evaluation function may not be very strong but the approach is notable for the formation of stable niches (i.e. stable groups) during the evolutionary process. The individuals of each group in a generation have similar characteristics and the results show that the individuals in a group continuously evolve during the coevolutionary process.

Anaconda [6] is a checkers playing neural network which is evolved using competitive coevolution.The authors have used Evolutionary Programming to coevolve the neural networks in a competitive environment and the strongest neural network, Anaconda, is rated at expert level according to a tournament conducted at website www.zone.com.

This paper investigates the effectiveness of two coevolutionary approaches. In the first approach, initial populations, of varying sizes, containing random neural networks, are evolved against 10 strong hand-crafted AI players. The weights of neural networks are evolved using a Genetic Algorithm. In the second coevolutionary approach, a large number of neural networks are trained using gradient based learning methods under the supervision of 10 fixed hand-crafted AI players. The trained neural networks are then coevolved against the same and different fixed opponents. Thus we are able to investigate whether pre-training of the population is effective.

2. EXPERIMENTAL DESIGN

2.1 Neural Network Design

The architecture of our neural networks is represented by I-H_1-H_2-O where 'I' represents number of input units, 'H_1' means the number of hidden neurons in first hidden layer, 'H_2' represents total number of hidden neurons in second hidden layer and 'O' represents number of output neurons. The architecture 64-58-27-1 (smallest among last three architectures) is used in all experiments.

In the selected neural networks architecture, there are 64 input units where each input unit is associated with a square of board. A location-based input encoding scheme [14] is used where each input unit is associated with a square of board. If the corresponding square of an input unit has a black piece on the board then its value is assigned as '1', if square is occupied with white piece the input value for this unit is '-1' and value '0' is used for empty square.

All hidden neurons have sigmoid [11] activation function. All weights are encoded by real numbers in the range [-0.5, 0.5]. The output neuron has linear activation function and gives a real number as the output of the neural network.

2.2 Evolutionary Model

A Genetic Algorithm (GA) is used to evolve the connection weights of a population of neural networks. A chromosome contains all connection weights of one neural network. The fitness of a chromosome is the total score of its corresponding neural network in playing two games, one as a black and other as white, with each of 10 fixed AI opponents. If a neural network wins a game against an opponent, its score is increased by 3; if the match is draw then score is increased by 1 and for loss there is no change in the score. Each neural network plays twenty games per generation and its total score after twenty games is used to measure its fitness. All games are run to 1-ply only to keep search times manageable.

The Selection operator applies a rank selection method [17]. At the end of the tournament for a given generation, all chromosomes are arranged in a descending order according to fitness score. From the sorted list of N chromosomes, the top N/3 chromosomes are selected for sexual reproduction. These selected chromosomes are paired to reproduce 2N/3 new chromosomes through a two-point crossover operation. The process of two-point crossover is used to reproduce two new chromosomes and Gaussian Mutation Operator [16] is applied to each new offspring.

2.3 Gradient-based Learning

Neural networks are trained under the supervision of 10 hand-crafted AI players using gradient-based techniques: Backpropagation (BP), RPROP and iRPROP. In this case, 10 different training sets are created where each training sets contains a large number of board positions and their evaluation values as determined by a hand-crafted AI player. Therefore, each neural network is trained to learn the evaluation function of a given hand-crafted AI player. Each neural network is trained using one training set. Four neural networks are trained using iRPROP, three are trained using RPROP and three are trained using BP, for each hand-crafted AI player. BP uses a learning rate of 0.01. The learning parameters for RPROP and iRPROP are set to η^+=1.2, η^-=0.50, Δ_o=0.5, Δ_{min}=0 and Δ_{max}=50.

2.4 Coevolutionary Model

Our coevolutionary model uses two different approaches for generating the starting population. In the first approach, an initial population of neural networks with random weights is coevolved against ten fixed opponents. The coevolution of neural networks continues until at least one neural network in a given population beats all of the fixed opponents or there is no improvement in the scores of the best neural network for 10 consecutive generations. All games in this coevolutionary approach are played at 1-ply by both neural networks and fixed opponents.

In the second approach an initial population of trained neural networks is used. These neural networks are selected from the pool of 100 pre-trained neural networks. If the population size is less than or equal to 10, all neural networks used are based on different hand-crafted AI players. If population size is greater than 10, these are selected randomly in such a way as to ensure that a neural network trained by each of the 10 hand-crafted AI player is in the starting population . Therefore two populations of size 20 would have 10 identical and 10 different neural networks in the initial population. The trained neural networks are

coevolved against 10 fixed opponents. The stopping condition and ply depth is same as used in first approach.

To assess the generality of approach, we also test against another 10 hand-crafted AI players which are not used in pre-training or coevolution.

3. RESULTS

The results obtained from the randomly initialised neural networks are summarized in table 2 while the performance of the best of these coevolved neural networks against 10 hand-crafted AI players is shown in table 3. All experiments are run on Pentium IV 1.2 GHz using C#.Net running under Windows XP.

Table 1 shows the results of coevolution with populations starting from randomly created neural networks. The column *"Mean of Generation"* represents average number of generations over 10 runs and "mean CPU time" represents usage of average CPU time in seconds by a population during coevolutionary process over 10 runs.

We can see that small populations give much worse results than large ones. The difference between the performance of small and large populations is probably due to the number of parallel directions for exploring potential solutions in the search space. The experimental results also demonstrate that small populations use less CPU time than large populations, essentially since less neural network evolution are required. The average maximum scores with 0% and 20% crossover rates demonstrate that neural networks evolved using crossover generally have higher scores than those which do not. The crossover operation appears to help the evolutionary process to explore a more interesting region of the search space. The crossover operator speeds convergence, and the population converges to better solution than without crossover. The mean of min and mean values show that diverse populations are maintained.

Table 1. Results from of coevolution with an initial population of randomly created neural networks.

Pop Size	Cross over Rate	Mean of Max	Mean of Min	Mean of Mean	Mean of Gen	Mean CPU Time
6	0%	3	0	2	11	9
6	20%	18	0	9	10	13
15	0%	9	0	1	10	28
15	20%	6	0	1	11	40
20	0%	12	0	5	11	53
20	20%	30	0	3	6	57
30	0%	30	0	16	11	61
30	20%	60	0	34	6	72
50	0%	21	0	22	11	132
50	20%	60	0	35	5	72

The results of the tournament played between the best evolved neural network from each population where coevolution was started from an initial population of randomly generated neural networks and ten hand-crafted AI players. All the best neural networks with 0% crossover rate have won games as black against all AI opponents but none is able to win when playing as white. The best-evolved neural networks of all populations (except population of size 15) with 20% crossover rate won all games when playing as black and white. Table 1 and these results provide clear evidence for the effectiveness of the crossover operator, and support the advantage of Black and White in the Virus game, at least for approaches using 1-ply search.

Table 2 shows the mean of maximum, mean, and minimum scores of coevolved neural networks with a starting population of pre-trained neural networks. The table also shows the mean number of generations and CPU time (in seconds) for each population with different crossover rates. Again we see that small populations perform more poorly than larger populations. If we compare the results of tables 1 and 2, it can be construed that starting from a population of pre trained neural networks gives better and more consistent performance (in terms of playing strength) than starting from a population of random neural networks. Large sized populations (started with initial population of pre trained neural networks) use a smaller number of generations but still require more CPU time during the coevolution of neural networks than small size populations. The values of mean of minimum and maximum scores show the diversity is maintained in the population. Note that CPU times in this case include the time need for learning of the initial population

Table 2: Results from of coevolution with an initial population of pre-trained neural networks.

Pop Size	Cross over Rate	Mean of Max	Mean of Min	Mean of Mean	Mean Of Gene	Mean CPU Time
6	0%	40	0	28	13	12
6	20%	60	0	31	5	7
15	0%	60	0	22	3	18
15	20%	60	0	28	1	18
20	0%	60	0	26	3	31
20	20%	60	0	31	1	25
30	0%	60	0	29	2	66
30	20%	60	0	36	1	55
50	0%	60	0	32	1	84
50	20%	60	0	34	1	84

Generally, we see from table 2 that coevolution with crossover needs fewer generations and less CPU time than without crossover

The experimental results show that the populations of large size (starting from the initial population of pre trained neural networks) generally have at least one pre trained neural network which beats all the opponents before the start of coevolution when playing as black and white.

Table 3. Summary of results from 10 runs of coevolution of pre-trained neural networks using 20 opponents.

Pop Size	Cross over Rate	Mean of Max	Mean of Min	Mean of Mean	Mean of Gen	Mean CPU Time
6	0%	80	0	66	12	102
6	20%	120	60	73	4	96
15	0%	120	0	45	2	48
15	20%	120	0	52	1	48
20	0%	120	0	45	2	48
20	20%	120	0	55	1	78
30	0%	120	0	59	1	102
30	20%	120	0	70	1	132
50	0%	120	0	87	1	168
50	20%	120	0	95	1	210

In order to further investigate the generality of populations starting from an initial population of pre trained neural networks, we introduced 10 more hand-crafted AI opponents which were not used in the training of the initial population. Table 3 gives the average results of coevolution with different populations starting from initial population of trained neural networks where neural networks play as black and white against 20 AI opponents in each generation which include the 10 previously unseen opponents. In this case, pre trained neural networks in larger populations were able to beat not only AI players used for initialisation but also the AI players which were unseen by them during the supervised training of initial population of pre trained neural networks. The results shown in tables 2 and 3 are quite similar. The results of these both tables with populations of small sizes show that coevolution with 20% crossover rate has better performance than coevolution with 0% crossover rate. The coevolution of populations of large size has same performance with both crossover rates (essentially since the initial populations were very strong).

According to results at the start of coevolutionary process, in the populations of large size, there is at least one trained neural network, which beats all 10 opponents when playing as black and white against them.

Tables 4 and 5 present the results for a tournament between the best player from each set of experiments starting with random initial weights, $R_{6,0}$, $R_{6,20}$, ..., $R_{50,20}$, and the best player from each set of experiments starting with pre-trained networks, $P_{6,0}$, $P_{6,20}$, ..., $P_{50,20}$. Each of these players plays 19 games as black against each other player, and 19 games as white against each other player. Since *every* one of $P_{6,0}$, $P_{6,20}$, ..., $P_{50,20}$ is ranked higher than every one of $R_{6,0}$, $R_{6,20}$, ..., $R_{50,20}$ this provides very clear evidence for the superiority of pre-training, particularly since the training times for pre-trained networks were significantly smaller than for the random initial population. Within $R_{6,0}$, $R_{6,20}$, ..., $R_{50,20}$ and within $P_{6,0}$, $P_{6,20}$, ..., $P_{50,20}$ we see significant variations in performance. It is clear that there are still significant variations in the phenotype among these best players.

Table 4: Summary of Tournament Results for coevolved neural networks from initial populations of randomly created neural networks.

Players No	No of games Won as Black Player	No of games Won as White Player	Total Score	Rank (1-20)
$R_{6,0}$	4	4	26	18
$R_{6,20}$	1	7	32	16
$R_{15,0}$	4	5	28	17
$R_{15,20}$	5	2	22	19
$R_{20,0}$	5	5	32	15
$R_{20,20}$	9	6	47	12
$R_{30,0}$	9	9	54	11
$R_{30,20}$	2	4	20	20
$R_{50,0}$	7	7	42	13
$R_{50,20}$	7	6	39	14

Table 5: Summary of Tournament results for coevolved neural networks from populations starting from pre trained neural networks.

Players No	No of games Won as Black Player	No of games Won as White Player	Total Score	Rank (1-20)
$P_{6,0}$	10	10	63	10
$P_{6,20}$	14	13	83	5
$P_{15,0}$	16	15	93	1
$P_{15,20}$	16	15	93	2
$P_{20,0}$	16	15	93	3
$P_{20,20}$	16	7	70	8
$P_{30,0}$	16	7	70	9
$P_{30,20}$	13	13	84	4
$P_{50,0}$	16	8	73	6
$P_{50,20}$	16	8	73	7

4. CONCLUSION

A coevolutionary model is presented where artificial neural networks adapt and learn to play the Virus Game using competitions with fixed strong hand-crafted (deterministic) AI players. The neural networks are provided with only raw board positions. The results presented in this paper demonstrate the potential advantages of a combination of coevolution and supervised learning techniques for building knowledge into artificial neural networks. The combination of coevolution and gradient-based learning techniques gives improved playing performance and faster learning when compared to either approach combined in isolation.

It would be interesting to explore the combination of gradient-based learning techniques and coevolution for other games and in the case where a deeper search is used.

The Genetic Operators of Crossover and Mutation are also analysed and we have noted that an evolutionary algorithm with crossover has much better performance than one with mutation alone in most experiments. In future we aim to investigate the dynamics which make crossover an effective operator.

5. REFERENCES

[1] P.J. Angelin and J.B. Pollack, "Competitive Environments Evolve Better Solutions for Complex Tasks", in the proceedings of 5th International Conference on Genetic Algorithms (GAs-93), 1993, pp. 264-270.

[2] R. Axelrod, "The Evolution of Strategies in the Iterated Prisoner's Dilemma", Genetic Algorithms and Simulated Annealing, in Lawrence Davis (ed.), Morgan Kaufmann, 1997, pp. 32-41.

[3] M. Campbell, Jr. A.J. Haone, F-h. Hsu, "Deep Blue", Artificial Intelligence, Vol.134, 2002, pp.57-83.

[4] P.I. Cowling, R. Fennell, R. Hogg, G. King, P. Rhodes, N. Sephton, "Using Bugs and Viruses to Teach Artificial Intelligence", in the proceedings of 5th Game-on International Conference on Computer Games: Artificial Intelligence, Design and Education, 2004, pp. 360-364.

[5] P.I., Cowling, "Board Evaluation for the Virus Game", in the proceeding of IEEE 2005 Symposium on computational Intelligence and Games (CIG'05), Graham Kendall and Simon Lucas (editors), 2005, pp. 59-65.

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

[7] C., Igel and M. Husken, "Empirical Evaluation of the Improved RPROP Learning Algorithms", Neurocomputing, Vol. 50C, 2003, pp.105-123.

[8] J.H. Miller, "The Coevolution of automata in the repeated prisoner's dilemma", Journal of Economics Behavior and Organization, Vol.29, 1996, pp.87-112.

[9] D.E., Moriarty and R. Miikkulainen, "Discovering Complex Othello Strategies Through Evolutionary Neural Networks", Connection Science, Vol.7 No.3, 1995, pp. 195-209

[10] M. Riedmiller and B. Heinrich, "A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm", in the proceedings of IEEE International conference on Neural Networks, 1993, pp.586-591.

[11] D.E. Rumelhart, J.L. McMlelland and the PDP Research Group, "Parallel Distributed Processing", Exploration in the Microstructure of Cognition, Vol. 1, 1986, MIT Press

[12] R.E. Smith and B. Gray, "Co-Adaptive Genetic Algorithms: An Example in Othello Strategy", in the proceeding of The Florida Artificial Intelligence Research Symposium, 1994.

[13] A.L. Samuel, "Some Studies in Machine Learning using the Game of checkers", IBM Research and Development Journal, 1959, pp. 211-229.

[14] G.J. Tesauro, "Temporal Difference Learning and TD-Gammon", Communications of the ACM, Vol. 38, No. 3, 1995, pp. 56-68.

[15] M. Buro, "The Othello Match of the Year: Takeshi Murakami vs. Logistello", ICCA Journal, Vol. 20, No.3, 1997, pp.189-193.

[16] X. Yao and Y. Liu, "Fast Evolutionary Programming", in the proceedings of 5th annaual conference on Evolutionary programming, 1996, pp. 451-460.

[17] M. Mitchell, "An Introduction to Genetic Algorithms", MIT Press