Coevolving Stacking-Based Game Playing Agent Using Particle Swarm Optimisation Algorithms

BS Steyn
(Department of Computer Science)
Stellenbosch University
Stellenbosch, South Africa
21740178@sun.ac.za

***Abstract*—Coevolutionary techniques in combination with particle swarm optimisation and neural networks have been shown to be very successful in finding strong game-playing agents. These have been particularly useful in deterministic games. This paper investigates the effectiveness of this approach when the extra complexity of stacking is involved in the game. The game used to test is The Last Captain. The performance is measured against random playing agents. The performance is also tested against a static evaluation function. The particle swarm and neural network are optimised for The Last Captain.**

I. Introduction

Artificial intelligence and game optimisation have long been involved together. [6] This is due to the fact that games provide challenging "puzzles" to be solved. Game environments are an ideal domain to investigate AI techniques and approaches, as games have set rules that must be adhered to, allowing them to be structured like problems. Even though deterministic games can be solved, the time required can be enormous, hence why approaches have been developed to estimate a good move without fully knowing the impact. This paper focuses on using a neural network tuned using a PSO to try to estimate whether a move is good. For a benchmark, a static evaluation is used.

II. Background

A. Particle swarm optimisation algorithm

Since the initial contributions of Kennedy and Eberhart, various PSO techniques have been developed. This paper makes use of the inertia weight model of PSO, developed by Shi and Eberhart [4]. The inertia weight model of PSO establishes a proportional relationship between a particle's velocity at the previous iteration and the particle's velocity at the current iteration. According to the inertia weight model of PSO, the equations to update the velocity and position of a particle are given by

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \mathbf{v}_i^{t+1} \tag{1}$$

$$\mathbf{v}_i^{t+1} = w\mathbf{v}_i^t + c_1\mathbf{r}_{1i}(\mathbf{y}_i^t - \mathbf{x}_i^t) + c_2\mathbf{r}_{2i}(\hat{\mathbf{y}}_i^t - \mathbf{x}_i^t) \tag{2}$$

Bold symbols in the equations above represent n-dimensional vectors that correspond to the dimensionality of the search domain. The symbols of equations (1), (2) and are clarified below:

- $\mathbf{x}_i^t$ represent the the position of i^th particle at iteration t. The initial positions of the particles are randomly allocated within the search space.
- $\mathbf{x}_i^{t+1}$ represent the the position of i^{th} particle at iteration $t+1$.
- $\mathbf{v}_i^t$ represents the velocity of the i^{th} particle at iteration t. The initial velocity of all particles in the swarm is initialised to the zero vector.
- $\mathbf{v}_i^{t+1}$ represents the velocity of the i^{th} particle at iteration $t+1$.
- $\mathbf{y}_i^t$ represents the position of the best function evaluation found by the i^{th} particle, defined as the personal best position. This position is updated if the i^{th} particle personally finds a position with a better function evaluation.
- $\hat{\mathbf{y}}_i^t$ represents the position of the best function evaluation found by the neighbourhood the i^{th} particle belongs to, defined as the neighbourhood best position. This position is updated if any particle belonging to the neighbourhood finds a better function evaluation.
- w represents the inertia coefficient. w determines the effect the previous velocity of the i^{th} particle has on the subsequent iteration.
- c_1 represents the cognitive coefficients. c_1 determines the effect the personal best position has on the iteration.
- c_2 represents the social coefficients. c_2 determines the effect the neighbourhood best position has on the iteration.
- $\mathbf{r}_{2i}$ and $\mathbf{r}_{1i}$ introduce stochasticity into the model. $\mathbf{r}_{2i}$ and $\mathbf{r}_{1i}$ are sampled from a uniform distribution over (0,1).

The algorithm used in this paper is clearly illustrated in Algorithm 1.

B. Control parameter configurations

A PSO particle is considered stable if it has control parameters w, c_1, c_2 that satisfy the following equation. This has been both theoretically and empirically proven that the particle will reach an equilibrium state while adhering to

$$c_1 + c_2 < \frac{24(1-w^2)}{7-5w} \text{ for } w \in [-1, 1] \tag{3}$$

The control parameters used in this paper will adhere to (1) by using the weights 0.7, 1.4, 1.4.

Algorithm 1 Standard PSO Algorithm

Create and initialize an n_x-dimensional swarm, S;
repeat
  for each particle i=1,...,$S.n_s$ **do**
    if $f(S.x_i) < f(S.y_i)$ **then**
      $S.y_i = S.x_i$;
    end if
    if $f(S.y_i) < f(S.\hat{y}_i)$ **then**
      $S.\hat{y}_i = S.y_i$;
    end if
  end for
  for each particle i=1,...,$S.n_s$ **do**
    update the velocity;
    update the position;
  end for
until stopping condition is true

C. Coevolution

Coevolution is a competitive process between several species that enables the species to evolve to outperform the other species. An excellent example of coevolution is given in [2]. The example provided in the paper above is an example of predator-prey coevolution, where a positive outcome for one species means a negative outcome for the species. One of the other variations is symbiotic evolution. In this variation, the species do not compete but rather cooperate for the general good of all species. The model used in this paper is closer to a predator-prey model but is also different as a positive outcome for one species does not mean a negative outcome for the other species.

III. The Last Captain

The Last Captain is a deterministic 2 player game developed by João Pedro Neto in 2002. The game is played on an 8x8 grid. Each player starts with 4 Captains and 8 ships. The game starts with the following setup:

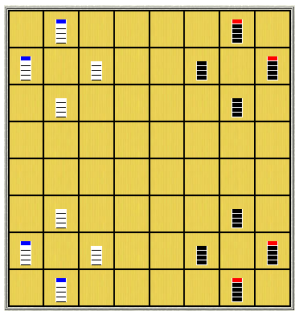

Fig. 1: Initial board for The Last Captain

In Figure 1, ships are represented as stacks of the same colour. White captains are defined by the blue piece, while the red pieces define black captains. On a player's turn, they can either move a captain and n pieces below him to another ship of the same colour without a captain or move the captain and all pieces under him to an empty square. Ships without captains can also be moved, although they cannot merge with other ships like captains. Ships can move in a straight line(orthogonal or diagonal). The maximum distance a ship can move is defined by the number of pieces that make up the ship. For example, a ship of size $n = 2$ can move a maximum of 2 squares. If a ship has a captain, the maximum range is increased by 1. Ships cannot pass over ships of the same colour. If a ship passes over an opposing ship, they remove pieces from the opposing ship equal to the size of their ship. A ship is sunk if it has no pieces left. A captain can be sunk if they have no pieces left in the ship besides themselves. Ships can pass over any number of opposing ships as long as the destination square is the square after the last opposing ship passed over. Ships remove pieces from all opposing ships passed over. The winner of the game is the last player to have a captain left. The white player plays first. Table 1 shows the probabilities of a win, a loss, and a draw for a game of The Last Captain. These probabilities were calculated using two random playing agents. These agents played against each other in 10000 games.

	Games	%
White Player	4558	45.58
Black Player	4832	48.32
Draw	610	6.1

TABLE I: The Last Captain probabilities

The table shows that there is a slight advantage to playing black. This advantage is due to the fact that the black player could have the opportunity to capture a piece on their first move, while the white player will never have this opportunity.

IV. The Game Playing Agents

The game-playing agents are represented by standard 3-layer feed-forward neural networks, consisting of summation units. The size of the input layer is the size of the board. So for The Last Captain, 64 neurons are required. The hidden layer's size is 33, determined by taking the mean of the input and output layer sizes. The size of the output layer is 1. The sigmoid activation function is used for each neuron in the hidden and output layers. The weights and biases are initialised between [-1] and [1]. The neural network is used to evaluate a given board state. The current board state is provided as an input, and an output value representing the favorability of that state is returned. The higher the return value, the more favourable the board state is. Assume the white player needs to choose their next move, then the following steps are followed.

1) Build a game tree with a depth of N, using the current game state as the root node and by adding all the possible moves for white for all odd depths and all possible moves for black for all even nodes.
2) Evaluate all leaf nodes using the neural network as an evaluation function as described below:

a) For all black captains, assign a value of 200 plus the size of the size they are on. For all white captains, assign a value of 100 plus the size of the ship they are on. For all black ships without a captain, assign a value of 50 plus the size of the ship. For all white ships without a captain, assign the size of the ship. For all empty square,s assign a value of 0.
b) Supply these values as the input values as inputs to the neural network and perform a feed-forward process. To determine the output.
c) Assign the output value as the score for that given leaf node.

3) Using the minimax[6] algorithm, determine the most beneficial state to execute the next move.

To avoid increased complexity only a single depth will be considered for the rest of this paper. Since a neural network is used to evaluate how good a state is, the objective is to find a set of weights which can determine a good and bad board state. The weights and biases are adjusted using coevolution and particle swarm optimisation.

V. The Game Training Process

This section explains how the coevolution and particle swarm optimisation algorithms are combined to train game-playing agents.

A. Step-by-Step

1) Create two new swarms that are made up of multiple neural network particles. Each neural network is initialised as described in Section 4. The best personal weights for each neural network are set to the current network for the first generation.
2) The two swarms compete against each other. The weights compete in the following way:
3) Every neural network in swarm 1 individually competes against all neural networks in swarm 2. The select challenger from swarm 1 plays both white and black. Based on each win, loss or draw, a score is assigned to the neural network.
4) The current weights of each neural network are then compared to its personal best weight of that network. If the current score is better, the best weights are replaced by the current weights.
5) The best weights of each neural network within the swarm are then compared, and the best weights are then set as the global best for that swarm. This is done within both swarms.
6) The global best from each swarm then plays 200 games against a random agent, wherein the win rate of the global best is recorded. The global best with the highest win rate between the two swarms then becomes the supreme best if its win rate is higher than the current supreme best.
7) The weights within each swarm are then updated using particle swarm optimisation.
8) If the algorithm hasn't converged, the process is repeated from step 2.

VI. Results

This section presents the results found in this paper. Each simulation was executed 10 times, with the best weights and scores stored for each.

A. Static Evaluation Function Results

The static evaluation used is

$$(\sum_{n=1}^{nsP1} SV_n + \sum_{n=1}^{ncP1} 10) - (\sum_{n=1}^{nsP2} SV_n + \sum_{n=1}^{ncP2} 10) \quad (4)$$

where $nsP1$ is the number of ships the current player has, $nsP2$ is the number of ships the opposing captain has, $ncP1$ is the number of captains the current player has, $ncP2$ is the number of ships the opposing player has, and SV_n is the value of the n^{th} ship of the player. Table 2 shows the results of how the static evaluation function performs.

	Games	%
Static Player	9943	99.43
Random Player	13	0.13
Draw	44	0.44

TABLE II: Static Evaluation Function

The results indicate that the static evaluation function performs well. For the static evaluation function to achieve the perfect game, the depth of the search tree should be expanded.

B. Coevonlutionary PSO Results

A global best topology was used for the PSO. No velocity clamping was used. Table 3 shows the results of the supreme player produced using the Coevolutionary configuration for this paper.

	Games	%
Supreme Player	92	46
Random Player	4	2
Draw	104	52

TABLE III: Static Evaluation Function

The high draw rate is due to the constraint of 100 moves for the game to be played. This was assigned as the neural networks in the early generation have a tendency to enter infinitely repeating sequences of moving back and forth. The constraint was added with the hope that it would encourage the neural networks that do not enter this state to receive higher scores and pull the rest of the swarm away from these infinite loop weights. This does seem to be successful, as the win rate does increase over the generations. This can be seen in Figure 2 below:

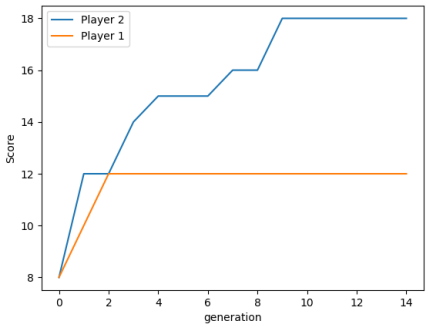

Fig. 2: Performance for best agent

Figure 2 shows how there is an improvement over the generations, but due to limited generations, the swarm can get stuck on a specific set of weights. This is likely due to this solution being a local maximum for the scores without a better solution nearby. This allows the weights of swarm 2 to keep improving while swarm 1 stagnates. Different hidden layer sizes were tested, and it was observed that changing from the current layer setup discussed earlier did not result in better results. Increasing the swarm sizes did lead to better results but this also increased the runtime substantially; hence, a feasible size of 4 was chosen for each swarm.

VII. Conclusion

The coevolutionary technique in combination with a particle swarm optimisation algorithm has shown promising results. The limits of this paper do not stem from the techniques used but rather from the hardware available. Ideally, the generations should be increased to around 5000. Larger swarm sizes would also be beneficial to increase the quality of the game agents produced. Although this paper also reveals the benefits of a well-designed static evaluation function with the same hardware limitations, at only depth 1 produced a nearly perfect game against the random player with an average amount of moves of around 35.

References

[1] A. Brudno. *Bounds and valuations for abridging the search for estimates. Problem of cybernetics*. Translation in Problemy Kibernetiki, 10:141-150., 1963.

[2] Cecilia Chio, Paolo Chio, and Mario Giacobini. “An Evolutionary Game-Theoretical Approach to Particle Swarm Optimisation”. In: vol. 4974. Mar. 2008, pp. 575–584. ISBN: 978-3-540-78760-0. DOI: 10.1007/978-3-540-78761-7_63.

[3] Johan Conradie and Andries Engelbrecht. “Training Bao Game-Playing Agents using Coevolutionary Particle Swarm Optimization”. In: May 2006, pp. 67–74. DOI: 10.1109/CIG.2006.311683.

[4] Nelis Franken and Andries Engelbrecht. “Evolving intelligent game-playing agents.” In: *South African Computer Journal* 32 (Jan. 2004), pp. 44–52.

[5] Anshul Gopal, Mohammad Sultani, and Jagdish Bansal. “On Stability Analysis of Particle Swarm Optimization Algorithm”. In: *Arabian Journal for Science and Engineering* 45 (July 2019). DOI: 10.1007/s13369-019-03991-8.

[6] L. Messerschmidt and A.P. Engelbrecht. “Learning to play games using a PSO-based competitive learning approach”. In: *IEEE Transactions on Evolutionary Computation* 8.3 (2004), pp. 280–288. DOI: 10.1109/TEVC.2004.826070.

[7] Evangelos Papacostantis, Andries Engelbrecht, and Nelis Franken. “Coevolving Probabilistic Game Playing Agents using Particle Swarm Optimization Algorithm.” In: Jan. 2005.

[8] J. Schaeffer. “The games computers (and people) play”. In: *Academic Pres* (2001).