Engineering

Industrial & Management Engineering fields

Okayama University *Year* 2002

Evolutionary constitution of game player agents

Hisashi Handa* Tadashi Horiuchi†

*Okayama University

†Matsue National College of Technology

sice02-0663

Evolutionary Constitution of Game Player Agents

H. Handa[1] and T. Horiuchi[2]

1 Okayama University, Tsushima-naka 3-1-1, Okayama 700-8530, JAPAN
2 Matsue National College of Technology, Nishi-ikuma 14-4, Matsue 690-8518, JAPAN
handa@sdc.it.okayama-u.ac.jp horiuchi@it.matsue-ct.ac.jp

Abstract: In this paper, we propose a constitution method of game player agent that adopts a neural network as a state evaluation function for the game player, and evolves its weights and structure by Evolutionary Strategy (ES). In this method, we attempt to acquire better state evaluation function by evolving weights and structure simultaneously.

Keywords: Evolutionary Computations, State Evaluation Function, Neural Networks

1. Introduction

The constitution of game player agents for Complete-information games such like Chess, Reversi and so on have been studied extensively[1)]. In the case of the complete-information games, the game tree, whose nodes and edges indicate situations and moves for each parent node, respectively, is adopted. To play game, player agents, including human players, must estimate the value of each situation (node) and decide a move (edge) for leading their wins. That is, the study with respect to the construction of game player agents can be classified into two categories: the game tree search and the state evaluation function constitution (state-value function approximation). In primitive AI researches, there are plenty of the study of the tree-search algorithms based upon α-β method, min-max method, A* Algorithms, and so on. Such studies assume that a proper state evaluation function is given by using some heuristics of expert players in advance. In general, we must assign or approximate a considerable amount of state values in order to constitute the state value function. Therefore it is quite difficult to constitute the adequate state value function by hand. Occasionally, furthermore, some popular heuristics that everyone believes it is effective misleads to constitute the adequate state value function.

In this paper, we tackle to constitute effective state value functions by using Evolutionary Strategy. We adopt a neural network as the function approximator of the state value function of a game player agent. We moreover use the Evolutionary Strategy to search effective weight vector of the neural network, that is, to make the game player agent more strong. In order to realize strong game playing agents, it is necessary to select relevant features that capture important information about the current state of the game and to evaluate the state using the selected features. Hence, the proposed method incorporates a simple pruning mutation operator in order to change the structure of the neural network.

Now, we compare the proposed method with some conventional approaches: Conventional game-playing programs often use the linear combination of several features (linear evaluation function) that characterize the current game situation. Another approach to construct the evaluation function is to the use of neural network (NN) which serves as an state evaluation function. That is, the inputs are values of the features and output is an evaluation value which describe the quality of the current state of the game. But it is impossible to use the well-known Back propagation (BP) method for training the neural network, because it is difficult to provide the exact value (teacher signal) of the evaluation function. In such a case, reinforcement learning can be regarded as one of the possible methods. In game-playing, however, one episode (length from start to end of game) is quite long and there are many branches in a game tree. Therefore, it is difficult to exactly propagate backward the reward which is given only at the end of game (credit assignment problem) and to realize the efficient learning for game-playing agent.

In this paper, we focus to the use of neural network populations such that each network represents a state evaluation function and evolves the weights and structures by Evolutionary Strategy (ES). In the proposed method, we attempt to acquire a better state evaluation function by evolving weights and structures simultaneously. The next section introduces the proposed method in detail. Section 3 explains the effectiveness of the pruning mutation operator by showing the simulation results of Tic-Tac-Toe and 6x6 Reversi. Section 4 concludes this paper.

2. Proposed Method

2.1 Representation of Evaluation Function by NN

State evaluation in game-playing programs means to give a score for the current game situation. That is, it is necessary to select relevant features that capture important information about the current state of the game and to give a score for the state using the selected features. Most of conventional game-playing programs use the linear or non-linear combination of the features in order to get the evaluation value for the game state. In this paper, we adopt the way to represent the state

evaluation function by using a neural network, whose inputs are values of the features and output is an evaluation value for the game state.

For example, the input features in the experiment for playing Reversi are as follows:

Local piece configuration on the board: In this case, the board is segmented to several partitions. Each input value is calculated in accordance with a ratio of pieces to opponents' pieces in each partition [4].

Differential in the number of pieces: The purpose of Reversi is to acquire more pieces at the end of games. Hence, this input feature is considered as one of most important ones.

Phase of game: In general, strategy to generate moves should be changed during a play of game: the opening moves, the middle game, and the end game. This attribute simply presents such phases of game.

Number of possible moves: This shows how many kinds of possible moves for me/opponent can be achieved in current state. It is well known that the more kinds of possible moves you have, the more chance of beating the opponent increases you have.

In this paper, we use the above neural network to decide the next move. More concretely, we compute the all evaluation values of the possible (legal) moves from the current state, and we choose the move which has best evaluation value among them (greedy method).

2.2 Evolution of Evaluation Function by ES

Coding Method In order to evolve the state evaluation function (neural network) by evolutionary computation approach, it is necessary to represent the neural network as an individual. In this paper, we adopt the way to assign the element of weight vector in a neural network to each locus of the individual as depicted in Fig. 1. This coding method enables us to easily reconstruct a neural network which represents a state evaluation function by decoding an individual.

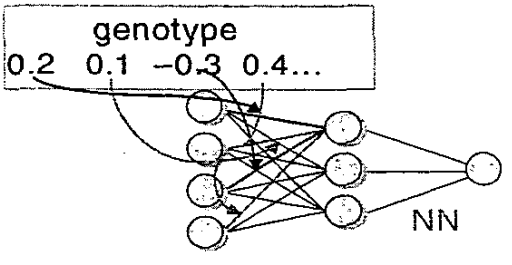

Figure 1: The Coding Method of the Proposed Method

Evolutionary Strategy with Pruning Mutation We propose to evolve a population of neural networks, where each network represents a state evaluation function, and evolve the weights and structures by Evolutionary Strategy (ES) [5]. In other words, we aim to generate individuals (neural networks) with high fitness that represent good evaluation functions.

Details of the procedure (1+λ)-ES are described in the following:

Step 1. Create one parent randomly.

Step 2. Create their descendants by adding perturbation depending on random numbers using normal (Gaussian) distribution and also by applying the *pruning mutation* which changes the value of weight selected randomly to zero by force. The number of descendants is λ.

Step 3. Decode each genotype to phenotype (neural network) and derive fitness values based on the result of game playing among the individuals.

Step 4. Select the best individual among (1+λ) individuals as a parent for the next generation.

Step 5. Go back to **Step 2.**, if terminal conditions have not satisfied yet.

One of the original points in the proposed method is the introduction of *pruning mutation*. Mutation based on random numbers using normal distribution only changes the value of a certain weight in neural network. However, pruning mutation which changes the value of weight selected randomly to 0 means the cut of link between two neurons. That is, it can change the structure of neural network. By using both mutations (standard mutation and pruning mutation), we can evolve the weights and structure of neural network simultaneously.

Fitness Evaluation by Co-evolution We adopt the *coevolutionary* framework between two populations; one is the population of the individuals which learn only first (initiative) moves and another is the population of the individuals which learn only second (following) moves. The fitness value of each individual is calculated as the sum of the score derived from the results of one match again the parent in another population and nine matches again the standard program as shown in Fig. 2. Here, we used a program easily to get through Internet as the standard program. The score of one match is set up in the following: one can get higher (positive) score if he/she wins with bigger differential in the number of pieces, and one can get lower (negative) score if he/she looses with bigger differential in the number of pieces.

Each individual is estimated its fitness value by accumulating the results of one play against the elitist individual in the opponents' population and nine games against the standard program, where each result is weighted by logistic function taking into account the

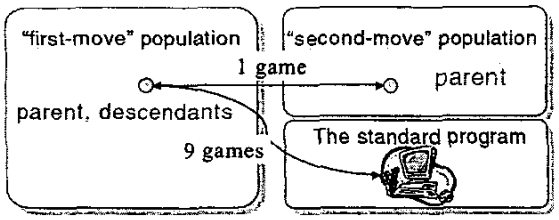

Figure 2: Game Plays for a Fitness Evaluation

differential of the number of pieces at the end game. That is, the fitness F_i of individual i is defined as

$$F_i = f(d_e) + \sum_{j}^{9}(d(j)),$$

where $f(\cdot)$ is the logistic function, d_e and $d(j)$ denote the differential of the number of pieces at the end of game against elitist individual in the opponents' population and jth play with the standard program.

3. Experiments and Results

Tic-Tac-Toe First, we examine the proposed method on Tic-Tac-Toe. We employ following parameters predecided by trial-and-error for experiments: the probabilities of normal (Gaussian) mutation and the pruning mutation proposed in this paper are set to be 0.2 and 0.005. The 1/5-rule is adopted to control the magnitude of the variance of the normal mutation. The initial variance for that mutation is set to be 0.05. The number of decedents λ is set to be 9. As the standard program for Tic-Tac-Toe, we adopt well-known algorithm for Tic-Tac-Toe such that it does never lost completely.

Figure 3 shows the temporal aspects of the number of draws against the standard program. The solid lines and dashed lines in these graphs denotes how many times the elitist, respectively, in ES with the prune mutation, and in ES without the prune mutation, draw with the standard program for 50 games. Note that games with the standard program in order to depict these graphs don't affect the evolutionary process in experiments, and these graphs are averaged results over 10 runs. As depicted in these graphs, the proposed method, i.e., "with pruning mutation", shows higher draw-rate.

Figure 3 shows the changes the number of eliminated weights by pruning mutation. The solid line and the dash line in this graph indicate the number of eliminated weights, from the input layer to the hidden layer and from the hidden layer and the output layer, respectively. Furthermore, an evolved neural network is shown in Fig. 5. Note that mean-less weights between the input layer and the hidden layer, i.e., weights whose subsequent weight between the hidden layer and the output layer is eliminated, are omitted to depict in this figure. In the case of Tic-Tac-Toe, the input attributes

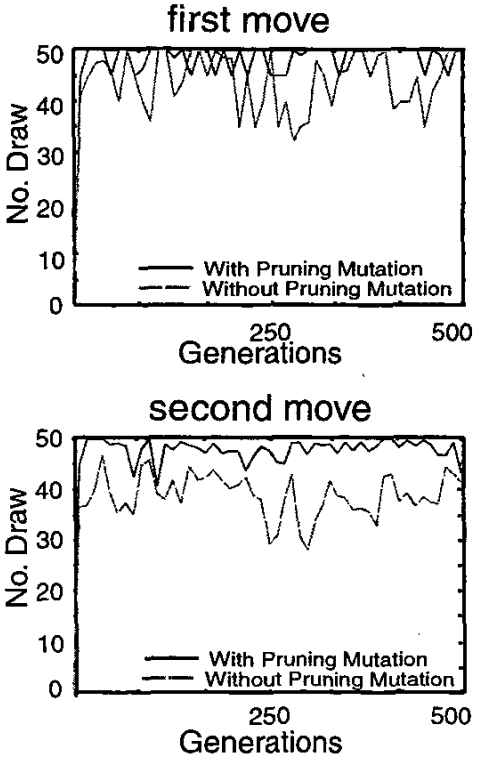

Figure 3: Experimental Results on Tic-Tac-Toe

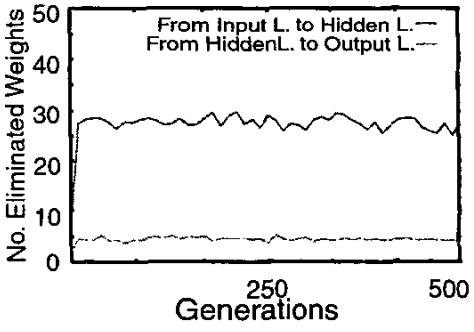

Figure 4: The Number of Eliminated Weights

consist of 7 attributes such as Att.0: state of marginals, Att.1: state of corners, Att.2: state of a center, and so on. As depicted in this figure, the evolved neural network ignores the attribute 0 for making his decisions of next-moves.

6 × 6 Reversi This subsection examines the proposed method on 6 × 6 Reversi such that the size of the board is set to be 6 × 6. The reason why we do not examine conventional 8 × 8 Reversi is that 8 × 8 Reversi causes extensive computational time. We adopt the same set-

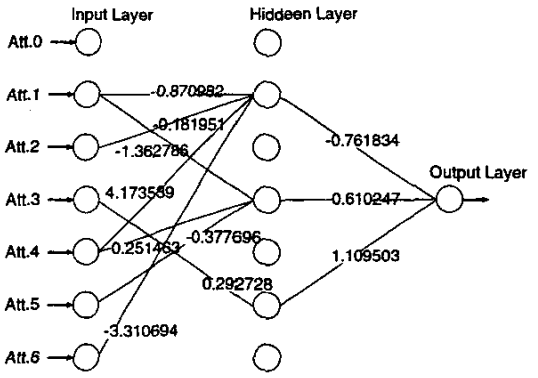

Figure 5: An Example of Evolved Networks for Tic-Tac-Toe

tings of the parameters we mentioned in previous subsection exception that the number of decedents λ is set to be 19.

Figure 6 shows experimental results for first move population (LEFT) and second move population (RIGHT) on 6×6 Reversi. As the same as Fig. 3, the solid lines and dashed lines in these graphs denotes how many times the elitist beats with the standard program for 100 games. This examination process is separately carried out with the evolutionary process, too. As delineated in these graphs, stronger players are evolved by ES. The pruning mutation proposed in this paper works well in both cases: first move and second move.

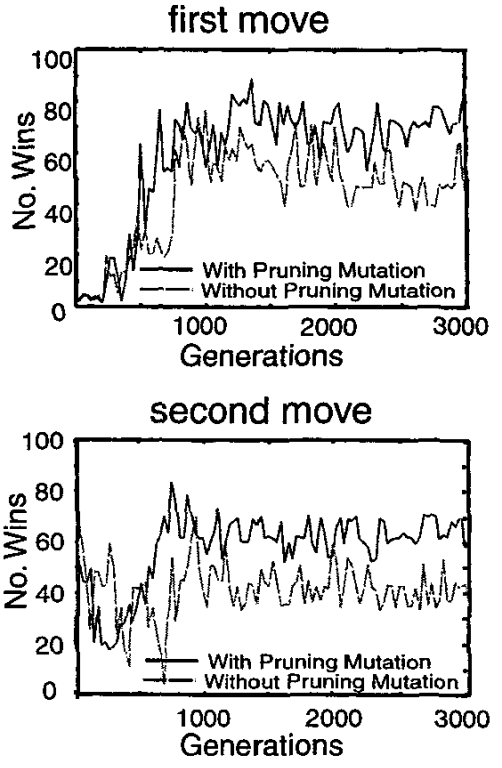

Figure 6: Experimental Results on 6×6 Reversi

4. Conclusion

In this paper, we proposed a method to represent the state evaluation function by using a neural network, whose inputs are values of probable features and output is an evaluation value for the game state, and to evolve simultaneously its weights and structure by Evolutionary Strategy in order to acquire better state evaluation function. Throught the applications to Reversi, we showed the effectivenes of the proposed method.

As future work, we will improve the proposed method by incorporating the mechanism of not only the autogeneration of attributes and the connection pruning between neurons but also the evolutionary acquisition of the configuration of Neural Networks.

Acknowledgement

Authors would like to thank to Mr. Shigeoka in Okayama University for his kindful support.

References

[1] D. Ishikawa, Current Statements of Computer Reversi, (in japanese) bit Vol. 32, No. 5 pp. 59–64, 2000.

[2] D. E. Goldberg, Genetic Algorithm in Search, Optimization, and Machine Learning, Addison-Wesley, 1989.

[3] G. Tesauro, Temporal Difference Learning and TD-Gammon, Communications of the ACM, Vol. 38, No. 3, pp. 58-68, 1995.

[4] K. Chellapilla and D. B. Fogel, Anaconda Defeats Hoyle 6-0: A Case Study Competing an Evolved Checkers Program against Commercially Available Software, Proc. of the 2000 CEC, pp. 857–863, 2000.

[5] T. Bäck, Evolutionary Algorithms in Theory and Practice, Oxford Press, 1996.