변형된 유전 알고리즘을 이용한 축구하는 다개체 시스템의 진화

Evolving Soccer-Playing Multi-Agents Using a Modified Genetic Algorithm

Il-Kwon Jeong* and Ju-Jang Lee
Department of Electrical Engineering
Korea Advanced Institute of Science and Technology
373–1 Kusong-dong Yusong-gu Taejon 305–701 Korea
Fax : 82-42-869-3410 E-mail : jik@odyssey.kaist.ac.kr

Abstract

It is difficult to design controllers for multi-agent systems without a comprehensive knowledge about the system. One of the ways to overcome this limitation is to implement an evolutionary approach to design the controllers. This paper introduces the use of genetic algorithms to discover rules that govern emergent co-operative behavior. A modified genetic algorithm was applied to automating the discovery of rules for multi-agents playing a simplified soccer game. A model consisting of movable agents in a cellular space is introduced. Simulation results indicate that, given the complexity of the problem, an evolutionary approach to find the appropriate rules seems to be promising. The implications of the results are discussed.

1 INTRODUCTION

Studying computational models of co-operative structures accomplishing a given task is an interesting area in the field of artificial life. However, generally it is difficult to design such models by analysis. As the problem size grows, it becomes more difficult. In the field of self-learning reactive systems it is not even desirable to have a clear idea of a computational model. Autonomous agents being adaptable implies an minimally pre-programmed systems. The aim of this paper is making agents learn to accomplish tasks by interacting with the environment and adapt future behavior on the basis of feedback from the present (or past) action[1].

Patel and Maniezzo solved a soccer-playing agent problem using neural networks to control agents, and genetic algorithms (GAs) to train the neural networks[1]. However, the result was not good enough to validate their approach to the problem, and only one team with two players was simulated. Recently, it has been shown that GAs are not suitable for training neural networks, and most GA-based learning algorithms for neural networks use GAs and another calculus-based algorithm (e.g. GA + BackPropagation)[2][3] or modified GA specially designed for neural network training[4].

Genetic algorithms are search methods based on natural selection and genetics. GAs are used in various problems including control problems. In control area, the GA has been used in identification, adaptation and neural network control[4][5][6][7]. The GA is different from conventional optimization methods in several ways. The GA is a parallel and global search technique that searches multiple points so it is more likely to obtain the global solution. It makes no assumption about the search space, so it is simple and can be applied to various problems[8]. It has been empirically proved that GAs are especially suitable for solving combinatorial optimization problems.

Lohn and Reggia used GAs to discover automata rules that govern self-replicating processes[9]. They developed a modified cellular automata (CA) model, named effector automata (EA). In the EA model, a cellular space is defined where individuals receive input from their local neighborhood, and using a predefined rule, produce an output. In EA models, each cell is a location in space, and automata are entities that can occupy cells.

In this paper, we use a modified genetic algorithm (MGA) to discover rules that govern emergent co-operative behavior. The genetic algorithm is applied to automating the discovery of rules for multi-agents playing soccer game. A model consisting of movable agents in a cellular space is introduced. The paper is organized as follows. In section 2, a simplified soccer model is described. In section 3, rule discovery process using MGA is described. Simulation results are provided in section 4.

2 SIMPLIFIED SOCCER MODEL

A simplified soccer (SS) model is similar to the effector automata (EA) model[9]. In both EA and SS models, a cellular space is defined where individual processing units (automata or agents), operating in

Table 1. Actions used for the SS model.

action	description
MOVE [DIR]	move one cell in the specified direction.
DRIBBLE [DIR]	move one cell with the ball in the specified direction only when currently possessing the ball.
KICK [DIR]	kick the ball two cells in the specified direction only when currently possessing the ball.

parallel, receive input from their local neighborhood, and produce an output using a pre-defined rule. Each cell is a location in space, and agents (automata) are entities that can occupy cells. In the SS model, the output is an action to effect, such as moving to a neighboring cell.

Time is discretized in the SS model, and space is an isotropic two-dimensional rectilinear grid composed of w (width) $\times$ l (length) cells. A cell may be empty or occupied by agents or ball, whereas a cell in a EA model can be occupied by only one automaton. Each agent is represented by a symbol T_i, $i = 1, 2, \cdots, n$, indicating the ith agent of agent-type $T \in \{A, B\}$, where n is the number of agents (players) of an agent-type (team). Agents with the same agent-type use identical rules.

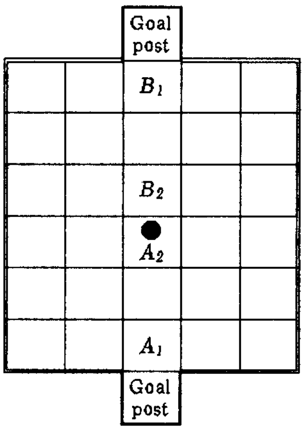

Figure 1. A playground of the SS model. • represents the ball.

Fig. 1 shows a playground of the SS model with 4 agents, A_1, A_2, B_1, and B_2, and a ball. It is assumed that each agent can move, or detect other objects (agents, goal posts and a ball) in one of the following directions: top, left, right, and bottom. The orientation of an agent is fixed during all the game. Fig. 2 shows a view window of an agent. An agent can detect objects in view window except for the ball, which is detected regardless of the ball position. An agent can detect other agents in a neighboring cell, and agents of the same agent-type also can be detected in a cell next to the neighboring one. The ball is detected according to the sector it belongs to (see Fig. 2).

The behavior of each agent is governed by a rule table. An entry of a rule table is a condition-action rule of the form:

$$\mathrm{C}(\cdots)\mathrm{L}(\cdots)\mathrm{R}(\cdots)\mathrm{T}(\cdots)\mathrm{B}(\cdots) \rightarrow \textit{action} \qquad (1)$$

where CLRTB stands for center, left, right, top, and bottom. For example the rule,

$$\mathrm{C}(A)\mathrm{L}(A)\mathrm{R}(\bullet)\mathrm{T}(\mathrm{ball})\mathrm{B}(A, B) \rightarrow \text{MOVE RIGHT}$$

means that if agents of agent-type A exist at the left and the bottom of A, agent B is at the bottom of A, the ball is at the top of A, and no others (• means empty), then the agent A at the center moves to its right at the next time step. The actions possible for the SS model are listed in Table 1. There are three actions; MOVE, DRIBBLE, and KICK. DRIBBLE and KICK are possible only when the agent going to do the actions has the ball. Values for the direction parameter (shown as [DIR] in Table 1.) are either left, right, top, or bottom. MOVE and DRIBBLE actions move an agent (and the ball in the case of DRIBBLE) to a neighboring cell, and KICK moves only the ball by two cells.

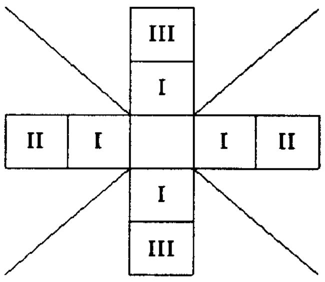

Figure 2. A view window of an agent. The center of the view window corresponds to the center of the agent. The agent can detect an agent of the same agent-type in the region I, II, and III, and can detect an agent of different agent-type in the region I. Goal posts in the region III can be detected. The ball is always detected; the position of the ball is either center, upper, lower, left, or right, according to the sector it belongs to.

We need a simulator for the SS model. The simulator simulates the movements of agents and the ball according to the rule table for a given time steps, and scores a game. The simulation stops when a team scores a goal or the given time steps is over. When an action by an agent is outside the ground the simulator disables the action. Because actions modify neighboring cells, a collision policy should be specified to address the possibility of two or more agents attempting to occupy the same cell. Two example policies are mutual annihilation which results in all agents being disabled to move, and the random winner policy which randomly selects one agent to occupy the cell in question[9]. In contrast to those policies, our SS model allows two or more agents into the same cell. The ball in a cell is regarded as belongings of the agent chosen probabilistically in the following manner. If there are n agents and the ball in a cell, then one of the agents obtain the ball with the probability $\frac{1}{n}$.

3 RULE DISCOVERY USING A MODIFIED GA

3.1 Problem Description

Our objective is to investigate how relatively simple agents can adaptively learn to solve a complex problem. Each agent should learn simple behaviors which are collectively sufficient to solve the problem. Agents have to decompose the problem effectively but this decomposition should be an emergent property of adaptive learning and not pre-programmed. It is an important motivation of this work that a problem should be solved with the minimal possible direction from the programmer or the trainer. We apply a modified genetic algorithm to finding a rule table for agents.

The experimental problem is a simplified soccer game. The players (agents controlled by rule tables) have to learn to play the game. Two players with the same agent-type make up a team in our simulation. It is assumed that the behavior of the players of one agent-type is pre-defined, because it is difficult to design an appropriate fitness function capable of evaluating the behavior of two teams simultaneously. The task of a team is to score a goal in a limited period while at the same time not to lose control of the ball to the opposing team. Success of a team depends on players learning to co-operate in order to score a goal.

Each player behaves independently of the other, and only knows about the existence of other players in the view window and the ball as described in the previous section. So there is no direct communication between players, and their knowledge of the aims of other players is also indirect. Hence each player interacts with a highly dynamic environment. Learning (modifying the rule table) takes place through feedback gained from actions in the environment.

In this paper, the simplified soccer model consists of 5×6 cells, that is, $w = 5$ and $l = 6$. The goal posts are located at the center of the top and the bottom of the ground, and each players and the ball are located as shown in Fig. 1 at the beginning of a soccer game. There are two agent-types; A and B. The behavior of the agent-type B is predefined and that of the agent-type A is to be learned. We have simulated the following three situations.

Case 1: agent B_1 and B_2 is fixed during the game, that is, they just act like obstacles.

Case 2: B_1 in front of the goal post is fixed. It may be thought as a goalkeeper. B_2 moves to capture the ball. So, it interferes in other player's DRIBBLE and KICK actions.

Case 3: The ball is initially at the left end of the ground. Other conditions are the same as in the case 2.

3.2 Methodology

We used a modified genetic algorithm (MGA)[4]. For a description of the algorithm readers are referred to the reference. The MGA is described briefly here for convenience. The MGA consists of a fitness modification and a modified mutation probability. The fitness value for a certain string is determined by the following rule.

$$fitness' = \begin{cases} k \times fitness_{avg}, \\ \text{if } fitness \geq k \times fitness_{avg} \\ fitness, \text{ other case} \end{cases} \tag{2}$$

where $fitness$ is the original fitness value and $fitness'$ is the modified value. $fitness_{avg}$ is the average of fitness values and k is a constant greater than 1. The modified mutation probability, p_m is given as

$$p_m(i_{gen}+1) = \begin{cases} p_{m0}, \text{ if the fittest is the same} \\ \text{ for } N_{reset} \text{ generations} \\ p_{m\_low}, \text{ if } p_m(i_{gen}) \times k_1 \leq p_{m\_low} \\ p_m(i_{gen}) \times k_1, \text{other case} \end{cases} \tag{3}$$

where i_{gen} is the generation number. p_{m0}, p_{m_low}, and k_1 is a positive constant less than 1. N_{reset} is a positive integer constant.

Some aspects to be considered to use the modified genetic algorithm are as follows:

• *Chromosome representation and Population size*: A rule table of condition-action rules is indexed implicitly by the neighborhood pattern in the view window. In the view window, there are $11 = {}_4C_0 + {}_4C_1 + {}_4C_2$, $5 = {}_4C_0 + {}_4C_1$, 2, and 2 possible configurations of agents of agent-type B, an agent A, the goal post of B, and the goal post of A, respectively. There are 5 (4 sectors and the center) configurations of the ball. The action part of a rule requires three bits; two bits for direction and one bit for an action type; MOVE or KICK. When an agent has the ball, MOVE is automatically recognized as DRIBBLE. Thus, a rule table encoded in binary string requires $3300 = 11 \times 5 \times 2 \times 2 \times 5 \times 3$ bits. A population consists of 50 chromosomes in our simulation.

• *Fitness*: The fitness function is defines as

$$F = \sum_{i=1}^{i=n_{iter}} F'(i) \tag{4}$$

$$F'(i) = F_0 + \sum_{t=0}^{t_{final}} (f_{goal-A}(t) + f_{possess-A}(t) + f_{goal-B}(t) + f_{possess-B}(t)) \tag{5}$$

where i represents the ith simulated game, n_{iter} is a given iteration number of simulations, t_{final} is the time when a simulation stops. When no goal occurs t_{final} is set to 30. In our simulation, $n_{iter} = 5$ and $F_0 = 400$. If the team A scores a goal at time t, then

$f_{goal-A}(t) = 1000$. $f_{goal-B}(t) = -90$, if the team B scores a goal at time t. In other cases, $f_{goal-A}(t) = f_{goal-B}(t) = 0$. If the team A possesses the ball at time t, then $f_{possess-A}(t) = 10$. $f_{possess-B}(t) = -10$, if the team B possesses the ball at time t. In other cases, $f_{possess-A}(t) = f_{possess-B}(t) = 0$.

- *Reproduction*: A chromosome is reproduced using the standard roulette wheel selection method. We used the elitist strategy, that is, the best chromosome is always reproduced without any alterations.
- *Crossover*: we used one point crossover. From experimentation, we found that a crossover probability of 0.8 yielded best results.
- *Mutation*: we used the modified mutation probability. The MGA parameters are as following: $p_{m0} = 0.5$, $p_{m\_low} = 0.03$, $N_{reset} = 5$, $k = 2.5$, and $k_1 = 0.9$.

4 SIMULATION RESULTS AND DISCUSSION

Fig. 3, Fig. 4, and Fig. 5 show the results for the case 1, case 2, and case 3 respectively. Each graph illustrates the maximum fitness value at each generation of the team A. Though we have used the elitist strategy the maximum fitness value is fluctuating. This is due to our probabilistic ball possessing policy. So, in absolute terms the fitness values indicate little of importance. The trend is far more revealing. In the case of the present experimental task the increase over generations indicates that the team is learning more and more appropriate behavior. The players have displayed co-operative behavior such as passing (using KICK action) the ball to their team-mates in order to make it easier to score a goal.

In the case 1, the MGA found a solution (rule-table capable of making a players in the team A score a goal) after about 30 generations. In the case 2, the MGA found a solution after 10 generation, though a player in the team B interferes with the players of the team A. However, due to the interference the maximum fitness value is more oscillatory than that of the case 1. The case 3 is the most difficult problem to solve; the MGA found a solution after about 120 generations. The players moved to the ball first and scored a goal using dribbles and passes. Usually, scoring a goal occurs after 20 time steps in the case 3, while it occurs after about 10 generations in the case 1 and case 2. Due to the increased interacting period with other players the maximum fitness curve in the case 3 is the most oscillatory among the three cases.

5 CONCLUSION

We have implemented an evolutionary approach using a genetic algorithm to design controllers for multi-agent system playing soccer game. For that purpose, a simplified soccer model consisting of movable objects in a cellular space was introduced. A modified genetic algorithm was applied to automating the discovery of rules for multi-agents playing soccer. Though we did not use comprehensive knowledge about the system the genetic algorithm successfully discovered rules that govern emergent co-operative behavior. Simulation results indicate that, an evolutionary approach to find the appropriate rules for emergent co-operative behavior seems to be promising.

Applying the proposed approach to a real multi-agent system consisting of wheeled mobile robots remains for further study.

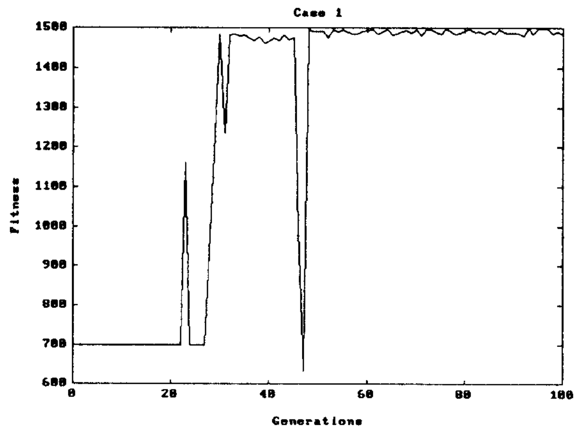

Figure 3. The result of the case 1.

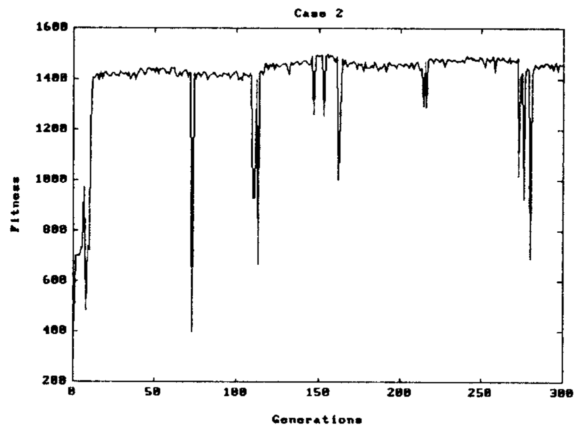

Figure 4. The result of the case 2.

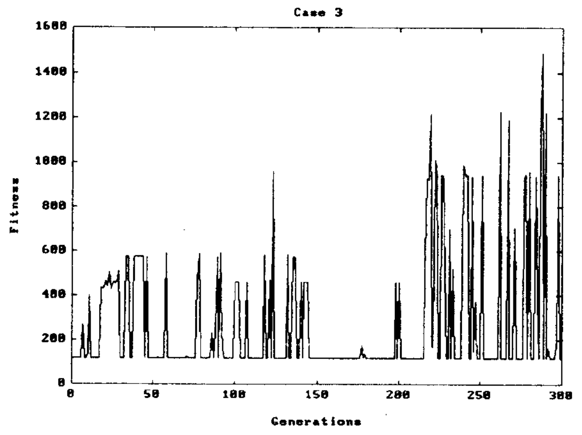

Figure 5. The result of the case 3.

References

[1] M. J. Patel and V. Maniezzo, "NN's and GA's: Evolving co-operative behavior in adaptive learning agents," *IEEE International Conference on Evolutionary Computation*, pp. 290-295, 1994.

[2] I. K. Jeong and J. J. Lee, "Adaptive simulated annealing genetic algorithm for control applications," *Int. J. Sys. Sci.*, vol. 27, no. 2, pp. 241-253, 1996.

[3] M. McInerney and A. P. Dhawan, "Use of genetic algorithms with backpropagation in training of feed-forward neural networks," *IEEE International Conference on Neural Networks*, pp. 203-208, 1993.

[4] I. K. Jeong and J. J. Lee, "A modified genetic algorithm for neurocontrollers," *IEEE International Conference on Evolutionary Computation*, pp. 306-311, 1995.

[5] K. Kristinsson and G. A. Dumont, "System identification and control using genetic algorithms," *IEEE Trans. Syst., Man, Cybern.*, vol. 22, no. 5, pp. 1033-1046, Sep., 1992.

[6] C. L. Karr and E. J. Gentry, "Fuzzy control of pH using genetic algorithms," *IEEE Trans. Fuzzy Syst.*, vol. 1, no. 1, pp. 46-53, Feb., 1993.

[7] Y. Ichikawa and T. Sawa, "Neural network application for direct feedback controllers," *IEEE Trans. Neural Networks*, vol. 3, no. 2, pp. 224-231, Mar., 1992.

[8] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning.* Reading, MA, Addison-Wesley, 1989.

[9] J. D. Lohn and J. A. Reggia, "Discovery of self-replicating structures using a genetic algorithm," *IEEE International Conference on Evolutionary Computation*, pp. 678-683, 1995.

[10] L. Davis, *Handbook of Genetic Algorithms.* New York, Van Nostrand Reinhold, 1991.