An Evolutionary Approach for Learning Soccer Strategies

Tomoharu Nakashima, Masahiro Takatani, Masayo Udo, and Hisao Ishibuchi
College of Engineering, Osaka Prefecture University
Gakuen-cho 1-1, Sakai, Osaka 599-8531, Japan
{nakashi, takatani, udo, hisaoi}@ie.osakafu-u.ac.jp

Abstract - This paper shows an idea of evolving team strategies for the RoboCup soccer simulator. In our evolutionary method, the soccer field is divided into 48 subareas. The action of an agent is specified for each subarea. The agent's action is also determined by the position of the nearest opponent agent. The action is taken from a prespecified set that consists of ten action behaviors. The results of our computer simulations show that the team strategy is successfully evolved. We also show some extensions to our evolutionary computation method.

I INTRODUCTION

RoboCup soccer [1] is a competition between soccer robots/agents and its ultimate purpose is to win against the human soccer champion team by the year 2050. There are various soccer leagues in RoboCup such as humanoid robots, middle-sized real robots, small-sized real robots, four-legged robots, and simulated robots. In this paper, we focus on developing simulated robot soccer teams. Developing RoboCup teams involves solving the cooperation of multiple agents, the learning of adaptive behavior, and the resolution to noisy data handling. Many researchers have been tackling these problems. An example of them is the application of soft computing techniques [5].

In general, the behavior of the soccer agents is hierarchically structured. This structure is divided into two groups. One is low-level behavior that performs basic information processing such as visual and sound information. Basic actions such as dribble, pass, and shoot are also included in the low-level behavior. On the other hand, high-level behaviors make a decision from the viewpoint of global team strategy such as cooperative play among the teammates.

For low-level behavior, Nakashima et al.[5] proposed a fuzzy Q-learning method for acquiring the ball intercept skill. In [5], the performance of the agent gradually improved in an on-line manner.

Evolutionary computation has been used to evolve strategies of games. For example, Chellapilla and Fogel [2, 3] proposed a method based on the framework of evolutionary programming to automatically generate a checker player without incorporating human expertise on the game. An idea of coevolution is employed in [2, 3]. For RoboCup soccer agents, a genetic programming approach has been applied to obtain the soccer team strategy in [4]. In [4], the idea of coevolution is also employed. The evolution of team strategy from the kiddy soccer (i.e., all players gather around the ball) to the formation soccer (i.e., each player maintains its own position during the game) is reported.

In this paper, we propose an evolutionary method for acquiring strategies of RoboCup soccer teams. High-level behavior of soccer teams can be obtained by the proposed method. The characteristic feature of the proposed method is that the behavior of the soccer teams is represented by a integer string. This integer string is a concatenated string of individual agent's integer strings. A set of action rules is encoded into the integer strings. The actions of soccer agents are specified by the action rules when they keep a ball. The antecedent part of the action rules is the position of the agent and its nearest opponent agent. The actions to be taken are indicated by the consequent part of the action rules. We examine the effectiveness of the proposed method through computer simulations where a population consists of a single integer string. We also show some future research direction of the evolution of soccer team along with the preliminary experimental results.

II EVOLUTIONARY METHOD

a. Rule-Based Action Selection of Soccer Agents

The task of our evolutionary method in this paper is to learn team strategies that outperform a fixed opponent team strategy. Each soccer agent in our evolutionary method has a set of action rules. The action rules are used only when the agent keeps a ball. In this paper, the agent is considered as keeping a ball if the ball is within the agent's kickable area. The action rules of the following type are used in this paper:

$$\begin{aligned} R_j &: \text{If the agent is in Area } A_j \text{ and} \\ &\quad \text{the nearest opponent is } B_j \\ &\quad \text{then the action is } C_j, \quad j = 1, \ldots, N, \end{aligned} \tag{1}$$

where R_j is the rule index, A_j is the antecedent integer value, B_j is the antecedent linguistic value, C_j is the consequent action, and N is the number of action rules.

The soccer field is divided into 48 subareas as in Fig.1. Each subarea is labeled an integer value. The antecedent integer value A_j of the action rule R_j is one of the integer values in the interval $[1, 48]$. The antecedent linguistic value B_j examines whether the nearest opponent is near the agent

or not. We use two linguistic values for the antecedent part B_j. The nearest opponent agent is regarded as *near* if the distance between the agent and its nearest opponent agent is less than a prespecified value. Otherwise the nearest opponent agent is regarded as *not near* from the agent.

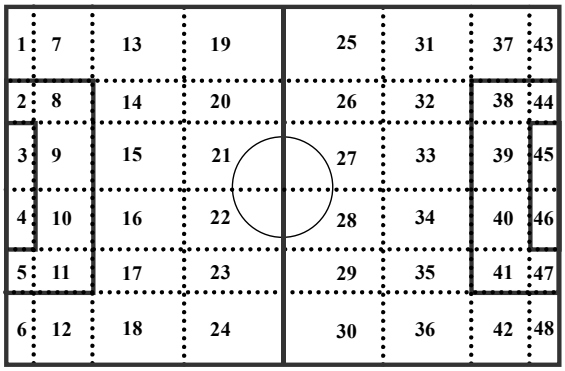

Fig. 1 Divided areas on the soccer field.

The consequent action C_j represents the action that is taken by the agent when the two conditions in the antecedent part (i.e., A_j and B_j) are satisfied. In this paper, we use the following ten actions for the consequent action C_j:

1: Dribble toward the opponent side. The direction is parallel to the horizontal axis of the soccer field.
2: Dribble toward the opponent side. The direction is the center of the opponent goal.
3: Dribble carefully toward the opponent side. The direction is the center of the opponent goal. The dribble speed is low so that the agent can avoid the opponent agent.
4: Pass the ball to the nearest teammate. If the nearest teammate is not ahead of the agent, the agent does not kick to the nearest teammate. Instead, it clears the ball toward the opponent side.
5: Pass the ball to the second nearest teammate. If the second nearest teammate is not ahead of the agent, the agent does not kick to the second nearest teammate. Instead, it clears the ball toward the opponent side.
6: Clear the ball toward the opponent side.
7: Clear the ball toward the nearest side line of the soccer field.
8: Kick the ball toward the penalty area of the opponent side (i.e., centering).
9: Perform a leading pass to the nearest teammate.
10: Shoot the ball toward the nearest post of the opponent goal.

There is a special case where agents do not follow the action rule. If the agent keeps the ball within the penalty area of the opponent side (i.e., if the agent is in Areas 38 ~ 41 or 44 ~ 47 in Fig.1), the agent checks if it is possible to shoot the ball to the opponent goal. The agent decides to shoot the ball if the line from the ball to either goal post of the opponent side is clear (i.e., there are no agents on the line). If the line is not clear, the agent follows the action rule whose antecedent part is compatible to the agent's situation.

The behavior of the agents that do not keep the ball is prespecified in this paper. Each agent has its home position. The position of the agents is determined as an internally dividing point of the line segment between the ball and its home position. Every time step each agent checks if it is the fastest to the ball considering the positions and the velocities of the ball and all the agents. The fastest agent to the ball approaches the ball at its maximal speed. Once some agent reaches the ball, it follows the action rules in (1).

b. Encoding Scheme

As described in Subsection II.a, the action of the agents is specified by the action rules in (1) when they keep the ball. Considering that the soccer field is divided into 48 subareas (see Fig.1) and the position of the nearest opponent agent (i.e., it is *near* or *not near* the agent) is taken into account in the antecedent part of the action rules, we can see that there are $48 \times 2 = 96$ action rules for each agent. In this paper, we apply our evolutionary method to ten soccer agents excluding the goal keeper. Thus, the total number of action rules for a single team with ten agents is $96 \times 10 = 960$. We encode a set of action rules for a single team into an integer string of length 960. The task of our evolutionary method is to evolve integer strings of length 960 to obtain team strategies with high attacking performance. We do not change the length of the integer strings during the application of the evolutionary method. That is, all the 960 action rules are used by a single team.

c. Evaluation of Integer Strings

In this paper, we evaluate the performance of integer strings by using the results of soccer games. Specifically, we use the scores of the soccer games as performance measure in our evolutionary method. We first check the scored goals by soccer teams that are represented by integer strings. Those teams that attain many goals are considered to be good teams. When the number of the goals is the same among multiple soccer teams, the lost goals are used as a second performance measure. The soccer teams with lower lost goals are evaluated better.

d. Evolutionary Operation

We use only mutation as our evolutionary operation in the evolutionary method in this paper. In the mutation operation, the value of each integer bit is replaced with an integer value in the interval $[1,10]$ with a prespecified mutation probability. It is possible that the replaced value is the same as the one before the mutation operation.

In our evolutionary method, generation update is performed in the following manner. First, a single integer

string is randomly selected from the current population. A new integer string is then generated through the mutation operation. After a prespecified number of new integer strings are generated, we select the best integer strings from the merged of the current integer strings and the new integer strings. The selected integer strings form the next population. This generation update is similar to the (μ+λ)-strategy of evolution strategy [6].

To summarize, our proposed evolutionary method is written as follows:

[Procedure of the proposed evolutionary method]

Step 1: Initialization. A prespecified number of integer strings of length 960 are generated by randomly assigning an integer value from the interval for each bit.

Step 2: Generation of new integer strings. First randomly select a integer string from the current population. Then the mutation operation is performed for the selected integer string in order to generate a new integer string. This process is iterated until a prespecified number of new integer strings are generated.

Step 3: Performance evaluation. The performance of both the current integer strings and new integer strings generated by Step 2 is evaluated through the results of soccer games. Note that the performance of current integer strings is evaluated every generation because the game results are not constant but different game by game.

Step 4: Generation update. From the union set of the current integer strings and new ones, select best integer strings according to the performance evaluation in Step 3. The selected integer strings form the next generation.

Step 5: Termination of the procedure. If prespecified termination conditions are satisfied, stop the procedure. Otherwise go to Step 2.

III COMPUTER SIMULATIONS

a. Experimental Settings

The following parameter specifications were used for all the computer simulations in this paper:

• The number of integer strings in a population: 1,

• The number of new integer strings generated by the mutation operation: 1,

• The probability of mutation for each bit: 1/96,

• Generation of initial integer strings: Hand-coding.

In Fig.2, we show the position of the ten soccer agents. The team of agents maintains a so-called 4-3-3 formation where Agent 1 ~ 4 are defenders, Agent 5~7 are mid-fielders, and Agent 8 ~ 10 are forward players. Note that the actions of the goal keeper are not considered in our evolutionary method. The position of soccer agents is determined by the internally dividing point on the line segment between the ball and their home positions.

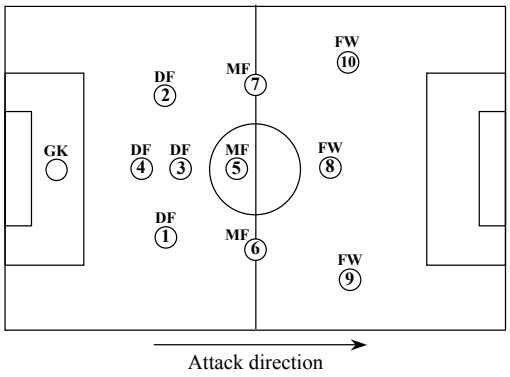

Fig. 2 Ten soccer agents and their home positions.

Since we want to speed up the evolution process of the proposed method, the initial integer string was hand-coded by us. That is, we manually generated an initial integer string. We show a part of the initial integer string in Fig.3.

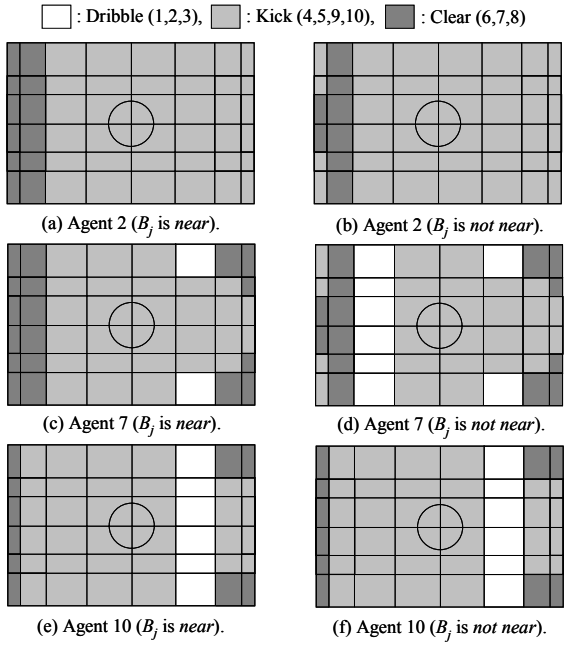

Fig. 3 Initial integer string for Agent 2, 7, and 10.

Each of the new integer string and the current integer string plays a soccer game with some fixed team strategy in the second evaluation scheme. For the fixed team strategy, we use UvA Trilearn2003 base team [7]. UvA Trilearn is the champion team in RoboCup 2003 world competition. They release source codes of the soccer team after codes for high-level decision making are omitted. The strategy of UvA Trilearn agent is to kick the ball toward the opponent goal whatever the position of the agent is in. After both teams play the soccer game with UvA Trilearn 2003 base team, the game results are compared. The integer string that scored higher goals becomes the next integer strings. If the scored goals are the same, the integer string with the lower lost goals becomes the next string.

b. Simulation Results

We examined the performance of the obtained integer strings by our evolutionary method. In our evolutionary computation method, each of the current integer string and the new integer string competes with the fixed team strategy. We use the UvA trilearn base team [7] as a fixed team strategy. The better integer string becomes the next integer string. We performed the evolutionary process for 600 generations. We examine the performance of the integer strings obtained at 100th, 200th, 300th, 400th, 500th, and 600th generations.

Table 1 Simulation results.

Generation	Win	Lost	Draw	Goals	Goals lost
0	1	3	6	3	6
100	2	4	4	3	7
200	3	1	6	7	3
300	1	4	5	1	4
400	1	2	7	3	4
500	2	3	5	2	4
600	4	2	4	8	5

IV FURTHER EXTENSIONS

a. Population of More Than One Integer Strings

Although only one integer string was used in our evolutionary computation, a population consists of multiple individuals in general. The problem here is that it takes ten mins. to complete a single simulated soccer game. That is, if we have 100 integer strings, it takes $10 \times 100 = 1000$ mins. (about 17 hours) to evaluate all the 100 integer strings. One solution to this problem is to use a parallel distributed computing. We are now undergoing computer simulations with five integer strings by using a 16-node cluster machine. Since three nodes are used to complete a single game (two nodes for soccer teams and one for the soccer simulator), we can run five soccer game at the same time.

b. Structured Mutation

Another possible extension to our evolutionary computation is to modify the mutation operation. Since the ten actions that are described in Subsection II.a are categorized into three groups (dribble, kick, and shot), we can have a structure in the mutation operation. That is, the integer value in the mutated integer bit is likely to be replaced with an integer value that is in the same group. This modified local search makes only a little change in the behavior of the soccer team.

c. Simulation Results

We show simulation results with the two extensions described in Subsections IV.a and IV.b in Table 2. In Table 2, we only show the simulation results up to 300 generations. This is because the computer simulation with five integer strings has just begun and we do not have enough simulation results. Nevertheless, from Table 2, we can see that the performance of the soccer teams gradually improve with the number of generations. By comparing the simulation results in Table 2 with those in Table 2, we can see that the two extensions work effectively on the performance of our evolutionary computation.

Table 2 Simulation results with the two extensions.

Generation	Win	Lost	Draw	Goals	Goals lost
0	1	9	0	3	28
100	2	4	4	11	15
200	3	5	2	11	12
300	7	2	1	15	10

V CONCLUSIONS

In this paper, we showed an evolutionary method for learning team strategies of RoboCup soccer agents. The action of soccer agents that keep the ball is determined by the proposed method. The antecedent part of the action rules includes the position of the agents and its nearest opponent. The soccer field is divided into 48 subareas. The candidate actions for the consequent part of the action rules consist in a set of ten basic actions such as dribble, kick, and shoot. The strategy of a soccer team is represented by an integer string of the consequent actions. The results of the computer simulations showed that the attacking performance is improved during the execution of the proposed evolutionary method. We also showed two extensions to our evolutionary computation method. The results of the computer simulation showed that the effect of the two extensions on the performance of the soccer teams.

REFERENCES

[1] RoboCup official page, http://www.robocup.org.

[2] K. Chellapilla, and D.B. Fogel, "Evolving Neural Networks to Play Checkers Without Relying on Expert Knowledge", IEEE Trans. on Neural Networks, Vol. 10, No. 6, pp. 1382-3191, 1999.

[3] K. Chellapilla, and D.B. Fogel, "Evolving an Expert Checkers Playing Program Without Using Human Expertise", IEEE Trans. on Evolutionary Computation, Vol. 5, No. 4, pp. 422-428, 2001.

[4] S. Luke and L. Spector, "Evolving Teamwork and Coordination with Genetic Programming", Proceedings of the First Annual Conference on Genetic Programming, pp.150-156, 1996.

[5] T. Nakashima, M. Udo, H. Ishibuchi, "A Fuzzy Reinforcement Learning for a Ball Interception Problem", RoboCup 2003: Robot Soccer World Cup VII, in press.

[6] T. Bäck, Evolutionary Algorithms in Theory and Practice. Oxford University Press, New York, NY, 1996.

[7] UvA Trilearn description page, Available at http://carol.science.uva.nl/~jellekok/robocup/index_en.html.