2006 IEEE Congress on Evolutionary Computation
Sheraton Vancouver Wall Centre Hotel, Vancouver, BC, Canada
July 16-21, 2006

Robust Evaluation of RoboCup Soccer Strategies by Using Match History

Tomoharu Nakashima, *Member, IEEE*, Masahiro Takatani, *Student Member, IEEE*,
Naoki Namikawa, *Student Member, IEEE*, Hisao Ishibuchi, *Member, IEEE*, Manabu Nii, *Member, IEEE*

***Abstract*— In this paper we improve the performance of an evolutionary method for obtaining team strategies in a simulated robot soccer domain. In the previous method each team strategy was evaluated based on the goals and the goals against of a single game. It is possible for a good team strategy to be eliminated from the population in the evolutionary method as there is a high degree of uncertainty in the simulated soccer game. In order to tackle the problem of uncertainty, we propose a robust evaluation method using match history. The performance of team strategies in the proposed method is measured by the average goals and average goals against. Through a series of computational experiments, we show the effectiveness of our robust evaluation method.**

I. Introduction

RoboCup soccer [1] is an international project of developing autonomous soccer robots/agents. Its ultimate aim is to win against the human soccer champion team by the year 2050. Developing RoboCup teams typically involves solving the cooperation of multiple agents, the learning of adaptive behavior, and the problem of noisy data handling. Many approaches have been presented to tackle these problems. An example is the application of soft computing techniques [2].

In general, the behavior of a soccer agent is hierarchically structured. This structure is divided into two groups. One is low-level behavior which performs basic information processing such as visual and aural information. Basic actions such as dribble, pass, and shoot can also be included in the low-level behavior. On the other hand, the high-level behavior makes a decision from the viewpoint of global team strategies such as cooperative play among the teammates.

For the low-level behavior Nakashima et al. [2] proposed a fuzzy Q-learning method for acquiring a ball intercept skill. It is shown that the performance of the agent gradually improves by trial and error.

Evolutionary computation has been used to evolve strategies of various games. For example, Chellapilla and Fogel [3], [4] 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 also employed in [3], [4]. For the RoboCup domain a genetic programming approach has been applied to obtain the soccer team strategy in [5]. In [5] the idea of coevolution is also employed. The evolution of the team strategy from kiddy soccer (i.e., all players gather around the ball) to formation soccer (i.e., each player maintains its own position during the game) is reported.

Another evolutionary method for RoboCup soccer has been proposed by Nakashima et al. [6] where a set of action rules is used to determine the action of players. The antecedent part of the action rule concerns the positions of the nearest opponent and the position of the player. Each action rule is examined whether its antecedent part matches with the players' current situation. The action of the player is specified by the consequent part of the action rule that exactly matches the current situation. A strategy for a single team is constructed by concatenating a set of action rules of the agents in the team. Each set of action rules is represented by an integer string and treated as an individual through the course of evolutionary process. It was shown that the performance of the generated soccer team improves against a fixed opponent team.

The performance of the team strategy in [6] is evaluated by scores (i.e., goals and goals against) in the game. However, a problem arises because there is a high degree of uncertainty in the game. Different game results can be obtained when a team played against the same opponent team multiple times. We observed that some good teams are eliminated from the evolution process because their performance is not good enough in a single evaluation. On the other hand, poor teams can remain in the evolution process as they happen to score more than the potentially high-performance teams.

In order to tackle the problem of uncertainty, we propose a robust evaluation method of the soccer team strategy where the performance of soccer team strategies is evaluated by using the match history of the teams. In the evolutionary process, those integer strings with high average goals have a higher chance of survival than those with low average goals. Through a series of computational experiments, we show that potentially high-performance teams survive more than low-performance teams.

II. Team Setup

A. UvA Trilearn: Base Team

In this paper we use UvA Trilearn for our evolutionary computation method. UvA Trilearn won the RoboCup world competition in 2003. The source codes of UvA Trilearn are available from their web site [7]. Low level behaviors

Tomoharu Nakashima, Masahiro Takatani, Naoki Namikawa and Hisao Ishibuchi are with the Department of Computer Science and Intelligent Systems, Osaka Prefecture University, 1-1 Gakuen-cho, Naka-ku, Sakai, Osaka 599-8531, Japan (email: nakashi@cs.osakafu-u.ac.jp, takatani@ci.cs.osakafu-u.ac.jp, namikawa@ci.cs.osakafu-u.ac.jp, hisaoi@cs.osakafu-u.ac.jp).

Manabu Nii is with Department of Computer Engineering, University of Hyogo, 2167 Shosha, Himeji, Hyogo 671-2201, Japan (email: nii@eng.u-hyogo.ac.jp).

0-7803-9487-9/06/$20.00/©2006 IEEE

such as communication with the soccer server, message parsing, sensing, and information pre-processing are already implemented. Basic skills such as player's positioning, ball intercept, and kicking are also implemented in the UvA Trilearn source codes. High level behaviors such as strategic teamwork, however, are omitted from the source codes.

UvA Trilearn players take rather simple actions as high level behaviors are not implemented in the released source codes. We show the action tree of UvA Trilearn players in Fig. 1. There are two action modes: One is ball handling mode, and the other is positioning mode. Each player uses one of these two modes in every time step depending on its situation in the soccer field. If a player is nearer to the ball than the rest of the team players, ball handling mode is invoked. On the other hand, when a player is not the nearest one to the ball, it goes into positioning mode.

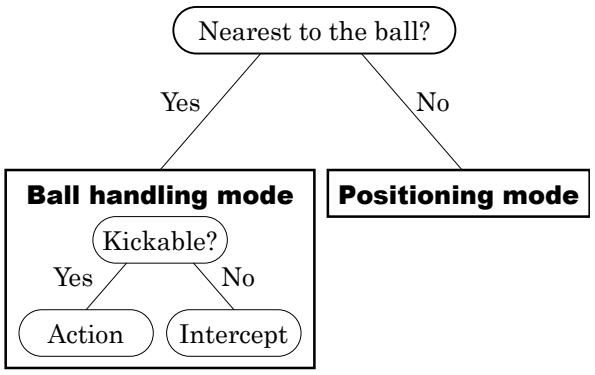

Fig. 1. Action tree of UvA Trilearn players

Note that the action tree in Fig. 1 is common for all players. Thus, the action of players is the same if they are in the same situation. Since their conditions and home positions are different from each other, the action taken at a time step is not necessarily the same for all players. The following subsections explain these two modes in detail.

B. Ball Handling Mode

The ball handling mode is employed when a player is the nearer one to the ball than the rest of the team players. In this mode, the player checks whether it is possible to kick the ball or not. A kickable margin is defined by the RoboCup soccer server. We show the kickable margin of a player in Fig. 2. A player can kick the ball if the ball is in the kickable area of the player, otherwise it is impossible to kick the ball. In the latter case, the player moves towards the ball until the ball is within its kickable area.

According to the UvA Trilearn source codes, the player always shoots the ball to the opponent goal if the ball can be kicked. We modify this behavior for our evolutionary computation. We use action rules to determine the action of the player that can kick the ball (i.e., the ball is within the kickable area of the player). The action rule set represents the strategy of a soccer team. In this paper we evolve action rule sets to find a competitive soccer team strategy.

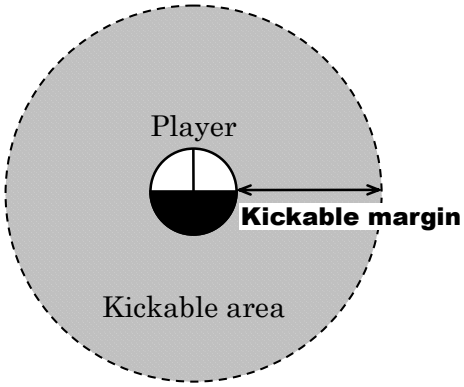

Fig. 2. Kickable margin and kickable area

The action rules of the following type are used in this paper:

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

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 antecedent integer value A_j, $j = 1, \ldots, N$ refers to a subarea of the soccer field. We divide the soccer field into 48 subareas as in Fig. 3.

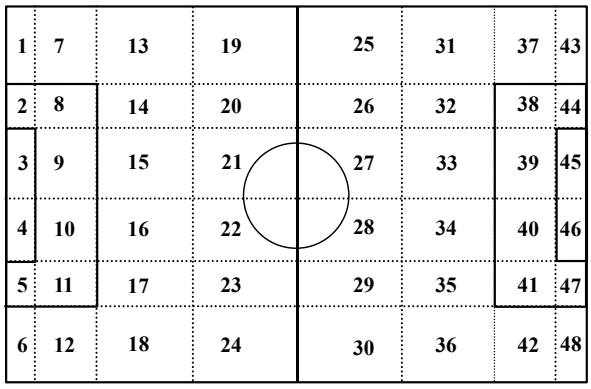

Fig. 3. Soccer field

Each subarea is indicated by an integer value. The antecedent value A_j of the action rule R_j is hence an integer value in the interval [1, 48]. In this way, the action of a soccer agent depends on the position of the agent. The action of the soccer agent also depends on the distance between the agent and its nearest opponent. The antecedent B_j takes one of two linguistic values *near* or *not near*. The player that is able to kick the ball examines whether the nearest opponent is near the agent or not. The nearest opponent is regarded as *near* if the distance between the agent and its nearest opponent is less than a prespecified value. If not, the nearest opponent is regarded as *not near*. The consequent action C_j represents the action that is taken by the agent when the two conditions in the antecedent part of the action rule R_j (i.e., A_j and

B_j) are satisfied. In this paper we use the following twelve 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 opponent agents.
4. Dribble toward the nearest post of the opponent goal.
5. Dribble carefully toward the nearest post of the opponent goal. The dribble speed is low so that the agent can avoid the opponent agents.
6. Dribble toward the nearest side line.
7. 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.
8. 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.
9. Clear the ball toward the opponent side.
10. Clear the ball toward the nearest side line of the soccer field.
11. Kick the ball toward the penalty area of the opponent side (i.e., centering).
12. Perform a leading pass to the nearest teammate.

Note that each player has a set of action rules. Since there are 48 subareas in the soccer field and *near* and *not near* are available for the second antecedent part in action rules (i.e., B_j), the number of action rules for a single player is $48 \times 2 = 96$. There are $96 \times 10 = 960$ action rules in total for a single team with ten field players. Action rules for a goal keeper are not considered in this paper.

There is a special case where players do not follow the action rules. If a player 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. 3), the player checks if it is possible to shoot the ball to the opponent goal. The player decides to shoot the ball if the line segment from the ball to either goal post of the opponent side is clear (i.e., there are no players near the line segment). If the line is not clear, the player follows the action rule whose antecedent part is compatible to the player's condition.

If the ball cannot be kicked by a player that is in the ball handling mode, the player's action is to intercept the ball, that is, the player moves to catch the ball. In the intercept process the player determines whether it dashes forward or turns its body angle based on the relative distance of the ball to the player.

C. Positioning Mode

UvA Trilearn players have their own home positions in the field (see Fig. 4). We use a 4-3-3 formation system where there are four defenders, three mid-fielders, and three forwards. This formation system is fixed and never changes throughout a game. The home position of a player and the ball position are used to determine the players' position when it is in the positioning mode. The position is specified as an externally dividing point of the ball position and the home position. If the current position of the player in the positioning mode is different from the determined position, the player moves toward the determined position. If the difference between the two positions is not large, the player remains in its current position.

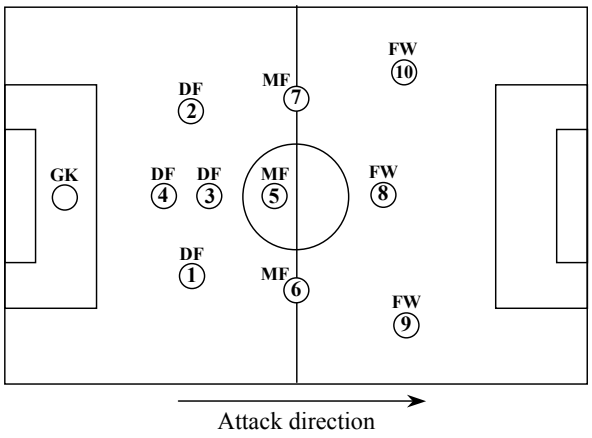

Fig. 4. Home position of the soccer agents

III. Evolutionary Computation

In this paper we use an evolutionary method to obtain team strategies for soccer agents that are effective for playing soccer. Specifically, our aim is to find the best action rule set for ten soccer players. Each player has its own set of action rules that are used when it is in the ball handling mode (see Subsection II-B). We encode an entire team strategy into an integer string. In this paper we refer the rule set for the ten soccer players as a strategy. It should be noted that we do not optimize player's individual behavior but a team strategy as a whole. Thus, we evaluate the performance of a team strategy only from its match result, not from players' individual tactics. We show in our computational experiments that the performance of team strategies successfully improves through the evolution process. The following subsections explain our evolutionary method in detail.

A. Encoding

As described in Subsection II-B, 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. 3) and the position of the nearest opponent agent (i.e., it is *near* the agent or *not near*) 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 player. We apply our evolutionary method to ten soccer agents excluding the goal keeper. Thus, the total number of action rules for a single team is $96 \times 10 = 960$. We use an integer string of length 960 to represent a rule set of action rules for ten players. The task

of the proposed evolutionary method is then to evolve the integer strings of length 960 to obtain team strategies with high performance. We show in Fig. 5 the first 96 integers of an integer string in our evolutionary method. This figure shows an integer string for a single agent. Thus, the integer string of an individual in our evolutionary method has ten such integer strings as we optimize the team strategy with ten soccer agents excluding the goal keeper.

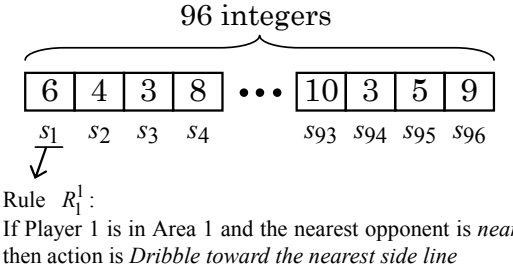

Fig. 5. Integer string for a single agent

In Fig. 5, the first 48 integers represent the action of an agent when the nearest opponent agent is near the agent. The order of the integers is based on the division of the soccer field (see Fig. 3). On the other hand, the actions of the agent in the case where the nearest opponent agent is not near the agent are shown in the other 48 integers. The value of each integer ranges from an integer interval of [1, 12] as the number of possible actions for each rule is twelve. These integer values correspond to the index number of twelve actions described in Subsection II-B.

B. Evaluation of Integer Strings Using Match History

Generally, the main idea of evolutionary methods is to exploit the information of those individuals with a high fitness value is highly evaluated. In the previous work [6], we evaluated the performance of integer strings from a single game result. Since the game has a high degree of uncertainty such as noise in object movement and sensing information, it is usually the case that different game results are obtained from the iteration of the same games. Thus it is possible for potentially high-performance teams to have poor results and conversely it is also possible for low-performance teams to have better results. This problem is caused because the performance of a team is measured from only a single game. One solution to this problem is to play the game multiple times. However, it takes a long time to complete the evolutionary algorithm when multiple games are performed for the evaluation of a single team strategy.

We therefore propose a new evaluation method that uses match history of integer strings. Each integer string has match history since it is newly generated by genetic operations. The average goals and the average goals against are used in the proposed method.

In the proposed method we first check the average goals scored by the soccer teams that are represented by the integer strings. The performance of integer strings is evaluated as high when the number of average goals is high. When the number of the average goals is the same among multiple soccer teams, the number of average goals against is used as a second performance measure. The soccer teams with lower goals against are evaluated as better teams. We do not consider the average goals against at all when the average goals are different between different soccer teams to be evaluated.

C. Evolutionary Operation

We use one-point crossover, mutation, and ES-type selection as evolutionary operations in our evolutionary method. New integer strings are generated by crossover and mutation, and selection is used for generation update.

In the crossover operation, we first randomly select two integer strings. Then latter part of both strings is exchanged with each other from a randomly selected cut-point. Note that we do not consider any evaluation results when two integer strings for the crossover operation are selected from the current population. In the mutation operation, the value of each integer is replaced with a randomly specified integer value in the interval [1, 12] with a prespecified mutation probability. It is possible that the replaced value is the same as the one before the mutation operation. It should be noted that new integer strings generated by the crossover and the mutation operations do not have their match history. Thus the fitness evaluation of the new integer strings is made by using the game result of only a single game.

Generation update is performed by using ES-type selection in our method. We use a so-called $(\mu + \lambda)$-ES [8] for our generation update scheme. By iterating the crossover and the mutation operations we produce the same number of new integer strings as that of current strings. Then the best half integer strings from the merged set of the current and the new strings are chosen as the next population. The selection is based on the match results as described in Subsection III-B. Note that the current strings are also evaluated in this selection process. Thus, it is possible that a current integer string with the best performance at the previous generation update is not selected in the next generation update because the average goals of the integer string after the next performance evaluation may become lower if the result of the game at the next evaluation is poor.

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 [1, 12] for each integer.

Step 2. Generation of new integer strings. First randomly select two integer strings from the current population. Then the one-point crossover and the integer-change mutation operations are performed to generate new integer strings. 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 the new integer strings generated by Step 2 is evaluated through the results of soccer games. Note that the performance of current integer strings is also evaluated every generation because the game results are not constant but different game by game.

Step 4. Generation update. From the merged set of the current integer strings and new ones, select best integer strings according to the performance evaluation using match history in Subsection III-B. The selected integer strings form the next generation.

Step 5. Termination of the procedure. If a prespecified termination condition is satisfied, stop the procedure. Otherwise go to Step 2.

IV. Computational Experiments

The following parameter specifications were used for all the computational experiments in this paper:

The number of integer strings in a population: 5,
The probability of crossover: 1.0,
The probability of mutation for each integer: 5/96.

The population size is specified as five. This is a small number compared to commonly used parameter specifications. This is because it takes at least five minutes to complete a single soccer game. If the population size is specified large, it is difficult to perform the evolutionary method for a large number of generations. Currently we use a 16-node cluster system for the computational experiments in this paper. It still takes several days to perform a single run of the evolutionary process. The population size will be increased when more powerful computational environments are equipped.

The initial population was created by randomly assigning an integer value in the interval [1, 12] for each integer.

We performed the proposed method for 500 generations. Figure 6 shows the average goals and the average goals against at each generation. That is, we examined the performance of the five integer strings after they are selected as the next population. From Fig. 6, we can see that the performance of integer strings becomes better as the number of generations increases. For example, the average goals increase over generation. On the other hand, the average goals against do not change over generation. This is because we focus on the evolution of offensive ability of team strategies. Nevertheless, it should be noted that the average goals against do not increase because we consider not only offensive ability but also defensive ability.

V. Effect of Using Match History

In this section we examine the effectiveness of match history in the evaluation of integer strings. Specifically, we compare the performance of those integer strings that survived for a long period with those that disappeared from the population in a short time. We selected 21 integer strings that survived for different numbers of generations. In Table I we summarize the selected integer strings and the number of generations for which they survived in the population.

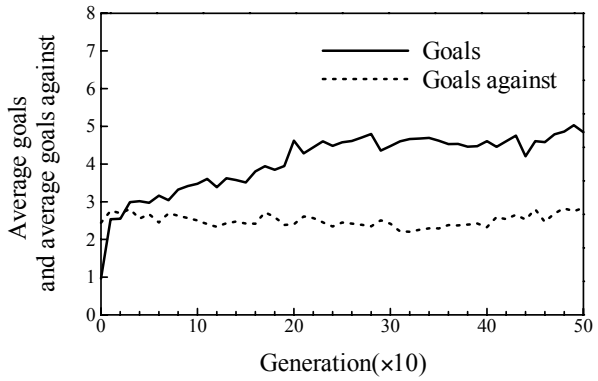

Fig. 6. Simulation results

TABLE I
Selected integer strings

Category (# of generations)	Generation of disappearance		
0-5	175	225	275
5-10	175	225	275
10-15	178	228	275
15-20	175	225	276
20-25	177	230	281
25-30	182	235	286
30-	187	270	300

Each integer string played soccer for 20 times against the fixed opponent team that was used in the evolutionary computation. We show the results of the simulation results in Table II. Table II shows the total results of the games for each category. Since each category has three integer strings, 60 games were conducted for each category in total.

TABLE II
Simulation results

Category	Win	Loss	Draw	Goals	Goals against
0-5	33	17	10	267	171
5-10	44	10	6	274	152
10-15	44	11	5	308	143
15-20	46	8	6	279	135
20-25	44	8	8	279	150
25-30	45	11	4	281	155
30-	44	8	8	305	123

From Table II we can see that those individuals that disappeared from the population in a short time do not perform well. On the other hand, the performance of those integer strings that survived for a long period of generations is good in terms of the number of wins, losses, goals, and goals against. Thus we can see that using match history is effective for increasing the chance of potentially high-performance integer strings to survive in the population for a long time and reducing the risk of maintaining low-performance integer strings.

VI. Frequency-Based Mutation Rate Assignment

Since the length of the integer string is 960, it is possible that some parts of the integer string are not useful or irrelevant for obtaining good team strategies. For example, changing the actions of defenders in Areas 31-37 (see Fig. 3) is not likely to have any effect on the performance of a team strategy. On the other hand, it can be helpful if the action rules are for forwards. The difference between these cases is the frequency of the area. That is, the more frequent an area is used, the greater impact there is on the team behavior. In order to avoid searching for the optimal action in less frequent areas and focus the search on frequently used areas, we employ a frequency-based mutation rate assignment scheme. In this scheme, the mutation rate is not uniformly specified over the integer string, but specified after the evaluation phase of the integer strings. The frequency of each area is monitored during the game. The frequency is used to calculate the mutation rate as follows:

$$P_m(i,j) = 5 \times \frac{c_j^i}{\sum_{k=1}^{96} c_k^i}, \tag{2}$$

where $P_m(i,j)$ is the mutation probability for Area j of Agent i, c_j^i is the frequency of Area j for each i, i.e., this count shows how many times Agent i took an action in Area j during a game. From (2) we can see that the average mutation rate is $5/96$, which is the same as the original mutation rate.

We examined the effect of the frequency-based mutation rate assignment scheme on the performance of the evolutionary algorithm. Except for the mutation rate exactly the same experiments as the previous ones are performed. We show the simulation results in Fig. 7. Figure 7 shows the evolution of team strategies during the execution of the evolutionary algorithm with the frequency-based mutation rate assignment scheme. The average results over five trials are shown in this figure.

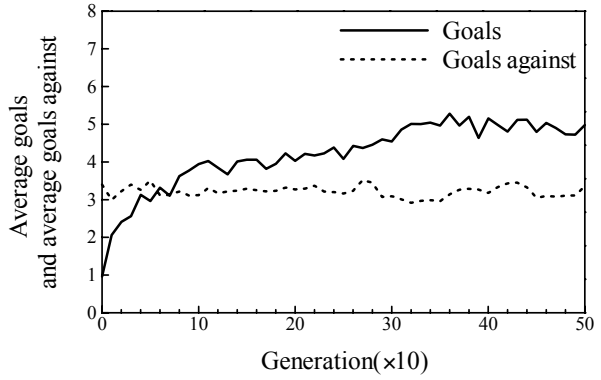

Fig. 7. Simulation results for the frequency-based mutation rate assignment scheme

From Fig. 7, we can see that team strategies with similar performance to Fig. 6 were obtained.

VII. Conclusions

In this paper we proposed an evaluation method of soccer team strategies by using match history. The match history is used to calculate the average goals and the average goals against. Those teams with high average goals are evaluated as better than those with low average goals. The average goals against are used when the average goals are the same among more than one soccer team strategies. This method avoids the problem caused by uncertainty in the RoboCup soccer such as noise in object movement and the sensing information.

In the evolutionary process of this paper the action of soccer players that keep the ball is determined by a set of action rules. The antecedent part of the action rules includes the positions of the agent and its nearest opponent. The soccer field is divided into 48 subareas. The action of the agent is specified for each subarea. The candidate actions for the consequent part of the action rules form a set of 12 basic actions such as dribble and kick. The strategy of a soccer team is represented by an integer string of the consequent actions. In the evolutionary process, one-point crossover, replacement mutation, and ES-type generation update are used as evolutionary operations. The generation update is performed in a similar manner to the $(\mu+\lambda)$-ES of evolution strategy. That is, the best integer strings are selected from a merged set of current integer strings and new integer strings that are generated from the current integer strings by the crossover and mutation operations.

In a series of computational experiments, we examined the performance of our evaluation method. We showed that the performance of the soccer team strategies becomes better over generation. For example, the average goals at the end of the evolution process is larger than in the initial population. We also observed that the average goals against did not increase as the evolutionary computation progressed.

We also showed another trick to improve the performance of the evolutionary algorithm, i.e., the frequency-based mutation rate assignment scheme. Experimental results showed that the performance of the evolutionary algorithm with this scheme can also be improved by the scheme. This is because the scheme successfully makes the evolutionary search focus on important parts of integer strings that are frequently referred during soccer games.

This paper focused on the offensive strategy rather than the defensive one. Developing a method for the defensive strategy is left for our future work.

References

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

[2] T. Nakashima, M. Udo, and H. Ishibuchi, "A Fuzzy Reinforcement Learning for a Ball Interception Problem," *Lecture Notes in Artificial Intelligence 3020: RoboCup 2003: Robot Soccer World Cup VIII*, pp. 559–567, Springer, Berlin, July 2003.

[3] K. Chellapilla and D.B. Fogel, "Evolving Neural Networks to Play Checkers Without Relying on Expert Knowledge," *IEEE Transactions on Neural Networks*, Vol. 10, No. 6, pp. 1382–1391, 1999.

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

[5] S. Luke, "Genetic Programming Produced Competitive Soccer Softbot Teams for RoboCup97," *Proc. of the third Annual Genetic Programming Conference*, pp.204–222, 1998. and L. Spector, "Evolving Teamwork and Coordination with Genetic Programming," in *Proceedings of the First Annual Conference on Genetic Programming*, pp. 150–156, 1996.
[6] T. Nakashima, M. Takatani, M. Udo, H. Ishibuchi, and M. Nii, "Performance Evaluation of an Evolutionary Method for RoboCup Soccer Strategies," *RoboCup 2005: Robot Soccer World Cup IX*, in press.
[7] UvA Trilearn team description page, http://www.science.uva.nl/~jellekok/robocup/index_en.html.
[8] T. Bäck, Evolutionary Algorithms in Theory and Practice, Oxford University Press, New York, 1996.