Proceedings of ITC-CSCC 2000, Pusan, Korea

Co-evolutionary Design of Team Level Play in Soccer Server

Masatoshi Hiramoto*, Hidenori Kawamura*, Masahito Yamamoto*,
Keiji Suzuki** and Azuma Ohuchi*,

* Graduate School of Engineering, Hokkaido University.
Nishi 8, Kita 13, Kita-ku, Sapporo, Hokkaido, 060-8628, Japan
Phone +81-11-716-2111(Ext.6498), Fax +81-11-706-7834
E-mail: {masatosi,kawamura,masahito,ohuchi}@complex.eng.hokudai.ac.jp

**Future University Hakodate
116-2, KanakanoNakano, Hakodate, 041-8655, Hokkaido
Phone+81-138-34-6444, E-mail: suzukj@fun.ac.jp

Abstract Recently, RoboCup soccer simulation has been regarded as a good benchmark problem for multiagent researches. Soccer agents have to make decision based on visual and auditory information, which are sent from the soccer server. In order to develop a strong team, we have to design decision-making process of each player agent. However, it is very difficult for us to design the decision-making processes in detail, because we don't know what actions of each player are effective for the team.

In this paper, we attempt to apply co-evolutionary method, which is one type of analogies of evolution, to improve the team play. Agents have hand coded basic skills, which include dribble, shoot, pass etc. Agents already can play autonomously and independently. Individual agent skills are characterized by some parameters. By coevolving teams with these parameters, we obtained relatively interesting teams, in which players behave cooperatively in order to win the soccer game. From some experiments, we discuss what teams are evolved.

1. Introduction

In the related works to soccer server of RoboCup, many kinds of artificial intelligence and multi-agent technologies are researched in various fields. In the simulation part of soccer server, there are many strong teams developed by detailed designs of hand-coded program based on top-down approaches, but it is exciting if it can be possible to design strong soccer team by the computer program itself. It is very difficult but interesting and challenging to let a computer program to design the intelligent program by itself. The aim of us is to design the strong soccer agent team based on evolutionary technology in the field of artificial intelligence and to overcome champion team by autonomously self-designed agent. As a first step of such aim, we construct co-evolutionary soccer agent team using hand-coded primitive actions. The outline of our approach is described as follows.

Our soccer agent makes a series of actions based on two important factors, that is, basic skills and meta-level decision-making. As known, available actions of soccer agent is very simple, e.g., turn, dash, kick and say. Therefore, we prepare several extended primitive actions, i.e., run, dribble, pass and shoot. These actions are appropriately constructed by combinations of turn, dash kick and say, and applied in the basic skill level with some parameters. In addition, each action is preliminary designed by hand coding. Meta-level decision-making is related with a coordinated strategy among agents, and executes a sequence of primitive actions. Here, generating of meta-level plays is controlled by some meta-level parameters and primitive parameters, e.g., formation parameter, offense-defense parameter, strategy parameter, and so forth. That means behavior of agents is developed by co-evolution. Meta-level parameters and primitive parameters strongly affect the strength of team.

Therefore, we try to let our agent team to evolve autonomously by adjusting meta-level parameters based on co-evolutionary technology often used in genetic algorithm techniques in the computer science area.

2.Agent Description

2.1Primitive Functions

The players have several predefined primitive functions as the basic strategies that are run, dribble, pass, shoot and so forth. The reason why predefined primitive actions were introduced is that it takes a lot of computational costs to let the players acquire these primitive abilities by using conventional learning systems although it is reported that the primitive functions acquired by using a learning system are more adaptive and flexible to the environments of MAS. We designed these primitive actions with some parameters that adjust the extent of the action. These actions are the followings.

MOVE: This function represents a movement from the current position to the destination. The input of this function is the only destination. The speed of this run action can be adjusted by changing dash power according to various situations, for example, when a ball is not controlled by anyone, namely when the player need to go to the ball earlier than any opponents in order to keep the ball, the player run with maximum dash power or when the player return to a home position, the player run with normal dash power to save their stamina.

GO_HOME: This function is an extension of MOVE function. This function enables the player to slightly go to own home position by using back-dash, if it is efficient. When a player is already in their home position, the player does nothing and only turns to the ball.

CHASE_BALL: This function is also an extension of MOVE function. When a player pursues the ball, this function predicts the ball position in future several steps.

DRIBBLE: This function is one of the most important functions in order to keep a ball and to get a score. To perform the Dribble function,. the speed and the destination of a player have to be determined. Speed represents that it takes for the player to move from a current position to the destination with a ball, and is determined according to the stamina of the player and the possibility to get a score. In the case of high speed, the ball is far from the player because the player kicks the ball more strongly and the risk of keeping the ball increases.

PASS: Generally, pass is realized by cooperation between two players. However, this primitive pass function is not defined as such advanced technical skill. Namely, the pass function defines only that the player kicks a ball in a certain direction. The arguments of the function are a relative direction to the destination and a kick power. Actual pass is realized by combination of the pass function and run of the other player.

SHOOT: This function is similar to pass action. The player kicks a ball to the goal in order to get a score by using the shoot function. The argument of the function is only a position in the goal. The player kicks a ball in the direction of the position with the maximum kick power.

2.2 Decision-making Process and Parameters

As we mentioned earlier, an agent (player) decides an appropriate action from the sensed current environment. In order to construct more complex and flexible actions, we adopted meta-level decision-making process. Meta-level decision-making process has a tree structure based on if-then rules using primitive functions. Such an agent architecture is simply shown in Fig.1.

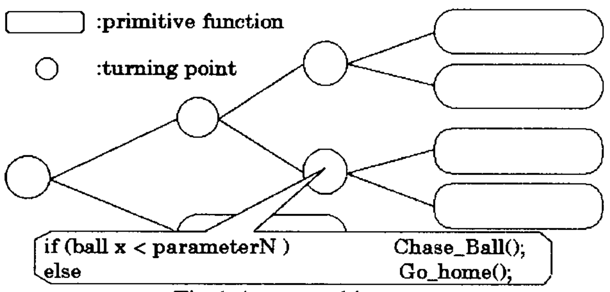


Fig.1 Agent architecture

Some of if-then rules have parameters that strongly influence to agent behavior and the style of plays. In the decision-making process, we defined 10 parameters. Decision-making process and parameters are as follows.

Notation:
<ParameterName>[min vale, max value]: <Explanation>

At the first step in the simulation,

1. Say Prob [0, 1]: probability to execute the say function. Player can tell other teammate about information that he have by using the say function.

2. Ball Reliable [0, 1]: a player decides whether search ball or not according to the value.

If a ball is in kickable area for a player,

3. Shoot Range [0, 50]: distance from the position of the player to the opponent goal. If the player is in nearer than this parameter value, then the player shoots the ball to opponent goal.
4. Pass Line [-54, 54]: represents x-axis in the soccer field. If x-axis of teammate's position is over this parameter value, the teammate is taken into account as a candidate of pass receivers. This parameter can be varied in the range from their own goal line to the opponent goal line.
5. Pass Dist [0, 50]: distance from the position of the player to other teammates. If teammate's distance from the player is over this parameter value, then the teammate is taken into account as a candidate of pass receivers.

Else if the player chase the ball,

6. Chase in Home [0, 50]: distance from player's home position to the ball. If the ball is in nearer than this parameter value, then the player chases the ball.
7. Chase Dist Home [0, 50]: distance from the player to his own home position. If the distance is over this parameter value, then the player doesn't chase the ball.

Else if the player does not chase the ball,

8. Go Home Range [0, 50]: distance from the player to his own home position. If the distance is over this parameter value, then the player goes back to home.
9. Stay Home Range [0, 50]: distance from the player to his own home position. If the distance is less than this parameter value, then the player stays here. That is, player does nothing.

10. Home Position [0,9]: this parameter is different from other parameters. This parameter defines player's home positions. We prepared 10 kinds of home position. This parameter represents one of them. Each player has unique own parameter regarding to home position.

All parameters with exception of Home Position have real values. The value of Home Position is an integer.

3. Co-evolutionary Procedure

Some of if-then rules may have some parameters, which strongly depend upon the style of plays. By representing a set of these parameters as an individual (a team), we performed our co-evolutionary procedure.

The framework of our proposed co-evolutionary procedure is similar to Blair's co-evolution model [Blair, 1999]. Our co-evolutionary procedure is summarized as follows.

1. Challenger is picked up in order.
2. Champion versus Challenger.
3. If Challenger beats Champion, Champion is replaced by the Challenger.
4. New Challenger is created by adding random noise to Champion's set of parameters.
5. New Challenger added to end of Challenger's queue.

This co-evolutionary procedure develops teams over and over. We hope that finally there are high performance teams.
We adopt co-evolutionary procedure based on only game score, as same as [Luke, 1998], because we can avoid defining fitness functions. It is not clear what actions or statistical factors, or which agents contribute to the victory. It has difficulties that we define effective fitness functions. So we assume that co-evolutionary procedure evaluated by only score is relatively clear and reasonable. It is natural to hope that latest generations are superior to early generations. But in general co-evolutionary method should not be expected to show continuous progressive adaptation, because of individual generated by only focusing on generations just before. The method does not consider generations that passed long time to generate new generations. This is also applied to our co-evolutionary procedure. But our experimental results show good performance of latest generations. Experiment setting is described in the following section.

4. Experiment

As similar to the general genetic algorithm, we have a population, actually consists of 10 individuals in our experiments. An individual represents one team. An individual consists of 4 sets of parameters. Each set of parameters has same size. Several players refer to a same set of parameters. 4 sets of parameters respectively represent goaltender, defender, midfielder and offender. One set of parameters is assigned to uniform number 1 as goaltender. Three other roles (sets of parameters) are assigned to players according to x-axis of each player's initial home position. Except Home Position parameter, all parameters that agents use are included by a set of parameters. Each agent has own unique Home Position parameter. These settings are to reduce computational cost, and enable to converge relatively early.
Beginning with a random initial population. Here we use only mutation as genetic operators. Only one match is held between champion and challenger. If the match results in a draw, challenger is disappeared and new challenger is created from the current champion. This procedure is executed over and over. The match is carried out in 4000 simulation steps. It takes almost 7 minutes. And the off side rule is off.
The experiments set is as shown in Table.1.

Table.1 experiment setting

	Mutation rate	Noise range	Population size	Number of trials
Experiment	**7.6%**	**-10% ~+10%**	**10**	**622**

Random noise added to champion. 10% noise means that 10% of max value of parameter is added to current value of parameter. If a parameter added random noise is over its max value, the parameter is set to its max value. If a parameter added random noise is less than its min value, the parameter set to its min value. At first experiment, we evaluate team's progressive evolution.

200 Champions appear in result. We evaluate latest champion by latest champion face ancestral champions every 5 generations. The result is shown as Fig.2

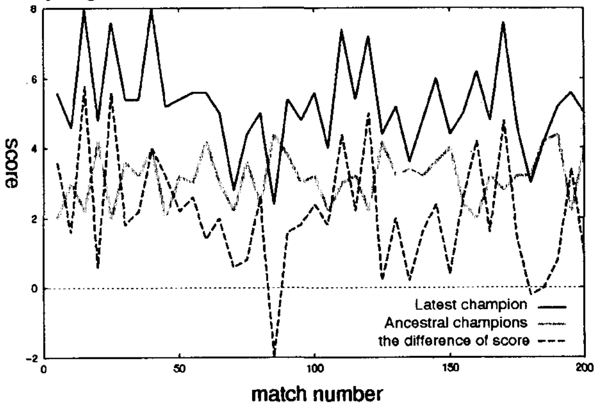


Fig.2 scores between Latest Champion and Ancestral Champions

The lines drawn in Fig.3 are score, one of lines latest champion get score from ancestral champions. Other line ancestral champions get score from latest champion. Another line is the score subtracted the score of ancestral champion from latest champion. The score is calculated means of scores summed 5 matches that latest champion faces same ancestral champion. Latest champion faces one of ancestral champions in turn that ancestral champions appeared. The Smaller match number become, the earlier ancestral champion faces latest champion. If match number is 1, latest champion faces first champion.
In almost all matches, latest champion get scores more than ancestral champions get score. This instructs that latest champion is superior to ancestral champions, and co-evolutionary procedure performs successfully.

Through all matches, ancestral champions get score constantly. We consider that ability of offence regarding ancestral champions doesn't change and the ability of defense regarding ancestral champions is valid through generations.

Now we investigate parameters regarding kick of each champion; parameter number is 3, 4 and 5. The details of these parameters are illustrated in Fig.3. B and C represent particular situations with regard to three parameters. When these parameters change, the strategy of player's action is varied. Player does not pass to teammate in black areas of field in Fig.3. The values of

Pass line or Pass dist become higher, player become not to pass other teammates. For example, at B in Fig.3 Player does not pass other teammate almost all situations. At C in Fig.3 player in opponent side always shoots to opponent goalmouth.

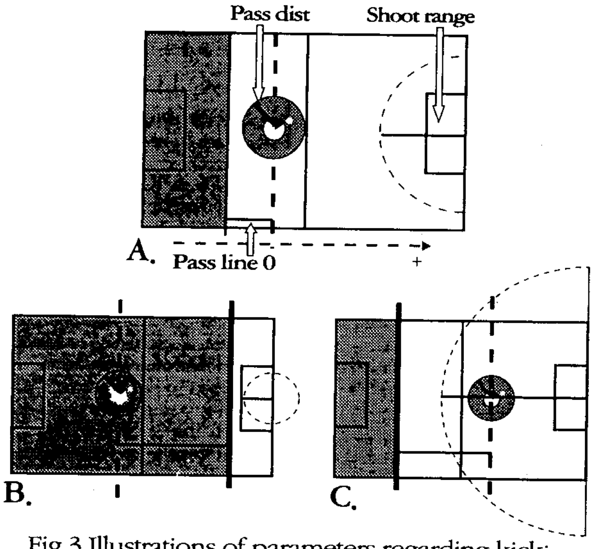


Fig.3 Illustrations of parameters regarding kick: Pass dist, Pass line and shoot range

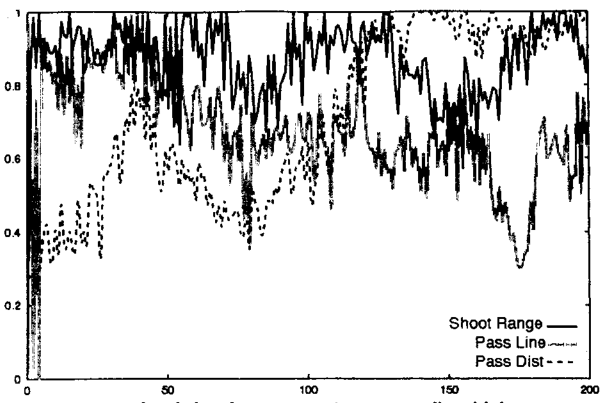
Fig.4 shows three parameters about 200 champions. The vertical axis is value whose ranges [min, max] are changed to [0,1]. A, B and C represent respectively defender, midfielder and offender. Defender gets following strategy. Defender almost does not pass. When defender has ball, defender dribbles to opponent area, and if in Shoot range, then defender shoots. Offender gets same strategy as defender. Offender shoots to goalmouth in long distance to opponent goal. Midfielder's strategy is different. At first stage of champion's midfielder almost does not pass. As generations go, the value of Pass dist becomes lower. That is, midfielder becomes to pass.

The strategy obtained finally as whole team is following thing. When defender possesses ball, defender dribbles to opponent area, and shoot in opponent area. Offender is near opponent goal. Offender picks up the ball shoot by defender and shoot to goalmouth. When Midfielder possesses ball, if there is no teammate near midfielder, midfielder dribbles, otherwise pass the ball to near teammate. And offender receives ball, then shoot to goalmouth.

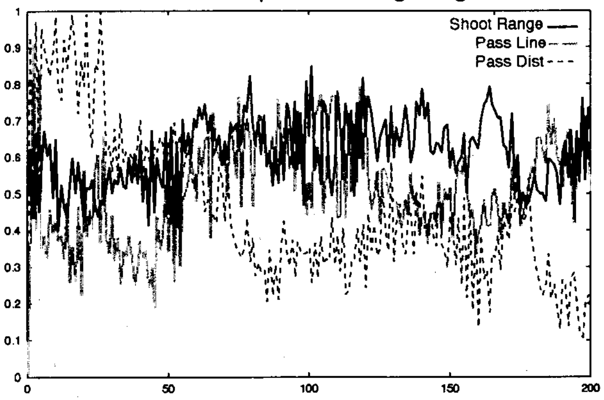
Though only score evaluate the match, team can develop their strategy. The champion obtained finally defeats almost all ancestral champions. We are surprised at this result.

Conclusion

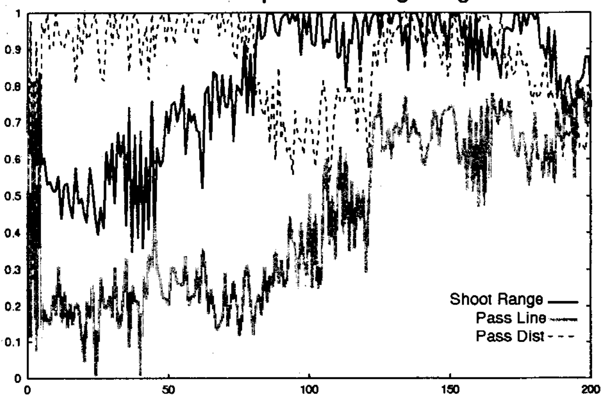
We construct co-evolutionary soccer agent team based on hand-coded primitive actions. Only score evaluate the match. The agent becomes to get own strategy as generations go. We showed that our co-evolutionary procedure performed successfully.



A. defender parameters regarding kick



B. midfielder parameters regarding kick



C. offender parameters regarding kick

Fig.4 parameters regarding kick of defender, midfielder and offender.

References

[Luke, 19998] Luke. S, C. Hohn, J. Farris, G. Jackson, and j. Hendler, Co-evolving Soccer Softbot Team Coordination with Genetic Programming. In RoboCup-97: Robot Soccer World Cup I (Lecture Notes in Artificial Intelligence No. 1395), H. Kitano, ed. Berlin: Springer-Verlag. 398-411 (1998).

[Blair, 1999] A.D. Blair, E. Sklar & P. Funes. Co-evolution, determinism and robustness, X. Yao et al. (Eds.): Proceedings of the Second Asia-Pacific Conference on Simulated Evolution And Learning (SEAL'98) LNCS 1585, 389-396 (1999).

[Pollack 1998] J. B. Pollack and A. D. Blair, ``Co-Evolution in the Successful Learning of Backgammon Strategy". Machine Learning 32, pp. 10-16 (1998).