Description of Team Erika

Takeshi Matsumura
(e-mail:matsu@futamura.info.waseda.ac.jp)

Graduate School of Science and Engineering,
Waseda University, Tokyo, Japan

Abstract. This paper introduces the learning of basic actions by Genetic Algorithm and the effective method for design of agents' cooperation based on role and scenario model.

1 Learning of Basic Actions

To realize cooperations of agents that are the reseach purpose of RoboCup, few actions are provided from soccer server. For example, to realize the pass play that is typical cooperation in soccer, a client would be required the ability to kick a ball in any direction.

Therefore, I made a client learn some basic actions by Genetic Algorithm (GA).

1.1 Genetic Algorithm

Genetic Algorithm(GA) is a kind of random searching technique based on random extraction of samples.[5] GA has a special feature which keeps a set of candidancies of solution and operate it. The candidancy of solution is called a chromosome or individual. An individual is consisted of gens. The set of individual is called population.

GA has a characteristic operator which is called crossover. Crossover operator takes two individuals in population and make new ones by exchange parts of them.

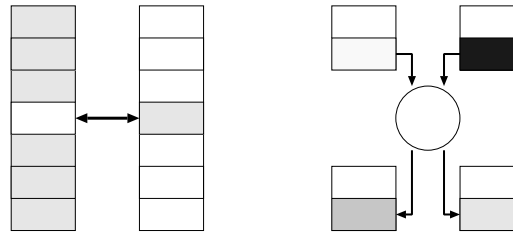

Fig. 1. Two type of crossover operator. Left one changes genes simply. The other takes a pair of genes, makes new pair and puts them back to the individuals.

M. Asada and H. Kitano (Eds.): RoboCup-98, LNAI 1604, pp. 377-383, 1999.

Each individual has a fitness to the solution. Size of populations is a constant value so some individuals whose fitness is lower are thrown away from the new population. This is called Natural Selection. GA obtains a set of individuals that have better fitness to the solution.

The followings are required for GA to leave a set of adaptive individuals [1]:

1. Generating individuals as various as possible in the first generation.
2. Keeping diversity of generation.
3. Preserving characters of individuals that have better fitness.

To keep diversity of population I adopted MGG generation exchange method.[1] To preserve characters of individuals I adopted BLX-α crossover operator.[1]

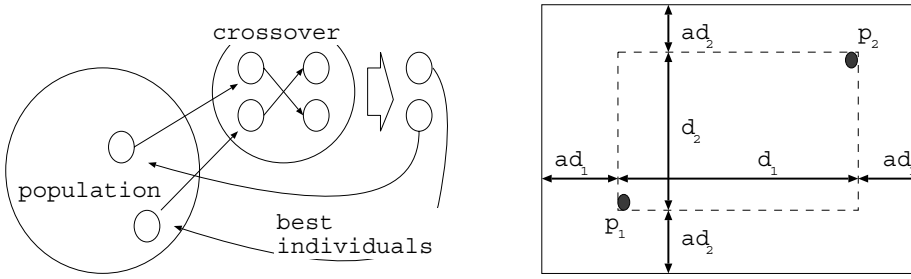

Fig. 2. Left figure showes MGG generation exchange method. A pair of individuals is taken from a population and they produce some individuals using crossover. Only a pair of individuals that have the best fitnesses in them are put back to the generation. The other figure showes BLX-α crossover operator. P_1 and P_2 are parents and a is a constant. Operator takes a pair of individuals in the area enclosed by solid line.

1.2 example of the design : kick-ball-around action

When a player kick a ball forward, we can use the kick command directory. But to kick a ball backwards, a client should kick the ball continuously at an angle tangent to his body in order to turn the ball to the disired direction and then kick it. I assumed the following expressions to calcurate the kick direction θ

$$\theta = a_b + \alpha \cdot vdist \cdot \delta t + \theta'$$
$$\theta' = cos^{-1}\frac{r_p + r_b}{d'} + 90 - tan^{-1}\frac{\epsilon}{d' sin\theta_2}$$
$$d' = d + \alpha \cdot vdist \cdot \delta t$$

where d is a distance between the player and ball, δt is one simulator cycle, a_0 is absolute angle of player, a_b is relative angle from a_0, r_p and r_b are respectively

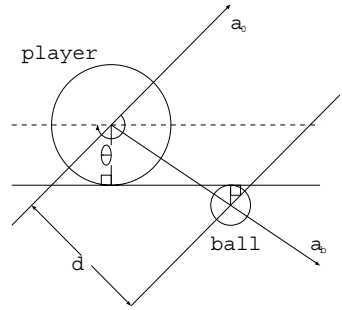

Fig. 3. kick direction

radius of player and ball. *vdist* and *vdir* are respectively DistChng and DirChng of ball which soccer-server takes. The term of $\tan^{-1}$ was added to avoid the colligion of player and ball.

Constants α and ϵ in the previous expressions, and another constant Pk that is the power player kick the ball make up an individual used by GA.

1.3 Result of Learning

I could get some result of the learning of kick-ball-around action.

α	ϵ	Pk	fitness
1.68	0.44	17	1403
1.71	0.60	17	1397
1.68	0.76	18	1396
1.68	0.23	17	1394
1.65	0.66	18	1372

Table 1. Best five individuals in the last population. α,ϵ and Pk consist an individual and the fitness is the total angle that player could turn the ball continuously.

I did this simulation with 1000 population for 10 generations. The best individual was got at the 7th generation, and the client which has these parameters could turn the ball around him continuously more than 3 times.

2 Cooperation based on Scenario and Role

2.1 Introduction

Since there may be a sudden change of situation which is caused by other player in the multi-agent soccer an intention control program is needed. Such a program

will be very complex and redundant it describing all the conditions to deal with such sudden changes. Therefore, I introduced two ideas, role and scenario.[3]

2.2 Behavior of Role and Scenario

```
Scenario   : Ball interception by DF(Defender)

Role       : 1. intercept and cut shoot line
             : 2. cut pass line
             : 3. cut another shoot line

Terminate  : * Ball has gone after the DF area.
condition      --> Back-up-of-Goalie
   and
  next       * Ball has gone before the DF area.
scenario       --> Return-his-position

               * Ball is out of side line.
                --> Kick-in-action

               * One of defender obtains Ball.
                --> Pass-forward-or-Clear

Condition  : * Nearest to Ball        --> role 1
 to get      * Be on the offside line --> role 2
 a role      * Otherwise              --> role 2,3
```

Fig. 4. Example of scenario. If a player of the Defender group is the nearest one to the ball,he gets role 1 to play and if he is able to get the ball, the terminal condition of this scenario is satisfied so he gets Pass-forward-or-Clear as the next scenario. But if there is another teammate who becomes closer to the ball while he is getting the ball, he changes his role to role 2 or 3.

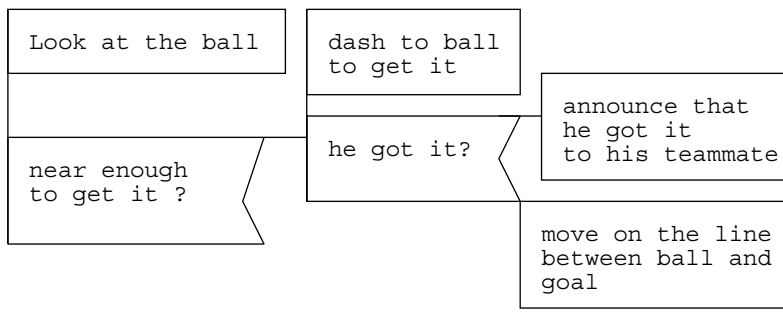

Fig. 5. Example of role. This PAD represents role 1 which is used in the example of scenario in figure 5.Noted that the role does not need any conditions to terminate the scenario.

A role is a function that an agent executes. Scenario is a structure consisted of two functions. Role-selector is a function for an agent to get his role and Scenario-terminator is terminal condition for the scenario. If Scenario-terminator is satisfied, it terminates current role, gives up the scenario and then take another one.

Every agent in the Team belongs to a group and they alway works under the same scenario. If a group gets a scenario, every agent in the group evaluates Role-selector of the scenario to get his own role to play.

Scenario-terminator is checked continuously as a back ground process. We do not take care of the terminal condition when describing a role.

After finished his role, an agent evaluates Role-selector of the scenario to get his role again.

2.3 Moving Agents between Neibouring Groups

There are four groups in my team, Forward(FW), Midfielder(MF), Defender(DF) and Goalie(GK). Every agent is in his initial group at first, but he can move to another group. If a group got a scenario that requires more agents than the group has, it can move some agents from neibouring groups.

```
Scenario : Goal Kick
Role     : 1. Kicker
         : 2. Receiver

Terminate:  * Receiver got the ball
condition     and he returned to DF group
   and       --> Return-home-position
  next       * Opponent got the ball
Scenario
             --> Intercept

Condition:  * Goalie --> Kicker
 to get
 a role      * Otherwise --> Receiver
```

Fig. 6. Scenario example which needs to get another agent. GK group is usually consisted of only goalie, but this scenario requires 2 roles so the goalie makes another agent move to GK group to safisfy this scenario.

A group can get forcely an agent from another group without permittance of the agent or the group. It can reduce the negotiations but it may cause the struggle of the agent between the groups. Such conflictions will be solved by designing scenarios which need less roles.

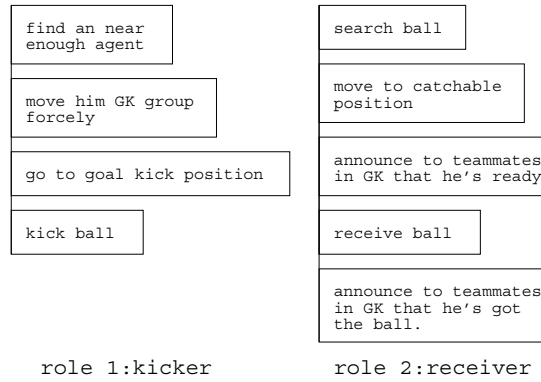

Fig. 7. Roles of kicker and receiver. Receiver is already in GK group when he gets this scenario. He do not know that he was in DF group. And he will be continuously in the GK group until another group orders him to move to.

3 Future work

3.1 Expansion of role-scenario model

At present, every agent always belongs to a group, and every group always has a scenario. There are some problems as follows:

1. It's difficult to move some agents between two groups.
2. The number of scenarios and that of groups are always same.
3. When there are scenario A and B, the scenario A may not require as many agents as the scenario B.

Here it is an improved model.

1. There are no groups.
2. Any agent can belong to any scenario and can have no scenario.
3. Any agent can suggest a scenario any time.
4. Any agent who hears that suggestion and doesn't belong to a scenario tries to take part in that suggested scenario.
5. If there is a role in the scenario which is fit to an agent, the agent takes part in the scenario and play the role.
6. In the case the agent who is belonging to an scenario A suggests another scenario B, the scenario A is terminated and every agent who was taking part in the scenario A is free.

This model gives flexibility to the relation of agent and scenario. In RoboCup soccer agents tend to depend on their places, but they should make a cooperation with other agents who are unexpectedly near of them in the more complex problems like Disaster Relief. This model is effective to make such a multi-agent system.

References

1. Masayuki Yamamura: The theory and practice of Genetic Algorithm. Tutorial of Japan Society for Software Science and Technology (1998)
2. Eiichi Osawa, Makoto Yokoo: MultiAgent. Tutorial of Japan Society for Software Science and Technology (1998)
3. Simon Ch'ng and Lin Padgham: Team description: Royal Melbourne Knights. The First International Workshop on RoboCup (1997) 125–128
4. Sean Luke, Charles hohn, Jonathan Farris, Gary Jackson, James Hendler: Co-Evolving Soccer Softbot Team Coordination with Genetic Programming. The First International Workshop on RoboCup (1997) 115–118
5. Lawrence Davis: Handbook of Genetic Algorithms. Van nostrand Reinhold, A Division of Wadsworth, Inc (1990)