Cooperative and Competitive Behavior Acquisition for Mobile Robots through Co-evolution

Eiji Uchibe **Masateru Nakamura** **Minoru Asada**
Graduate School of Eng., Dept. of Adaptive Machine Systems, Osaka University
2-1 Yamadaoka, Suita, Osaka, 565–0871, JAPAN
{uchibe,riteru}@er.ams.eng.osaka-u.ac.jp asada@ams.eng.osaka-u.ac.jp
Tel.: +81 (6) 6879–7349

Abstract

This paper discusses how multiple robots can emerge cooperative and competitive behaviors through co-evolutionary processes. A genetic programming method is applied to individual population corresponding to each robot so as to obtain cooperative and competitive behaviors. The complexity of the problem can be explained twofold: co-evolution for cooperative behaviors needs exact synchronization of mutual evolutions, and three robot co-evolution requires well-complicated environment setups that may gradually change from simpler to more complicated situations. As an example task, several simplified soccer games are selected to show the validity of the proposed methods. Simulation results with fixed and varying fitness functions are shown, and a discussion is given.

1 Introduction

Multiagent simultaneous learning is one of the major challenges facing Robotics and AI. As benchmark for this problem, Robotic soccer (RoboCup) has been increasingly attracting many researchers [1, 5]. They have been attacking a wide range of research issues. Among them, behavior learning in a multiagent environment has been attacked based on reinforcement learning (ex., [9, 10]).

However, it seems difficult to apply conventional learning algorithms such as reinforcement learning to co-evolution of cooperative agents since the environment including other agents may cause unpredictable changes in state transitions for learning agents. We have shown reinforcement learning supported by *Local Prediction Models* and learning schedule [10]. This method estimated the relationships between learner's behaviors and other robot ones through interactions. In this method, only one robot may learn and other robots had to fix their policies for successful learning.

As one alternative to realize multiagent simultaneous learning, co-evolution seems promising [8, 11]. In the realm of nature, we can see various aspects of behaviors emerged in multiagent environments, not only competition but also cooperation, ignorance, and so on. That means there could be artificial co-evolution for other than competition.

In our previous work [11], we have shown the preliminary results to obtain cooperative behaviors through co-evolutionary processes. This paper discusses how multiple robots can obtain cooperative behaviors with the fixed and varying fitness functions. As a task example, a simplified soccer game with three learning robots is chosen and a *Genetic Programming* (hereafter *GP*) method [6, 7] is applied so as to experimentally evaluate obtained behaviors in the context of cooperative and competitive tasks. Each robot has its own individual population, and attempts to acquire desired behaviors through interactions with its environment that is ever changing in the co-evolutionary process. The complexity of the problem can be explained twofold:

1. Co-evolution for cooperative behaviors needs exact synchronization of mutual evolutions.
2. Co-evolution requires well-complicated environment setups that may contribute to providing a wide variety of searching area from simpler to more complicated situations in which they seek for better strategies so that they can emerge cooperative and competitive behaviors simultaneously.

First, we describe our views on co-evolution in the context of cooperative and competitive tasks. Next, we explain our task example, a simplified soccer game in which cooperative and competitive tasks are involved. Then, we give a brief implementation of *GP* and two fitness functions: one is fixed and the other varying. Finally, the results of computer simulation are shown, and a discussion is given.

2 Co-Evolution in Cooperative Tasks

Generally, we have the following three difficult problems in multiagent simultaneous learning:

1. **Unknown Policy**
   Learning agents do not know other agents' policies in advance, therefore they need to estimate them through observations and actions. What's the worse is that the agent policies may change through a learning process.
2. **Synchronized Learning**
   Mutual learning robots have to improve their learned policies simultaneously. If the opponent learning converged much earlier than itself, one robot could not improve its strategy against the difficult environment that its opponent has already fixed.
3. **Credit Assignment**
   Credit assignment to learning robots for cooperation seems difficult. If the credit involves group evaluation only, one robot may accomplish a given task by itself and others do just actions irrelevant to the task as they do not seem to interfere the one robot's actions. Else, if only individual evaluation is involved, robots may compete each other. This trade-off should be carefully dealt.

Co-evolution is one of potential solutions for the first problem by seeking for better strategies in a wide range of searching area in parallel. The second and third ones might be solved by careful designs of environmental setups and fitness functions. Emerging patterns by co-evolution can be categorized into three.

1. **Cycles of switching fixed strategies**
   This pattern can be often observed in a case of a prey and predator which often shift their strategies drastically to escape from or to catch the opponent. The same strategies iterate many times and no improvements on both sides seem to happen.
2. **Trap to local maxima**
   This corresponds to the second problem stated above. Since one side overwhelmed its opponents, both sides reached to one of stable but low skill levels, and therefore no change happens after this settlement.
3. **Mutual skill development**
   In certain conditions, every one can improve its strategy against ever-changing environments owing to improved strategies by other agents. This is real co-evolution by which all agents evolve effectively.

As a typical co-evolution example, a competitive task such as prey and predator has been often argued [2, 4] where heterogeneous agents often change their strategies to cope with the current opponent. That is, the first pattern was observed. In a case of homogeneous agents, Luke *et al.* [8] co-evolved teams consisting of eleven soccer players among which cooperative behavior could be observed. However, co-evolving cooperative agents has not been addressed as a design issue on fitness function for individual players since they applied co-evolving technique to teams.

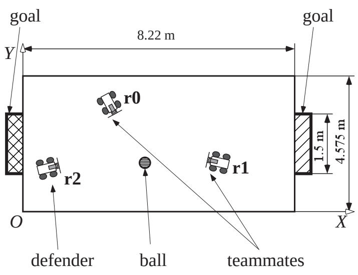
We believe that between one-to-one individual competition and team competition, there could be other kinds of multiagent behaviors by co-evolutions than competition. Here, we challenge to evaluate how the task complexity and fitness function affect co-evolution processes in a case of multiagent simultaneous learning for not only competitive but also cooperative tasks through a series of systematic experiments. First, we show the experiments for a cooperative task, that is, shooting supported by passing between two robots in Section 4.1 where unexpected cooperative behavior regarded as the second pattern was emerged. Next, we introduce a stationary obstacle in front of the goal area into the first experimental set up in Section 4.2 where the complexity is higher and an expected behavior was observed after longer generation changes than the previous one. Finally, we exchange an active learning opponent with the stationary obstacle to evaluate how both cooperative and competitive behaviors are emerged in Section 4.3. We have tried several fitness functions, and we may conclude that the same level fitness functions among them seems better to co-evolve cooperative and competitive agents, and different ones tend to evolve only one side, that is the second pattern.

3 Task and Assumptions

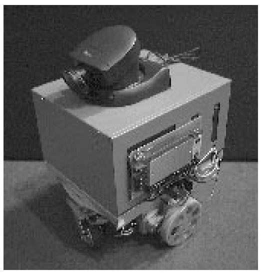
3.1 Environment and Robots

Before explanation of the proposed method, we show a concrete task for reader's understanding of the method. We have chosen a simplified soccer game consisting of two or three robots as a testbed for the problem because both competitive and cooperative tasks are involved as stated in RoboCup Initiative [5]. The environment consists of a ball and two goals, and a wall is placed around the field except the two goals. The sizes of the ball, the goals and the field are the same as those of the middle-size real robot league of RoboCup Initiative. Figure 1 (a) shows the size of the environment and the robot.

Figure 1 (b) shows the real robot used for modeling. The robots have the same body (power wheeled steering system) and the same sensor (on-board TV camera), that is, homogeneous agents. In this simulator, the robot can not obtain the complete information because of limitation of its sensing capability and occlusion of the objects.



(a) The size of an environment and three robots



(b) A real robot

Figure 1: An environment and three robots

Table 1: Function sets

a	ball, goal, other robot 0, other robot 1, $\cdots$
b	left, middle, right, small, medium, large, lost

3.2 Function and Terminal Sets

It is essential to design the well-defined function and terminal sets for appropriate evolution processes. This can be regarded as the same problem to construct the well-defined state space.

As sets of functions, we prepare a simple conditional branching function "IF_a_is_b" that executes its first branch if the condition "a is b" is true, otherwise executes its second branch, where a is a kind of image features, and b is its category. Table 1 shows these features and their categories.

Terminals in our task are actions that have effects on the environment. A terminal set consists of the following four behaviors based on the visual information:

1. `shoot` : the robot shoots a ball into the opponent goal.
2. `pass` : the robot kicks a ball to one teammate.
3. `avoid` : the robot avoids collisions with other robots.
4. `search` : the robot searches the ball by turning to left or right.

These primitive behaviors have been obtained by the reinforcement learning algorithms [10] in the real environment.

3.3 Fitness Measure

Another issue to apply an evolutionary algorithm is the design of fitness function which leads robots to appropriate behaviors. It is so-called "credit assignment problem" [3] in the field of the reinforcement learning: when a trial is complete, which agents get more credit (reward) for its success or failure?

We utilize the standardized fitness representation, that has a positive value. The smaller is the better (0.0 is the best). We first consider the following parameters to evaluate team behaviors such as cooperation between teammates and competition with opponents:

- $G(i)$: the total number of achieved goals for the team to which robot i belongs,
- $L(i)$: the total number of lost goals for the team to which robot i belongs.

With these parameters only, most robots tend to be idle (passive cooperation) except one that attempts at achieving the goal for itself, and therefore no active cooperation has been seen. Then, we introduce the following more individual evaluation to encourage robots to interact with each other while minimizing the number of collisions:

- $K(i)$: the total number of ball-kicking by robot i,
- $C(i)$: the total number of collisions between robot i and others.

In addition to the above, the following is involved to make robots achieve the goal earlier.

- *steps* : the total number of steps until all trials[1] end, where a step is defined as a time period for one action execution corresponding to the sensory input of a robot (1/30 [msec]).

Next, we combine these fitness measures to evaluate individuals. We examine the following two fitness function in the following experiments.

Fixed Fitness Function

The simplest fitness function is calculated by a linear combination of these parameters. In this method, the weights of the fitness function are fixed over the all generations. The fitness value which the robot i receives is given by:

$$\begin{aligned} f^f(i) &= \alpha_g^f \cdot h(G(i), T_{\max}) + \alpha_l^f \cdot L(i) \\ &\quad + \alpha_k^f \cdot h(K(i), \beta^f) + \alpha_c^f \cdot C(i) + \alpha_s^f \cdot \mathit{steps}, \end{aligned}$$

and $h(x,y)$ is a threshold function,

$$h(x,y) = \begin{cases} y - x & \text{if } x < y, \\ 0 & \text{otherwise,} \end{cases}$$

where $T_{\max}$, $\alpha_k^s \sim \alpha_s^s$, and β^s denote the maximum number of trials, and constants. We set $\alpha_g^f = 1.0$, $\alpha_l^f = 0.5$, $\alpha_k^f = 1.0 \times 10^{-3}$, $\alpha_c^f = 5.0 \times 10^{-3}$, $\alpha_s^f = 1.0 \times 10^{-4}$ and $\beta = 4000$, respectively.

[1]One trial is terminated if one of the robots shoots a ball into the goal or the pre-specified time interval expires.

Table 2: Other parameters used in *GP*

the size of each population	80
the number of generations for which the evolutionary process should run	60
the maximum depths during the creation	10
the maximum depths by crossing two trees	25
the crossover probability	95 %
the reproduction probability	5 %
the mutation probability	10 %

Varying Fitness Function

In a case of a fixed fitness function, we do not consider the change of each measure. Ideally, the individual factor (K) and the penalty (C and *steps*) are kept high and low, respectively, at the beginning of the evolution by *GP* in order to reduce the search area.

As the evolution proceeds, the individual factor decreases gradually with the generation index while the penalty factors increase. Near the end of the run, the individual factors reach appropriately small values. As one of such fitness function, a linearly varying fitness function can be expressed as

$$\begin{aligned} f^v(i) &= \alpha_g^v \cdot h(G(i), T_{max}) + \alpha_l^v \cdot L(i) \\ &\quad + \frac{G-g}{G} \cdot \alpha_k^v h(K(i), \beta^v) \\ &\quad + \frac{g}{G}(\alpha_c^v \cdot C(i) + \cdot \alpha_s^v \cdot steps), \end{aligned} \tag{1}$$

where g and G are the generation index and the maximum number of generations, respectively. We set $\alpha_g^v = 1.0$, $\alpha_l^v = 1.0$, $\alpha_k^v = 1.0 \times 10^{-2}$, $\alpha_c^v = 1.0 \times 10^{-2}$, $\alpha_s^f = 1.0 \times 10^{-2}$ and $\beta = 4000$, respectively. If two or more individuals have the same fitness value, we prefer to one with more compact tree depth.

3.4 The GP Implementation

Other parameters used in *GP* are shown in Table 2. The best performing tree in the current generation will survive in the next generation. In order to select parents for crossover, we use tournament selection with size 10. After each population selects one individual separately, the selected individuals participate in the game. We perform 20 games to evaluate them. As a result, it needs 1600 trials to alter a new generation. The hardware used for the simulation is DEC VT-Alpha 600, which takes about 16 hours to evaluate one experiment whole generations.

4 Experimental Results

4.1 Two Learners

At first, we demonstrate the experiments to acquire cooperative behaviors between two robots. Both robots belong to the same team, and they obtain the score if they succeed in shooting a ball into the goal. The number of function sets is 28 (= 7 (ball) +2 × 7 (two goals) +7 (teammate)).

The tree depths and the numbers of nodes in cases of the fixed and varying fitness functions are shown in Table 3 (a). The tree of the best **r0** (expected to be a passer) is deeper than that of the best **r1** (expected to be a shooter) in the fixed fitness functions, but the average depth and the number of nodes of **r0** are smaller than those of **r1** in both fitness functions. Actually, the acquired behavior of **r1** is purposive while **r0** does not move appropriately from a viewpoint of the designer.

One of the successful behaviors based on the fixed fitness function are shown in Figure 2. In this case, **r0** does not kick the ball by itself but shakes its body by repeating the behaviors `search` and `avoid`. On the other hand, **r1** approaches the ball and passes the ball to **r0**. After **r0** receives the ball, it executes a `shoot` behavior. However, **r1** approaches the ball faster than **r0**. As a result, **r1** shoots the ball into the goal while **r0** avoids collisions with **r0**. We checked the case of the varying fitness functions, and found that the resultant behaviors were similar to the behavior by the fixed case. In this task, the best **r0** does not kick the ball toward **r1** at the end of the generations.

We suppose that the reasons why they acquire such behaviors are as follows:

- In order for **r0** to survive by passing the ball to **r1**, **r1** has to shoot the ball which is passed back from **r0**. This means that the development of both robots needs to be exactly synchronized. It seems very difficult for such a synchronization to be found.
- **r1** may shoot the ball by itself whichever **r0** kicks the ball or not. In other words, **r1** does not need the help by **r0**.

In this task, **r0** and **r1** do not have even complexities of the tasks. As a result, the behavior of **r1** dominates this task while **r0** cannot not improve its own behavior. This is the second pattern explained in Section 2.

4.2 Two Learners and One Stationary Robot

Next, we add one robot as a stationary obstacle to the environment described in Section 4.1. The number of function sets is 35 (= 7 (ball) +2×7 (two goals) +2×7 (teammate and opponent)).

Table 3: The average tree depths and the number of nodes

(a) two learners experiments

fitness	depths	# of nodes
fixed	$(24.3, 15.0)$	$(335.5, 909.4)$
varying	$(17.5, 14.7)$	$(594.5, 821.4)$

(b) two learners and one obstacle

fixed	$(15.4, 17.7)$	$(1126.9, 239.7)$
varying	$(14.7, 18.3)$	$(1328.3, 1394.2)$

(c) three learners experiments

fixed	$(23.7, 16.1, 20.0)$	$(1373.2, 1081.8, 772.5)$
varying	$(27.3, 17.3, 19.8)$	$(1635.7, 1184.2, 1083.5)$

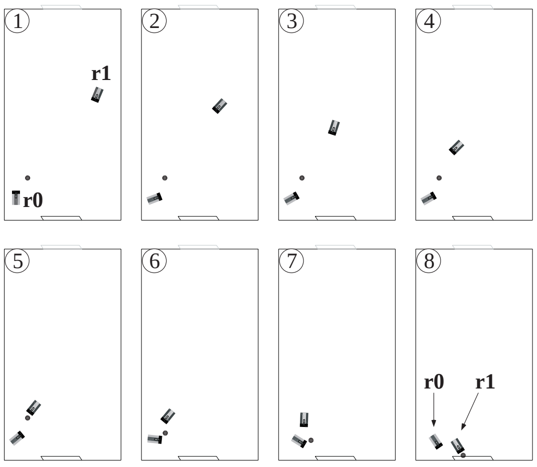


Figure 2: Two robots (**r0** and **r1**) succeed in shooting a ball into the goal

Table 3 (b) shows the tree depth and the number of nodes which are obtained by the fixed and varying fitness functions. The *GP* trees which **r1** acquires are more complicated than those of **r0**. The reason seems that **r1** has to consider both of the shooting behavior and avoiding collisions with **r2**. On the other hand, the resultant behavior of **r0** is only to push the ball to **r1**. At the end of generations, the number of scores is not different between both fitness functions. However, there are difference with respect to the variety of the final population.

Although both learning robots are placed in the same way as in the previous experiments, the acquired cooperative behaviors are quite different. We found the following three patterns in a case of the fixed fitness function:

1. First pattern (ball rolling and accidental goal)
   Because **r0** is placed near the ball, **r0** pushes the ball more frequently than **r1**. Most of individuals of **r0** kick the ball towards **r1** owing to the initial placement. However, some individuals push the ball towards **r2** in the neighborhood of **r0**. Consequently, the ball rolls towards the goal by accident (See Figure 3).

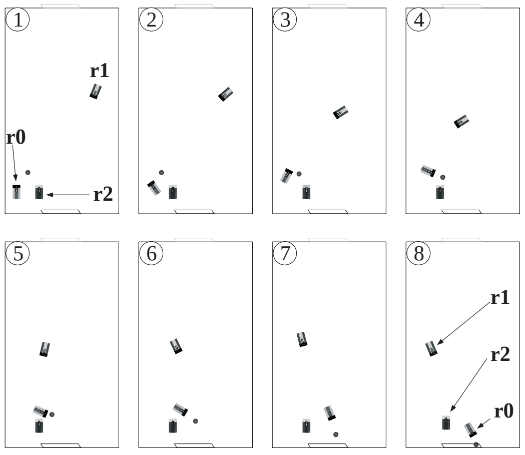


Figure 3: **r0** shoots the ball into the goal

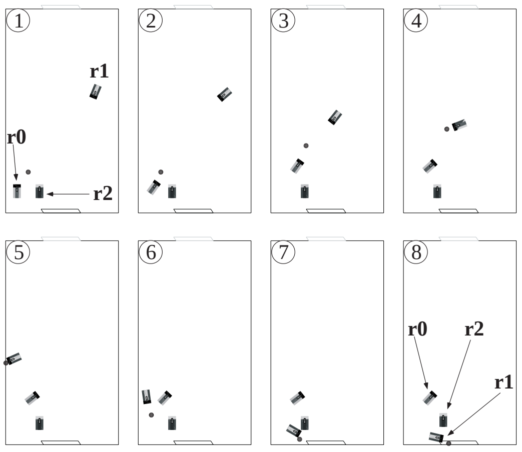


Figure 4: **r1** shoots the ball into the goal along the wall at generation 15

2. Second pattern (goal after dribbling along the wall)
   Although both **r0** and **r1** kick the ball until generation 4, **r0** begins to pass the ball towards **r1**. However, **r1** can not shoot the ball from **r0** directly because **r0** cannot pass the ball to **r1** precisely. Therefore, **r1** kicks the ball to the wall and continues to kick the ball to the opponent's goal along the wall until generation 15 (See Figure 4). After that, the rank of this pattern dropped down.

3. Third pattern (mutual skill development)
   After a number of generations, both robots improve their own behaviors and acquire cooperative behaviors at the end of generations, where **r0** kicks the ball to the front of **r1**, then **r1** shoots

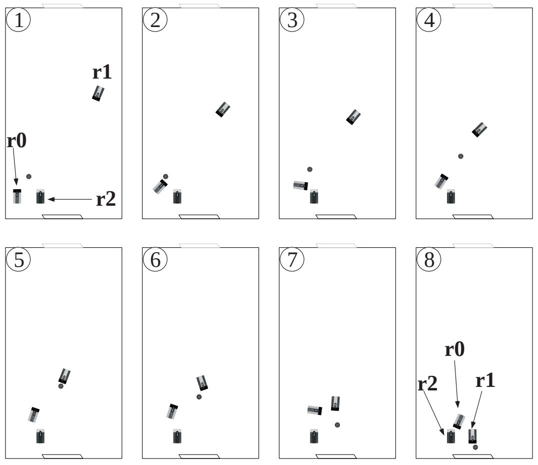


Figure 5: After **r0** pushes the ball toward the front of **r1**, **r1** shoots the ball into the goal avoiding collision with **r2**

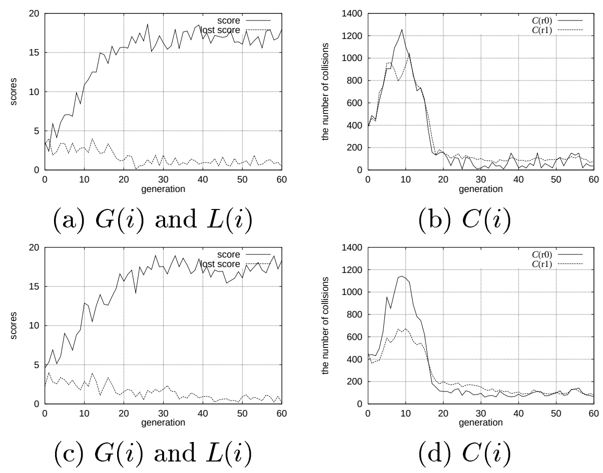


Figure 6: Experimental results in a case of two learners and one stationary agent (a, b:fixed, c, d:varying)

the ball into the opponent's goal shown in Figure 5. As a result, both robots improve the cooperative behaviors synchronously. This is a kind of the mutual development described in Section 2.

The individual of the third pattern obtained the high evaluation because it takes much shorter time to shoot the ball than the first and second patterns. Figure 6 shows the results of evolutionary processes where a good synchronization between the best individuals of **r0** and **r1** can be seen. Since it becomes more difficult for **r1** to shoot the ball for itself because of the existence of **r2**, **r1** has to evolve behaviors with **r0** synchronously. In other words, the complexity of the task for **r1** increased around the same level of **r0**.

On the other hand, when we use the varying fitness function, most of the individuals that are obtained at the end of generations is the third pattern shown in Figure 5. We can not find the first and the second patterns based on the varying fitness function. The reason seems that the varying fitness function does not permit a wider variety of individuals. As we can see from Figure 6, the total number of collisions decreases gradually because the corresponding terms (the fourth and fifth terms in Eq.(1)) become effective through evolutionary process. This leads to the extermination of the first and second patterns.

4.3 Three Learners

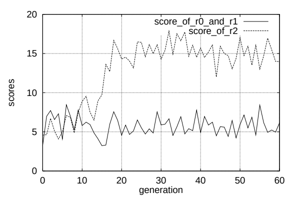
Finally, we test the co-evolution among three robots. That is, **r2** added in Section 4.2 evolves its behavior with **r0** and **r1** simultaneously. The difference from Sections 4.1 and 4.2 is involvement of competition between **r2** and a team of **r0** and **r1**. The number of function sets is the same as the case of Section 4.2.

Table 3 (c) shows the tree depths and the numbers of the nodes. The acquired *GP* tree of **r1** tends to be simple as compared with the cases of **r0** and **r2**. Furthermore, the results based on the varying fitness function is more complicated than those of the fixed one. However, the varying fitness function leads the *GP* agents to the local solutions as follows.

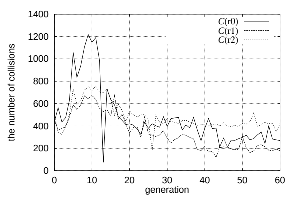
The results are shown in Figure 7. As compared with the only cooperative tasks in Section 4.2, fitness values rather oscillate than converge stably. Although $C(i)$ and *steps* decrease gradually through the evolution in a case of the varying fitness function, this game is dominated by **r2** at the 18th generation. This phenomenon is observed when we use the fixed fitness function, but the acquired performances of **r0** and **r1** are a little bit better than the a case of the varying fitness function.

We can see two typical settlements in this three-robot soccer game. One is the same behaviors described in Section 4.2 : **r0** kicks the ball toward **r1**, then **r1** shoots the ball into the goal avoiding collisions with **r2** (See Figure 8). The other one is that **r2** intercepts the ball and shoots the ball into the goal (See Figure 9). The ratio between the former and the latter is about 25 % : 75 %. It depends on whether **r0** or **r2** achieves its goal. However, **r2** can observe the ball and the opponent's goal at the same time and it may shoot the ball by itself while **r0** needs to pass the ball to **r1**. We suppose that the predominance of **r2** may be caused by the different complexity of the given tasks, that is, task complexities for **r0** and **r1** is higher than that for **r2**.

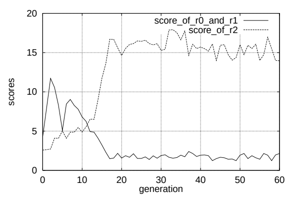
Finally, we analyze the results of the acquired trees. As we have no space to represent the whole tree, we show a part of the tree in Figure 10 (the box and the ellipse represent a conditional branching function and a terminal, respectively). As you can see from these



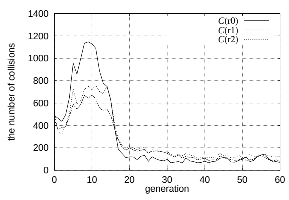
(a) $G(i)$ and $L(i)$



(b) $C(i)$



(c) $G(i)$ and $L(i)$



(d) $C(i)$

Figure 7: Experimental results in a case of three learners (a, b:fixed, c, d:varying)

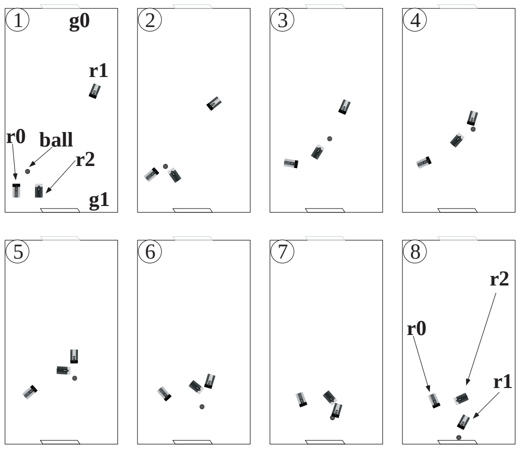


Figure 8: Two robots (**r0** and **r1**) succeed in shooting a ball into the goal against the defender (**r2**)

figures,

- **r2** : When **r2** won the game, offensive behavior (Figure 10(c)) was executed in many situations. We suppose that the reason why this simple tree can realize purpose behaviors is that the primitive behaviors such as `shoot` and `avoid` are obtained in similar situations [2]. It helped the *GP* to acquire the appropriate tree in a compact style.
- **r0** : `pass` behavior was selected in Figure 10 (a) when **r0** succeeded in passing a ball to **r1**. In addition, **r0** searched the ball and selected a `shoot` behavior if **r0** could not observe **r1**. The reason is that `pass` behavior are also acquired in a similar situation[3].

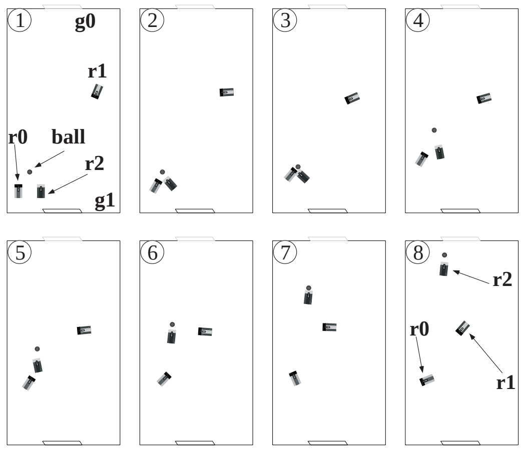


Figure 9: The defender (**r2**) succeeds in shooting a ball into the goal against the two robots (**r0** and **r1**)

However, the tree which **r1** acquired is ineffective. In other words, the primitive behaviors was not appropriate for **r1** to utilize. In order to acquire compact representation, we have to modify the primitive behaviors based on the tree structure.

5 Discussion and Future Works

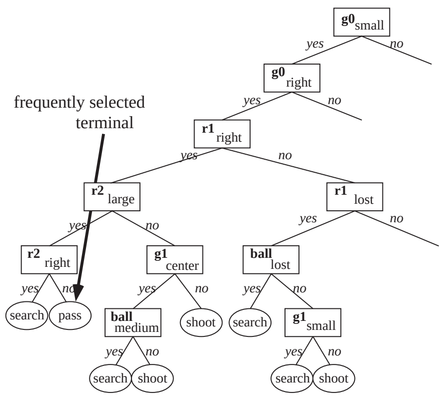
This paper showed how co-evolution technique could emerge not only competitive behaviors but also cooperative ones through a series of experiments in which two or three robots play a simplified soccer game. In order to co-evolve cooperative agents, it should be noted that robots must synchronize their evolutionary processes. Otherwise, there are many traps to local maxima (suboptimal strategies) as we can see in Section 4.1.

In a case of the most complicated situation (three agents and both cooperation and competition are involved), the task complexity should be equal to all agents so as to co-evolve cooperative and competitive agents simultaneously. This also suggests that the environment itself should co-evolve from simpler to more complicated situations to assist the development of desired skills of cooperations and competitions. Otherwise, co-evolution is prone to be settled into suboptimal strategies as shown in Section 4.3.

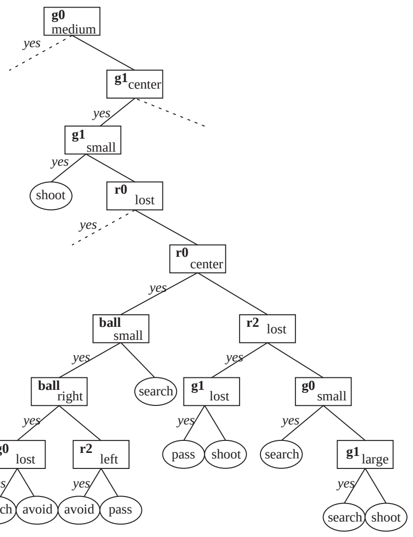
In the current system, state quantization and terminal actions were fixed in advance. Through the evolution process, the agent obtains its decision tree which tells how visual features are organized and connected to one terminal action. A different application of *GP* can be

[2]As described before, the terminals are obtained by the reinforcement learning. For more detailed settings, see [10].

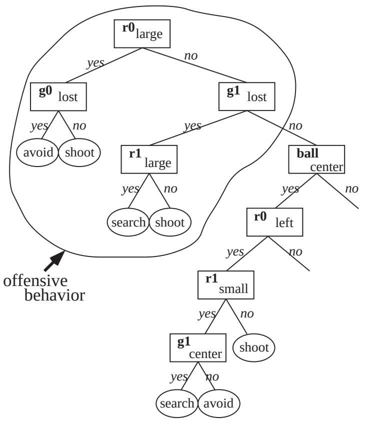
[3]In [10], **r0** learns to pass the ball to **r1** in Figure 2 based on the reinforcement learning.



(a) **r0**'s tree



(b) **r1**'s tree



(c) **r2**'s tree

Figure 10: A part of the acquired tree

considered to extract motor outputs by a feature tree in which sensory inputs are combined and calculated. Design issues of environments including agents, tasks, and fitness functions are our future work. Also, we are planning to implement real experiments.

Acknowledgments

This research was supported by the Japan Society for the Promotion of Science, in Research for the Future Program titled Cooperative Distributed Vision for Dynamic Three Dimensional Scene Understanding (JSPS-RFTF96P00501).

References

[1] http://www.robocup.org/.

[2] D. Cliff and G. F. Miller. Co-evolution of Pursuit and Evasion II : Simulation Methods and Results. In *Proc. of the 4th International Conference on Simulation of Adaptive Behavior: From Animals to Animats 4.*, pp. 506–515, 1996.

[3] J. H. Connel and S. Mahadevan. *Robot Learning.* Kluwer Academic Publishers, 1993.

[4] D. Floreano and S. Nolfi. Adaptive Behavior in Competeing Co-Evolving Species. In *Fourth European Conference on Artificial Life (ECAL97)*, pp. 378–387, 1997.

[5] H. Kitano, ed. *RoboCup-97 : Robot Soccer World Cup I.* Springer Verlag, 1997.

[6] J. R. Koza. *Genetic Programming I : On the Programming of Computers by Means of Natural Selection.* MIT Press, 1992.

[7] J. R. Koza. *Genetic Programming II : Automatic Discovery of Reusable SubPrograms.* MIT Press, 1994.

[8] S. Luke, C. Hohn, J. Farris, G. Jackson, and J. Hendler. Co-Evolving Soccer Softbot Team Coordination with Genetic Programming. In *Proc. of the RoboCup-97 Workshop at the 15th International Joint Conference on Artificial Intelligence (IJCAI97)*, pp. 115–118, 1997.

[9] P. Stone and M. Veloso. Team-Partitioned, Opaque-Transition Reinforcement Learning. In M. Asada ed., *Proceedings of the second RoboCup Workshop*, pp. 221–235, 1998.

[10] E. Uchibe, M. Asada, and K. Hosoda. State Space Construction for Behavior Acquisition in Multi Agent Environments with Vision and Action. In *Proc. of International Conference on Computer Vision*, pp. 870–875, 1998.

[11] E. Uchibe, M. Nakamura, and M. Asada. Co-evolution for Cooperative Behavior Acquisition in a Multiple Mobile Robot Environment. In *Proc. of the 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 425–430, 1998.