Proceedings of the 14th KACC, Octorber 1999

Learning soccer robot using genetic programming

Xiaoshu Wang* and Masanori Sugisaka**
Dept. of Electrical and Electronic Engineering
Oita University, 700 Dannahara,Oita 870-1192, Japan*
Tel: +81-97-5547831; Fax: +81-97-5547841
{wangxs*,msugi}@cc.oita-u.ac.jp

ABSTRACT

Evolving in artificial agent is an extremely difficult problem, but on the other hand, a challenging task. At present the studies mainly centered on single agent learning problem. In our case, we use simulated soccer to investigate multi-agent cooperative learning. Consider the fundamental differences in learning mechanism, existing reinforcement learning algorithms can be roughly classified into two types-that based on evaluation functions and that of searching policy space directly. Genetic Programming developed from Genetic Algorithms is one of the most well known approaches belonging to the latter. In this paper, we give detailed algorithm description as well as data construction that are necessary for learning single agent strategies at first. In following step moreover, we will extend developed methods into multiple robot domains. game. We investigate and contrast two different methods-simple team learning and sub-group learning and conclude the paper with some experimental results.

Keywords: Genetic Algorithms, Genetic Programming, Multi-agent system, Machine Learning

1 INTRODUCATION

Learning is the problem faced by an agent of how to obtain optimal problem-resolving policy through trial-and-error interactions with a dynamic environment. Consider the basic differences in learning mechanism, we can roughly classify existing reinforcement learning (RL) algorithms into two types. The first includes those that base on adaptive Evaluation Functions (EFs) such as Q-learning presented by Watkins in 1989 and all kinds of variants of it. To the contrary, methods from the second class do not require EFs. They search through the policy space directly to obtain optimal policies eventually. One of the most widely known EFs-free approaches is Genetic Programming(GP) proposed by Koza[1] in 1992. For explanation in more detail, a survey of RL written by Leslie[2] et al is a good reference, I think. Originally both of them were introduced in order to solve single agent learning problems. But recently multi-agent learning received more and more attentions. In our case, we use simulated soccer to investigate multi-agent cooperative learning. In order to extend existing approaches into multi-agent domains, following two thoughts are obvious. The first is to handle each agent in multi-agent domain just with existing single agent RL algorithms while other agents are treated as parts of environment. The another method is to treat all agents as an indivisible unity. Then some changes must be made to make existing strategies suitable to deal with new conditions.

2 MACHINE LEARNING

At first let's consider problems of machine learning in a general way. *Markov decision processes* (MDPs) define a standard model for it as following.

1) a set of states ***S***,
2) a set of actions ***A***,
3) a reward(or cost) function ***R: S×A □ IR***,
4) a state transition function ***T: S×A □□(S)***. where□is a discrete probability distribution over set ***S***.

Consider a MDP model shown above, the goal of learning becomes to search optimal policy to maximize expected reward (or, minimize total expected discounted cost). Let's Denote policy as □, then the optimal polity based on MDPs model can be described as

$$V^*(s) = \max_a (R(s,a) + \gamma \sum_{s'\in S} T(s,a,s')V^*(s')), \forall a \in A$$

$$\pi^*(s) = \arg V^*(s),$$

where R is the instantaneous reward, s' resulting state, and $T(s,a,s')$ the probability of taking action a in state s. V is evaluation function of a state. In other words, it is a *value* that agent can gain if it starts in state s and executes a policy □. Now the problem of searching optimal policy becomes how to define an easy-to-computing V^* function. A conceptually simple approach to solve the problem is Q-learning introduced by Watkins in 1989. He define

$$V^*(s) = \max_a Q^*(s,a), \forall a \in A,$$

Where $Q^*(s,a)$ is the expected reward (or discounted gain) of taking action a in state s, which is defined as

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') \max_{a'} Q^*(s',a')),$$

Where

$$Q(s,a) := Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a)).$$

Then $Q(s,a)$ can be computed in recursive. But we can find that in order to use Q-learning, knowledge about system's state transition probability $T(s,a,s')$ and reinforcement function $R(s,a)$ are essential. But to most system, especially multi-agent system, that can not be modeled well, it is still very difficult to get them.

By contrast with EF-based learning briefly explained above, recently studies of how to search optimal policy in policy place directly develop quickly. Among them Genetic Programming(GP) proposed by Koza in 1992 is most well known. In fact GP is a branch of genetic algorithms where the solution strings are replaced with variable length programs. Sugisaka[3] and Wang[4] show basic thoughts of how to apply genetic algorithms to robot control in some paper before. The goal of GP is to find a program that can be run so as to solve defined problem. Based on MDPs model above, a program in fact stands for a policyπ. Then the problem of learning becomes that of how to evolve programs so as to satisfactory results (optimal policies) can be found out finally. From now in this paper PROG denotes a genetic program. The genetic program is evaluated for fitness by executing PROG.

In usual PROGs are decoded in format of *n-ary* tree with n being the maximal number of sub-trees that a non-leaf node can contain. A non-leaf node encodes a function f_i. A leaf node stands for an action, which can be executed by the agent. So in order to initialize PROG population, a function set $F=\{f_1,f_2,\ldots,f_k\}$ with k functions and a terminal set $T=\{t_1,t_2,\ldots,t_l\}$ with l terminals must be decided definitely in advance. A most simple example is quadratic equation analysis problem with $F=\{+,-,*,\%,\square\}$ and $T=\{2,a,b,c\}$. GP follows same steps as GAs but the steps act a little differently in order to make them reasonable to the new tree-type structure. The best PROG for the problem will be that shown in figure 1. The flow of GP operation is shown in figure 2.

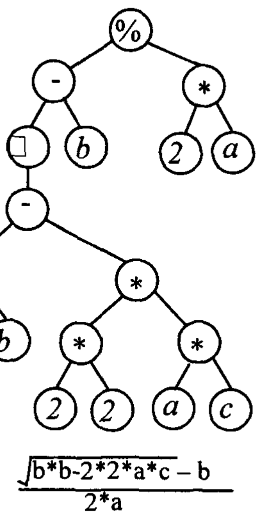

Figure 1. One of the best PROG for quadratic equation problem

```
Initialize the Population of PROGs arbitrarily
Loop until termination criterion satisfied or timeover
Evaluate PROGs in population
Loop until next generation population created
Select genetic operation probabilistically
Perform Reproduction
Perform Crossover
Perform Mutation
End loop
End loop
```

Figure 2. Genetic programming flowchart

Given the MDPs model, the set of action A can be used as terminal set T directly. But the set of function must be designed carefully so as to it can observe environment well. In this paper, we will use GP to solve robot soccer learning problem.

3 LEARNING ROBOT SOCCER BY GP

For many years robot game playing has been a popular problem area for studies in artificial intelligent and machine learning, for example those conducted by John WS[5] and Ho F[6] et al. In our work we use simulated soccer to study intelligent multi-agent system. The reasons for us choosing such a system as experiment bench are because (1) it is a standard multiple agent system suitable to be studied, (2) it is of flexibility. In other words we can expand or contract the scale of system easily to meet our needs. (3) it is ideal to tell whether an approach better than other or not since it is a kind of antagonistic game played by two teams. Initially our researches in such game are limited to constructing fixed strategies. But with the number of robot players increasing, huge complexity of state space makes it impossible for us to consider all conditions the agent may encounter in a practical playing. That is to say, pre-planned behaviors are not adaptive to practical environment. Efforts for solving this problem bring about ideas of learning robot soccer game strategies. Especially we put our emphasis on how to co-learn multiple agents simultaneously.

3.1 Robot soccer

Robot soccer game, denoted as *G{ Field(Size(l,w), CoordinateSystem), ball, $Team_{home}$, $Team_{Opp}$, Rule}*, is played by two robot teams, $Team_{home}$ and $Team_{Opp}$. Each team is composed of *n* robots, that is, $Team=\{P_1,...,P_n\}$ $n>1$. The game rule, *Rule,* comes from human being soccer version. There are several kinds of methods to organize an system to attend contest, such as (1) simulation game in computer, (2) micro-robot system with vision-based concentrated control, (3) simple robot system with autonomous and distributed control, (4) humanoid robot system and so on. In our case we conduct simulations in computer at first. Then we test obtained optimal strategies in vision-based micro-robot system.

The ball is described as $ball_t\{b(x,y)$, *direction, Velocity*}. *b(x,y)* is center coordinates of the ball. The ball moves with *velocity* along orientation of *direction.* A player also contains all of three variables of $P_i\,(x,y)$ and *direction velocity,* and moreover, the variable of *Size(l,w)* is included too. For vision-based centralized system, a Boolean variable for each ball and each player is added since in some time the ball or player can not be identified. At any sampling time point, a player can obtain information *i(p,t)* of present environment. *i(p,t)* includes 1) $ball_t$, 2) $Team_{home}$ and $Team_{Opp}$ 3) home players' behaviors in last sampling period. Of course the information of *Field* is also knowable since it is treated as global constant. With the number of players increasing, the environment information *i(p,t)* will become larger and larger. Processing of *i(p,t)* will become very time-consuming. In this case, we can just consider the information belonging to the round with the radius of *r* and center coordinates of *p(x,y)*, that is, it assumes that the behavior of a player mainly is affected by its circumference in nearby.

3.2 Learning Single robot strategy

We start doing experiment from single robot learning, that is, there are neither partners nor opponents. A simulated robot in the play field can execute any action from action set *A*. An example of set A is shown in table 1.

Table 1. An example of Action set.

A_1	*go-forward*	Move player *p* one step forward along its current direction, *(one step=Field.size.length/100)*
A_2	*go-backward*	Move player *p* one step opposite to its current direction
A_3	*drip-ball:*	If player have the ball, it will move with ball in one step. Otherwise it will execute action *go-forward.*
A_4	*kick-ball*	If player have the ball, it will move execute *go-forward* and at same time the ball obtain energy *E.* the speed of ball will be adjusted. Otherwise player just execute action *go-forward.*
A_5	*Trun-to-ball*	Change player's direction so that it directs to ball
A_6	*Turn-to-goal*	or opponent goal
A_7	*Turn-to-there*	or position(x,y).
A_8	*noise-shoot*	If player does not have the ball then do nothing. Otherwise, execute *turn(noise)* at first and then kick-ball. (This action is added in order to mimic the practical soccer game).

Except those actions listed in table 1, we also try some other actions, such as *avoid-obstacle* and *shoot-to-goal,* when we conducted experiment. More actions was included, larger the PROG would became. But from practical experiment results we can not find that better performances appear if we just add more actions into action set.

In our simulations, we design all members *f*, but a special function Proc2, in the function set *F* as two-branch conditional statement of *IF-Then.* In a condition, the left node is executed if the condition is TRUE, otherwise the right node is executed. Proc2 is not a condition. Both children nodes of it will be executed. Here we don't want to list all members included in the function set *F* since we continue trying new functions while doing experiment. We give some functions used in table 2.

Table 2 An example of function set.

F_1	**Proc2** (Note:a special function, both of its children nodes are executed.)
F_2	**IF** ball in hand **THEN**
F_3	**IF** obstacle ahead **THEN**
F_4	**IF** ball is near to opponent goal **THEN**
F_5	**IF** ball is near to home goal **THEN**
F_6	**IF** ball is between player with opponent goal **THEN**
F_7	**IF** ball is between player with home goal **THEN**

Based on the GP flowchart shown in figure 2. GP algorithm runs as following.

1. To Initialize the Population

In the case of GP, to initialize a population means to generate n 2-ary trees. Normally a 2-ary tree can be created by recursive algorithm. That is, at first a member from set ***AUF*** is picked out randomly and inserted in current position. If selected member belongs to set *F* then left sub-tree and right sub-tree of it are created separately. In our case, the probability of selecting each member is same. So this method will work only if there are more than twice as many *Actions* as *Functions*. Otherwise the tree maybe become infinitely big. By the way, we keep the depth of the tree below 10. An example of generated tree is shown in figure 3.

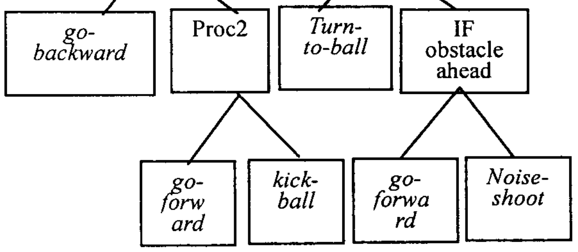

Figure 3. An example of PROG tree.

2. To evaluate PROGs in population.

In our case, we execute all PROGs in current population in order to evaluate them. One entire soccer game lasts 1 minute. When game is over, every PROG is scored with a scalar, non-negative fitness value, *Fitness(PROGi)=number of goals scored by PROGs.*

3. To perform GP operations(Reproduction, Crossover, Mutation)

Reproduction operation runs the same for a GP as a GAs. The best m PROGs in current population are reproduced in order to replace the worst m PROGs. But the crossover and mutation operation must be re-designed to be suitable for tree-type genome. For crossover operation, at first a crossover node in each parent PROG tree must be selected. Unlike GAs, the crossover points in both parents do not have to be the same. For mutation operation, the leaf nodes and non-leaf nodes are not interchangeable. A non-leaf node of *action member* obviously leads grammar error.

GP is a general problem-solving method. Basic operations of it are independent to practical problems. In order to re-use the programming codes, GP can be broken down into problem-dependent and problem-independent parts with a carefully designed interface between these two parts. Usually the interface is a kind of structured input/output files. Some general GP modules, for example the GP kernel developed by Helmut H[7] at Vienna university, can be downloaded for research. In our laboratory, we developed our own GP VBX using C++.

3.3□Learning multiple robots strategy

Robot soccer game is a kind of multiple robot tournament. In our case we try (3 by 3) and (5 by 5) games. Obviously we can extend single agent GP learning into multi-agent domain by 2 ways. The simplest method named as simple team learning is to learn all home player strategies separately where partners in the play field are treated as part of environment. Except those basic information such as position and speed, any player can not know what his partner had done and what he want to do in next step. Because now 6/10 robots are involved in the game, collisions between robots become so frequent that robots always gather around the ball and the game can not continue. So we have to add collision-examining functions into F and collision-avoiding actions into A.

Another approach named as sub-group learning is to treat all home players as an indivisible group or to divide them into several sub-groups. For learning team/sub-group strategies, some changes must be made on existing single agent learning algorithms. Now the actions in set A ought to could control not only single robot but also several robots simultaneously.

For example new action *pass-ball* will let one robot kick ball as well as move another robot to suitable position so that it can get the ball.

Because opponents are involved in the game, the fitness function becomes *Fitness(PROGi)=100+number of goals scored by PROGs-number of goals scored by opponent*. The offset 100 ensure the fitness always positive.

3.4□Experiments and results

We conduct three different types of simulations-single robot, 3 by 3 and 5 by 5 using single-agent learning, simple team learning and sub-group learning separately. The parameters for GP in the simulations are listed in Table 3. The opponent team is driven by our old version of pre-programming strategy.

Table 3. Parameters.

Team	1	3 by 3	5 by 5
Population size	100	50	50
Generation	2000	2000	2000
Maximum depth	10	10	10
Size of set *A*	10	14	16
Size of set *F*	4	6	7
crossover rate	0.2	0.2	0.2
mutation rate	0.05	0.05	0.05

Some experiment results are given in Figure 4.From results above, we can find single robot learning works quite well. In the 1000th generation, the satisfied policy can be generated. But in 3*3 game, home team can not win the game at all. So it seems that the simple team learning can not automatically generate cooperative behaviors among agents. In 5 by 5 game, because the cooperation behaviors are considered in advance when we designed the action set and function set. The performances of learning become better, but it is not optimal result, I think.

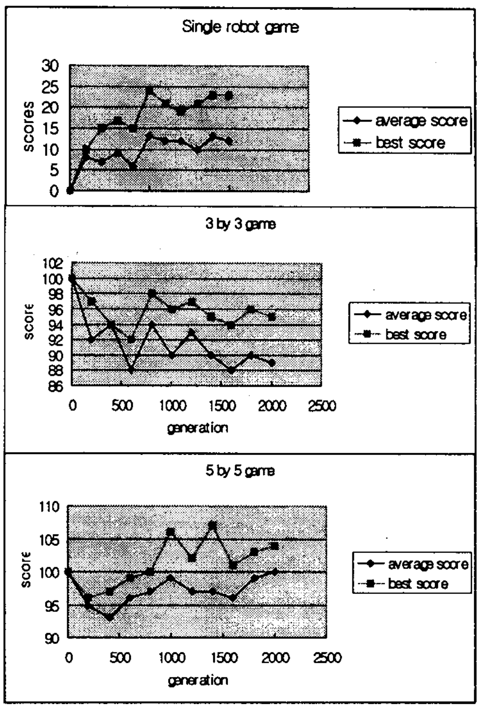

Figure 4 Results, from above
a)single robot game b)3x3 game c)5x5 game

4. CONCLUSION

We use simulated soccer game to study machine learning and multi-agent learning. In this paper, we compared single agent learning, simple team learning and sub-group learning based on GP. From results we know that single agent learning by GP works well. But in simple team learning case, expected cooperative behaviors among home players did not appear at all. In 5 by 5 game, we divide the home team into 3 sub-group of {2(attacker),2(defender),1(goalkeeper)}. With carefully designed action set A, the sub-group learning works better than simple team learning. But some problems still exist. For example if the number of player increases, the designing of action set becomes very awkward. In our next works we will 1) improving sub-group learning algorithms and 2) try EF-based RL algorithms and compare it with EF-free approaches

References

1. Koza JR (1992) Genetic programming-on the programming of computers by the natural selection. Cambridge, MA, MIT Press

2. Leslie PK and Michael LL (1996) A survey of reinforcement learning. Journal of Artificial Intelligence Research 4:237-285
3. Sugisaka M, Wang XS, Lee JJ (1998) The genetic algorithms to evolve multiple agent cooperative system. Proceedings of the International Symposium on Artificial Life and Robotics (AROB3), Beppu, Oita, Japan, Jan 19-21,1998, pp.178-182
4. Sugisaka M, Wang XS (1997) The autonomous evolution of cooperation activity of micro-robots by genetic algorithms(GAs). Proceedings of the 29th ISCIE International Symposium on Stochastic system and its Applications, Tokyo, Japan, Nov 10-12, 1997, pp.29-32
5. John WS (1998) Colearning in different games. Machine Learning 33:201-233
6. Ho F, Kamel M (1998) Learning coordination strategies for cooperative multiagent systems. Machine Learning 33:155-177
7. Helmut H (1996) A C++ class library for genetic programming: the Vienna university of economics genetic programming kernel-Operating Instruction, 1:1-69
8. Guy C, George B, Brigitte DN (1996) Structure properties and classification of kinematics and dynamic models of wheeled mobile robot. IEEE transactions on robotics and automation12(1) : 47-61