GENETIC PROGRAMMING OF MULTI-AGENT COOPERATION STRATEGIES FOR TABLE TRANSPORT

Dong-Yeon Cho[a], Byoung-Tak Zhang[b]

a. Dept. of Computer Engineering, Seoul National University
   Sinlim-Dong, Gwanak-Ku, Seoul 151-742 Korea
   Tel: +82-2-880-7302 Fax: +82-2-880-7302 E-mail: dycho@nova.snu.ac.kr
b. Dept. of Computer Engineering, Seoul National University
   Sinlim-Dong, Gwanak-Ku, Seoul 151-742 Korea
   Tel: +82-2-880-1833 Fax: +82-2-883-3595 E-mail: btzhang@scai.snu.ac.kr

Abstract

Transporting a large table using multiple robotic agents requires at least two group behaviors of homing and herding which are to be coordinated in a proper sequence. Existing GP methods for multi-agent learning are not practical enough to find an optimal solution in this domain. To evolve this kind of complex cooperative behavior we use a novel method called fitness switching. This method maintains a pool of basis fitness functions each of which corresponds to a primitive group behavior. The basis functions are then progressively combined into more complex fitness functions to co-evolve more complex behaviors. The performance of the presented method is compared with that of two conventional methods. Experimental results show that coevolutionary fitness switching provides an effective mechanism for evolving complex emergent behaviors which may not be solved by simple genetic programming.

Keywords: genetic programming, multiagent learning, artificial life, autonomous robots, fitness switching.

1. Introduction

Genetic programming has been used to evolve cooperative behaviors of a group of simple robotic agents. Koza and Bennett [4, 1] used genetic programming to evolve a common program that causes foraging of foods by an ant colony. Haynes et al. [2] showed that programs for solving a predator-prey problem can be generated by genetic programming without any deep domain knowledge. Luke et al. [6] explored various strategies for evolving teams of agents in the Serengeti world, a simple predator-prey enrvironment. Iba [3] studied three different breeding strategies (homogeneous, heterogeneous, and coevolutionay) for cooperative robot navigation.

These studies have attempted to evolve emergent collective behaviors immediately from primitive actions. However, more realistic complex tasks require more than one emergent behaviors and a proper coordination of these is essential for successful accomplishment of the task.

In this paper we study a table transportation problem, an example of important multi-robot applications [11]. This problem requires at least two emergent behaviors, i.e. homing and herding, to be achieved in sequence. The robots need first to get together around the object to transport (homing) and then transport it in team to the destination (herding). In this task, a group of robot agents "must" cooperate to ever achieve the goal since the table is too big to be transported by a few robots.

A successful solution to this problem using genetic programming requires a coordination of homing and herding behaviors in some way. A straightforward genetic programming method may not scale up to this problem since the agents are engaged in a strictly coordinated sequential task

We present in this paper some results of using a novel method called *fitness switching* [13]. In fitness switching, different parts of a genetic tree are responsible for different behaviors and for each of the subtrees a basis fitness function is defined. The evolution of the entire behavior is scheduled by a fitness switch that dynamically changes fitness types in a pool of fitness functions. The basic idea behind this approach is that fitness functions are a fundamental mechanism that guides the evolutionary process. An advantage of this approach is that it is easy to implement the progressive learning, i.e. learning easier tasks first and then harder tasks, which is a well-proven educational method in pedagogy.

It should be noted that our approach is different from other heterogeneous breeding methods [6, 3] in which different subtrees represent different *agents*. In the fitness
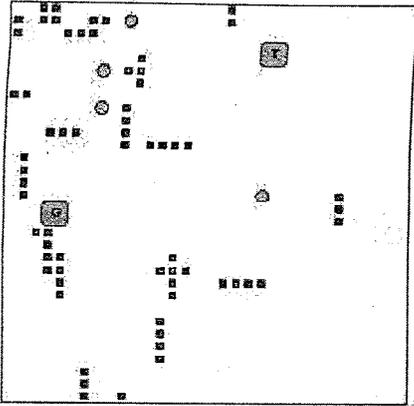
Fig. 1 The environment for the transpotation problem

Fig. 2 The framework for fitness switching

switching method, different subtrees represent different *behaviors* of a single agent which need to be coordinated.

The paper is organized as follows. Section 2 describes the task and related work on multiagent learning. Section 3 presents the general framework for fitness switching and its implementation methods for the evolution of complex group behaviors using genetic programming. Section 4 shows experimental results of the presented methods. Section 5 discusses the result and suggests further work.

2. The Task

The transportation problem we consider in this paper is defined as follows. In an n×n grid world, a single table and four robotic agents are placed at random positions, as shown in Figure 1. In addition, a specific location is designated as the destination. The goal of the robots is to transport the table to the destination in group motion. The robots need to move in herd since the table is too heavy and large to be transported by single robots.

Each robot has a repertoire of actions. It can move forward in the current direction (N, E, S, W, NE, SE, SW, NW) or remain on the current position. The direction of the movement can be chosen randomly to avoid collision with obstacles or other robots. The robots have a limited visual field of range 1 to each movable direction. A fixed number of obstacles are placed in the grid. The robots can recognize other robots and distinguish them from obstacles.

Each robot i ($i = 1, .., N_{robots}$) is equipped with a control program A_i. If $A_i \neq A_j$ for $i \neq j$, then control programs are said to be *private*. In case of *public* control programs, all instances of A_i are constrained to be the same A.

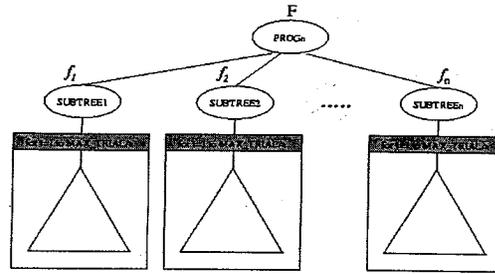The robots activate A_i's in parallel to run a team trial.

At the beginning of the trial, the robot locations are chosen at random in the arena. They have different positions and orientations. During a trial, the robots are granted a total of S_{max} elementary movements. The robot is allowed to stop in less than S_{max} steps if it reaches the goal. At the end of the trial, each robot i gets a fitness value which was measured by summing the contributions from various factors. The goal of genetic programming is to find control programs leading to efficiently transporting the table from the initial position to the goal position.

3. The Method

We use genetic programming to evolve the cooperation strategies of the transport robots. As discussed in the introduction, conventional genetic programming is not powerful enough to solve this problem. Thus, we have used a novel method called fitness switching [13]. It is a method for evolving complex behaviors with genetic programming. The procedure is based on incremental learning as described below (see Figure 2):

1. Define primitive actions for the problem domain.
2. Define a small number of micro-behaviors,
$$B = \{B_1, B_2, ..., B_n\} \qquad (1)$$
   that constitute the original problem-solving behavior.
3. Define a fitness function for each micro-behavior. This makes the pool of fitness function.
$$F = \{f_1, f_2, ..., f_n\} \qquad (2)$$
4. Design a sequence of micro-behaviors of their combinations to achieve the target behavior:
$$S_t = S_{t-1} \cup \{B_t\} \qquad (3)$$
   Where $t = 0, ..., n$ and $S_0 = \{\}$. The corresponding sequence of fitness function are defined as

$$F_t = F_{t-1} + f_t = \sum_{i=1}^{t} f_i \qquad (4)$$

Where $t = 0, \ldots, n$ and $F_0 = 0$.

5. Define the structure of a genetic program as having s subtree immediately under the root node.

6. Apply genetic programming to evolve S_t, $t = 1, \ldots, n$ in sequence. For each S_t, the fitness function F_t is used to evolve the first t subtrees of the entire tree.

The method is called fitness switching since evolution is guided by fitness functions switched from simpler ones to more complex ones.

In the following, we illustrate the fitness switching method applied to the table transportation problem. As described in Section 2, the transportation problem can be considered as a composition of the following cooperative behaviors:

- homing
- herding

Thus the set of micro-behaviors is $B = \{B_1, B_2\}$, where B_1 = *homing* and B_2 = *herding*. The set of fitness functions is $F = \{f_1, f_2\}$. Here f_1 is the fitness function for the homing behavior to the table and the f_2 is the herding behavior for transporting the table to the goal.

In the experiments we have used the following fitness functions for f_1 and f_2:

$$f_1 = \sum_{r=1}^{4} \{c_1 \max(X_r, Y_r) + c_2 S_r \qquad (5)$$
$$+ c_3 C_r - c_4 M_r + K\}$$

$$f_2 = \sum_{r=1}^{4} \{c_1 \max(X_r, Y_r) + c_2 S_r \qquad (6)$$
$$+ c_3 C_r - c_4 M_r + c_5 A_r + K\}$$

the definitions of the symbols used in above equations are provided in Table 1. The target position for homing behavior is the initial position of the table while the target position for herding behavior is the destination of the table.

Several implementational variants for fitness switching are possible. One simple choice is *naïve evolution* in which genetic programs are initialized with arbitrary structures which are shared for all the micro-behaviors. In each generation, the fitness of each program is measured as follows:

1. Measure the fitness of the *whole* tree by f_1.
2. Measure the fitness of the *whole* tree by f_2.
3. The fitness of the program is defined as $F = f_1 + f_2$.

Naive evolution is one extreme on which most existing GP studies are based. This method is very efficient in memory usage since the same tree is shared by multiple

Table 1: Symbols used for fitness definition.

Symbol	Description
X_r	x-axis distance between target and robot r
Y_r	y-axis distance between target and robot r
S_r	number of steps moved by robot r
C_r	number of collisions made by robot r
M_r	distance between starting and final position of r
A_r	penalty for moving away from other robots
c_i	coefficient for factor i
K	positive constant

behaviors. A disadvantage is that this representation is difficult to coordinate multiple cooperative behaviors.

Another extreme of fitness switching is *sequential evolution*. Here the left subtree is responsible for homing and the right subtree for herding. The left subtrees for homing behavior are evolved by a GP run and then the best program for this run is used to evolve the next GP run for evolving the herding behavior. The process is summarized as follows:

1. Run a GP to evolve left *subtrees* for homing behavior by measuring fitness by f1.

2. Let A_{best}^L be the best individual evolved above.

3. Run another GP to evolve right *subtrees* for herding behavior by using A_{best}^L and measuring the fitness of the whole tree A by $F = f_1 + f_2$.

This is another extreme in which the coordination is hard-coded both in representation and in evolutionary process. This approach seems the most practical in solving tasks which can be clearly decomposed into a sequence of independent subtasks. But most of interesting problems that need emergent computations do not belong to this class of problems

We choose a third option, fitness switching with *coevolution*. The coevolutionary switching is similar to the sequential evolution in that the subtrees are responsible for different micro-behaviors. The difference lies in the fact that fitness measures are switched within a single generation, which has some similarity to the naive evolution. Fitness of programs is measured at each generation as follows:

1. Measure the fitness of the left *subtree* by f_1.
3. Measure the fitness of the right *subtree* tree by f_2.
3. The fitness of the program is defined as $F = f_1 + f_2$.

The advantage of this method is the ability of concurrent evolution of primitive cooperative behaviors

Table 2 Tableau for the table transportaion problem

Prameter	Value
Terminal set	FORWARD, AVOID, RANDOM-MOVE, TURN-TABLE, TRUN-GOAL, STOP
Function set	IF-GOAL, IF-ROBOT, IF-TABLE, IF-OBSTACLE, PROG2, PROG3
Fitness cases	20 training worlds, 20 test worlds
Robot world	32 by 32 grid, 64 obstacles, 1 table to transport
Population size	100
Max generation	200
Crossover rate	1.0
Mutation rate	0.1
Max tree depth	10
Selection scheme	truncation selection with elitism

and their coordination.

Based on our previous work [12] a complexity term was used in all experiments to penalize large trees:

$$F = F + \beta C \qquad (7)$$

where C is the number of nodes in the tree and β is a small constant.

4. Results

Table 2 summarizes the experimental setup for genetic programming. The objective of a GP run is to find a multi-robot algorithm that, when executed by each robot in a group of 4 robots, causes efficient table transport behavior in group.

We have used the function set consisting of six primitives: IF-OBSTACLE, IF-ROBOT, IF-TABLE, IF-GOAL, PROG2 and PROG3. IF-OBSTACLE and IF-ROBOT check collisions with obstacles and other robots, respectively. IF-TABLE and IF-GOAL are used to detect the table and the goal position. PROG2 (PROG3) evaluates two (three) subtrees in sequence.

The terminal set consists of six primitive actions: FORWARD, AVOID, RANDOM-MOVE, TURN-TABLE, TURN-GOAL and STOP. FORWARD takes one step forward in the current direction. This movement can cause collision. AVOID checks the surrounding region and makes one step in the first direction that avoids the collision. The checking takes place clockwise starting from current direction. RANDOM-MOVE makes a random movement in any direction. This can cause collision. TURN-TABLE and TURN-GOAL make a clockwise turn to the nearest direction of the table and the destination, respectively. STOP makes no step and remains the same position. An example of genetic program is
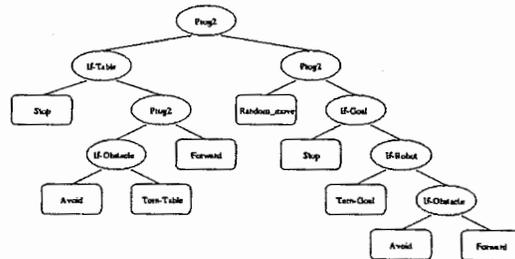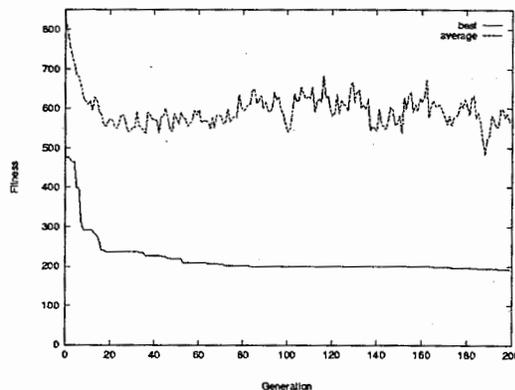

Fig. 3 A genetic program for solving the table

Fig. 4 Evolution of fitness values during a GP run

shown in Figure 3.

Each fitness case represents a world of 32 by 32 grid on which there are four robots, 64 obstacles, a table to betransported. A total of 20 training cases are used for transportation problemevolving the programs. A total of 20 independent worlds are used for evaluating the generalization performance of evolved programs.

Figure 4 shows the change in fitness values during a GP run with coevolutionary fitness switching: The fitness of a tree A is measured by $F(A) = f_1(A) + f_2(A)$, where $f_1(A)$ is the fitness for homing and $f_2(A)$ is the fitness for herding. A rapid decrease in fitness indicates the fast improvement in cooperative behavior.

The genetic programming with coevolutionary fitness switching was able to learn to solve the transportation problem for more than one environments. Figure 5 shows the behavior of the robots to the training environments. Shown are four cases out of 20 training cases in total.

The generality of the evolved programs was verified by running them on test environments. Figure 6 shows the behaviors of the robots to the test cases. Shown are also four cases out of 20 test cases in total. Comparison of Figures 5 and 6 suggest effectiveness of fitness switching with coevolution as a method for evolving composite cooperative behaviors.
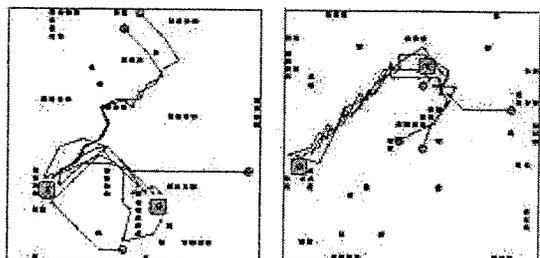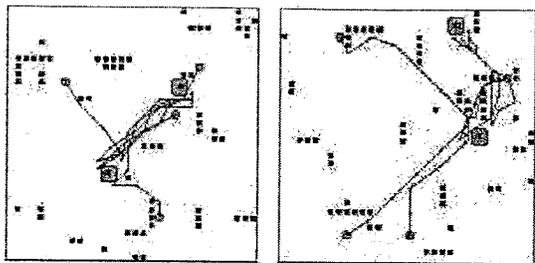
The performance of genetic programs can also be

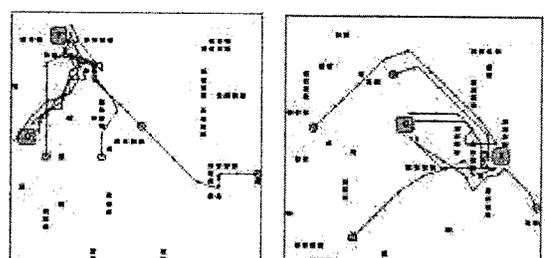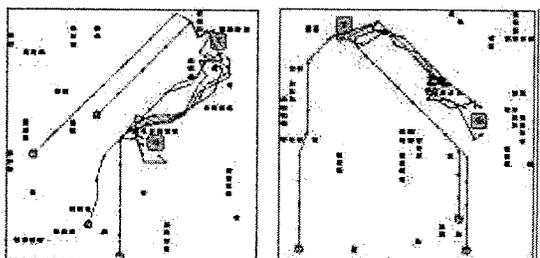Fig. 5 Trajectory of robots running the evolved program in the training cases

Fig. 6 Trajectory of robots running the evolved program in the test cases

measured by the number of hits: the number of times the goal was reached. Figure 7 shows the change in the number of hits during the run.

More generally, the hit ratio can be used as a measure of success in evolving cooperative behavior:

$$H = \frac{(\# \text{ fitness cases with success})}{(\text{total number of fitness cases})} \quad (8)$$

Table 3 compares the hit ratio for the three fitness switching methods described in the previous section. It is worth mentioning that the naive approach failed to solve this problem. As expected the fitness switching with
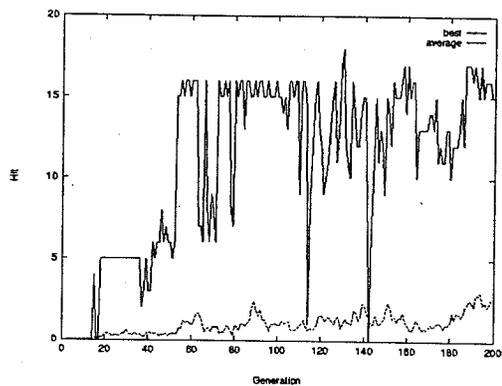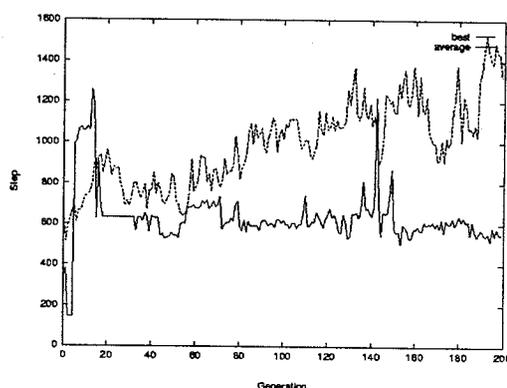

Fig. 7 Number of hits vs. generation

Fig. 8 Average number of steps vs. generation

sequential evolution, the most engineered version, was the best in hit ratio for training. The coevolutionary switching method was competitive to the sequential switching in hit ratio training.

The number of steps for the robot team to move is another important measure of performance for cooperative behavior. Thus it is useful to define the average number of steps, S, for a group of robots:

$$S = \frac{1}{N_{cases}} \sum_{c=1}^{N_{cases}} (\# \text{ steps for fitness case } c) \quad (9)$$

where N_{cases} is the number of fitness cases. In case that the trial failed to reach the goal, the number of steps for the fitness case was counted as the maximum number of steps allowed for each trial.

Figure 8 shows the evolution of the average number of steps made by four robots for 20 different training environments. Shown are the best-of-generation and population-average values. Table 4 compares the performance of three different methods for fitness switching. The values given are the average number of steps made by a group of four robots for 20 different environments for training and test, respectively. The table

Table 3 Hit ratio in comparison

Method	Hit Ratio	
	Training (20)	Test (20)
Naïve	0.05 (1)	0.15 (3)
Sequential	0.90 (18)	0.75 (15)
Coevolution	0.85 (17)	0.65 (13)

Table 4 Average number of steps in comparison

Method	# Nodes	Average Number of Steps	
		Training (20)	Test (20)
Naïve	49	153.8	176.5
Sequential	51	539.8	394.0
Coevolution	44	516.7	588.7

also shows the size of programs evolved by each method.

5. Conclusions

We have presented a genetic programming method for evolving composite cooperative behaviors of multiple robotic agents for table transport. The fitness switching method was based on the observation that, while GP is able to evolve emergent behaviors, the evolution can be more efficient if the program structure and sometimes the evolution strategy is constrained to match the problem structure.

A coevolutionary method was described that is guided by a pool of fitness functions which are defined to reflect to some extent the problem structure without too much need for domain knowledge. The method of coevolutionary fitness switching was suggested as a particular realization of this concept.

We have experimentally shown that coevolution with fitness switching can solve a class of tasks which can not be efficiently solved by naive genetic programming. Experimental results also show that, compared with the carefully designed sequential evolution, the coevolutionary fitness switching is competitive in training performance and better in generalization accuracy.

Acknowledgments

References

1. F.H. Bennett III, "Automatic creation of an efficient multi-agent architecture using genetic programming with architecture-altering operations," *Proc. of First Int. Conf. on Genetic Programming*, J.R. Koza *et al.* (eds.) Cambridge, MA: MIT press, 30-38, 1996.

2. T. Haynes, S. Sen, D. Schoenfeld, and R. Wainwright, "Evolving a team," *Proc. AAAI-95 Fall Symposium on Genetic Programming*, 23-30, AAAI Press, 1995.

3. H. Iba, "Multi-agent learning for a robot navigation task by genetic programming" *Proc. of Second Int. Conf. on Genetic Programming*, Morgan Kaufmann, 195-200, 1997.

4. J. R. Koza, "Genetic Programming: On the Programming of Computers by Means of Natural Selection," Cambridge, MA: MIT Press, 1992.

5. C. R. Kube and H. Zhang, "Collective robotics: From social insects to robots," *Adaptive Behavior*, 2(2), 189-218, 1994.

6. S. Luke and L. Spector, "Evolving teamwork and coordination with genetic programming" *Proc. 1996 Genetic Programming Conf.*, J.R. Koza *et al.* (eds.) Cambridge, MA: MIT Press, 150-156, 1996.

7. M.J. Mataric, "Learning in multi-robot systems," *Adaptation and Learning in Multi-Agent Systems*, G. Weiss and S. Sen (eds.) *LNCS 1042*, Springer, 150-156, 1996.

8. L. E. Parker, "The effect of action recognition and robot awareness in cooperative robotic teams," *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 212-219, 1995.

9. C. Version and L. M. Gambardella, "Learning real team solutions," *Distributed Artificial Intelligence Meets Machine Learning*, 40-61, 1997.

10. G.M. Werner and M.G. Dyer, "Evolution of herdingbehavior in artificial animals," *Proc. Second Int. Conf. on Simulation of Adaptive Behavior*, 393-399, 1993.

11. B.T. Zhang and Y.J. Hong, "A multinet neural architecture for evolving collective robotic intelligence," *Proc. of Int. Conf. on Neural Information Processing*, Springer, 971-974, 1997.

12. B.T. Zhang, P. Ohm, and H. Mühlenbein, "Evolutionary induction of sparse neural trees," *Evolutionary Computation*, 5(2), 213-236, 1997.

13. B.T. Zhang and D.Y. Cho, "Fitness switching: Evolving complex group behaviors using genetic programming," to appear in *Proc. 1998 Int. Conf. on Genetic Programming*, Wisconsin, Madison, July 1998.