Proceedings of the Third Annual Genetic Programming Conference, pp. 431-438, 19

Fitness Switching: Evolving Complex Group Behaviors Using Genetic Programming

Byoung-Tak Zhang
Artificial Intelligence Lab (SCAI)
Dept. of Computer Engineering
Seoul National University
Seoul 151-742, Korea
Phone: +82-2-880-1833
btzhang@scai.snu.ac.kr

Dong-Yeon Cho
Artificial Intelligence Lab (SCAI)
Dept. of Computer Engineering
Seoul National University
Seoul 151-742, Korea
Phone: +82-2-880-1835
dycho@scai.snu.ac.kr

ABSTRACT

This paper considers the problem of transporting a large table using multiple robotic agents. The problem requires at least two group behaviors of homing and herding which are to be coordinated in proper sequence. Existing GP methods for multiagent learning are not practical enough to find an optimal solution in this domain. To evolve this kind of complex cooperative behavior we present 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 coevolve 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.

1 Introduction

A number of attempts have been made to apply genetic programming to evolve cooperative behavior 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 environment. Iba [3] studied three different breeding strategies (homogeneous, heterogeneous, and coevolutionary) for cooperative robot navigation.

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

The problem domain of our study is 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.

To successfully solve this problem using genetic programming, a control program needs to coordinate 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.

In this paper we investigate a novel method that we call *fitness switching*. 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 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 Table Transport Problem

We consider a simplified version of the transportation problem defined as follows. In an $n \times n$ 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.

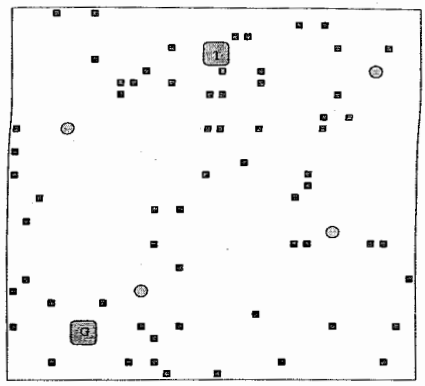


Figure 1: The environment for the transportation problem.

3 Coordination of Behaviors by Fitness Switching

Fitness switching is a method for evolving complex behaviors with genetic programming. It is based on the incremental learning procedure described as follows (see Figure 2):

1. Define primitive actions for the problem domain.
2. Define a small number of micro-behaviors,

$$\mathcal{B} = \{B_1, B_2, \cdots, B_n\} \tag{1}$$

that constitute the original problem-solving behavior.

3. Define a fitness function for each micro-behavior. This makes the pool of fitness functions

$$\mathcal{F} = \{f_1, f_2, \cdots, f_n\}. \tag{2}$$

4. Design a sequence of micro-behaviors or their combinations to achieve the target behavior:

$$S_t = S_{t-1} \cup \{B_t\} \tag{3}$$

where $t = 1, \cdots, n$ and $S_0 = \{\}$. The corresponding sequence of fitness functions are defined as

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

where $t = 1, \cdots, n$ and $F_0 = 0$.

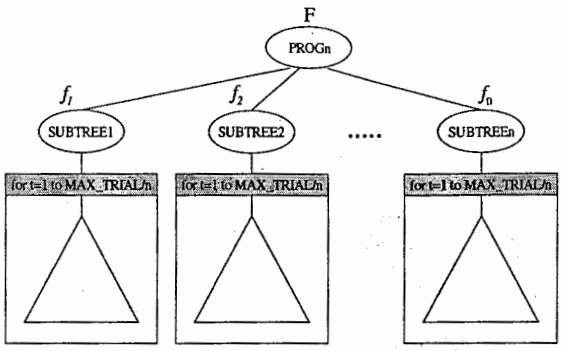


Figure 2: The framework for fitness switching.

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

6. Apply genetic programming to evolve S_t, $t = 1, \cdots, n$ 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 $\mathcal{B} = \{B_1, B_2\}$, where $B_1 = \textit{homing}$ and $B_2 = \textit{herding}$. The set of fitness functions is $\mathcal{F} = \{f_1, f_2\}$. 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:

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

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

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.

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 betw. starting and final pos. of r
A_r	penalty for moving away from other robots
c_i	coefficient for factor i (e.g., $c_1 = 5$)
K	positive constant (e.g., $K = 40$)

Table 1: Symbols used for fitness definition.

Several implementational variants for fitness switching are possible. One simple choice is *naive 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 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 their fitness by f_1.
2. Let A^L_{best} be the best individual evolved above.
3. Run another GP to evolve right *subtrees* for herding behavior by using A^L_{best} 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 the interesting problems that need emergent computations do not belong to this class of problems.

```
For g = 1 to G
  For all A ∈ Pop        // A = (A^1, A^2, ..., A^n)
      F(A) = 0
      For t = 1 to n
          f_t = Execute(A^t)
          F(A) = F(A) + f_t
  NewPop = { }
  For s = 1 to |Pop|/2
      Select P1 and P2 from Pop by F
      P1 = Modify(P1)
      P2 = Modify(P2)
      NewPop = NewPop ∪ {P1, P2}
  Pop = NewPop
A_best = Best(Pop)
```

Figure 3: Fitness switching with coevolution.

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.
2. Measure the fitness of the right *subtree* 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 and their coordination.

More detailed and general description of this procedure is given in Figure 3. The initial population is created with random individuals. Then, the fitness values of individuals are evaluated as described above. The best 50% of the population are selected as possible parents. Two parents are selected to generate two offspring by means of subtree crossover and node mutation. The operators are applied with their associated probability. This procedure is repeated until offspring of population size are generated. After generating all offspring, the best two of the parents replace two offspring selected at random (elitism). This completes one generation and the process is repeated G generations.

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

$$F = F + \beta C \tag{7}$$

where C is the number of nodes in the tree and β is a smll constant (e.g., 0.0001).

4 Experiments and Results

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. Table 2 summarizes the experimental setup for genetic programming.

The function set consists 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. The action of lifting the table is assumed to be included in the move actions. `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 shown in Figure 4.

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

All the robots use the same control program. Public control programs are faster to evolve and they show more consistent group behavior. In evaluating the fitness of robots we made a complete run of the program for one robot, before the fitness of another is measured. This is an efficient way of measuring the fitness, but is a bit different from the real situation in which robots are moving at the same time. Sequential execution of the program can approximate parallel execution if a sufficient number of training cases are used.

Figure 5 shows the change in fitness values during a GP run with coevolutionary fitness switching: The fitness of a tree A is measured by $F(A) = f_1(A) + f_2(A)$,

Parameter	Value
Terminal set	FORWARD, AVOID, RANDOM-MOVE, TURN-TABLE TURN-GOAL, STOP
Function set	IF-OBSTACLE, IF-ROBOT, IF-TABLE, IF-GOAL 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

Table 2: Tableau for the table transportation problem.

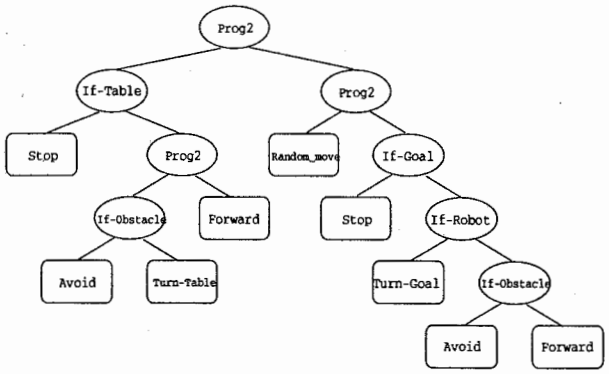


Figure 4: A genetic program for solving the table transportation problem.

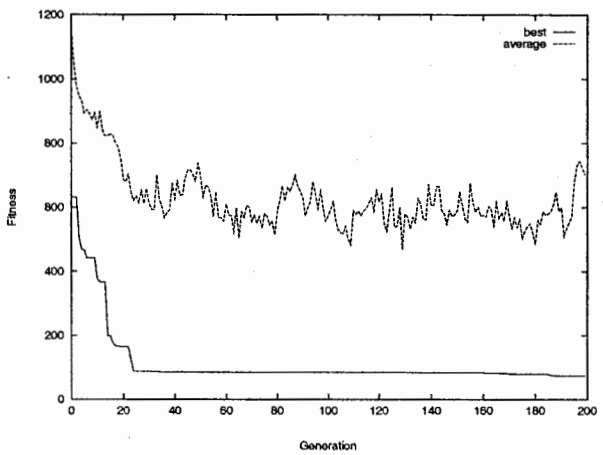


Figure 5: Evolution of fitness values during a GP run: (solid line) best fitness, (dotted line) average fitness.

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.

We examined the evolution process of cooperative behavior by analyzing the performance of best programs at each generation. Shown in Figure 6 are the performance at generations $g = 1, 10, 14, 71$. At $g = 1$, a program was evolved that controls the robots toward the table. At generation $g = 10$, the robots tend to move toward the goal position but no group behavior is observed. At $g = 14$, three of the robots learned the herding behavior but one failed. It took the robots 71 generations to learn perfect homing and herding behaviors to transport the table to the destination.

Figure 7 shows the movement of the robots using the finally evolved program. Shown are snapshots at time steps $s = 1, 20, 38, 70$, which respectively correspond to the initial position, reaching the table (homing), moving in herd to the goal (herding), reaching the goal (transporting).

The genetic programming with coevolutionary fitness switching was able to learn to solve the transportation problem for more than one environments. Figure 8 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 9 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 8 and 9 suggest effectiveness of fitness switching with coevolution as a method for evolving composite cooperative behaviors.

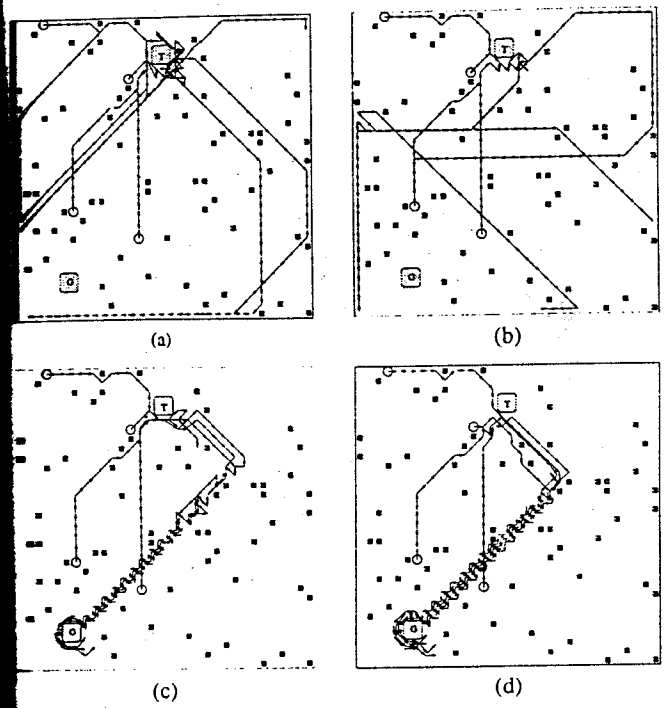
The performance of genetic programs can also be measured by the number of hits: the number of times the goal was reached. Figure 10 shows the change in the number of hits during the run.

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

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

H can be divided into H_{train} and H_{test}. H_{train} is the hit ratio for training cases and H_{test} is the hit ratio for test cases.

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



gure 6: Behavior of robots controlled by the best programs during a run. Shown are the trajectories for generations $g = 1, 10, 14, 71$.

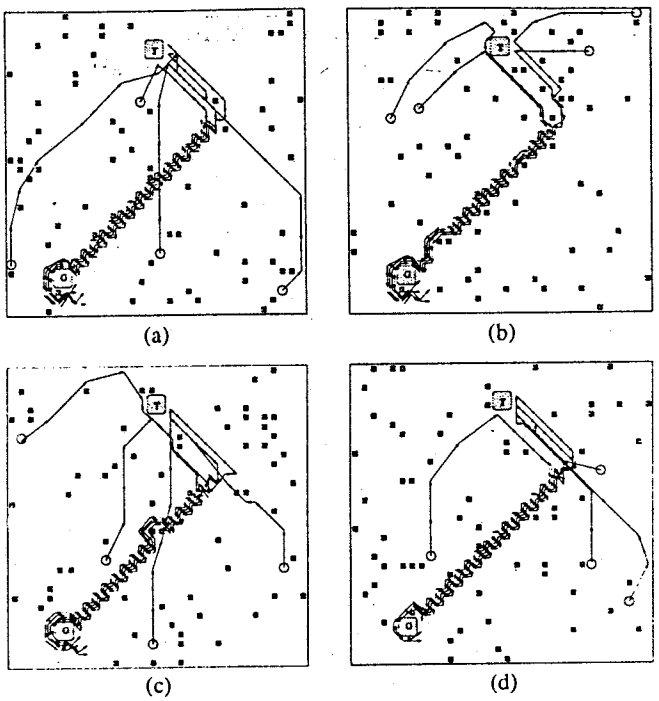
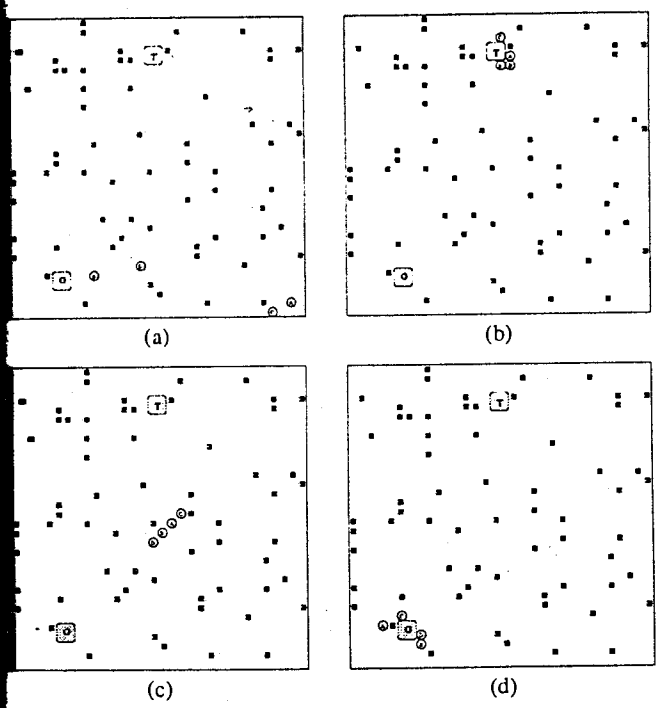


Figure 8: Trajectory of robots running the evolved program in the training cases.



gure 7: Snapshots of robot positions controlled by an olved program at time step $s = 1, 20, 38, 70$.

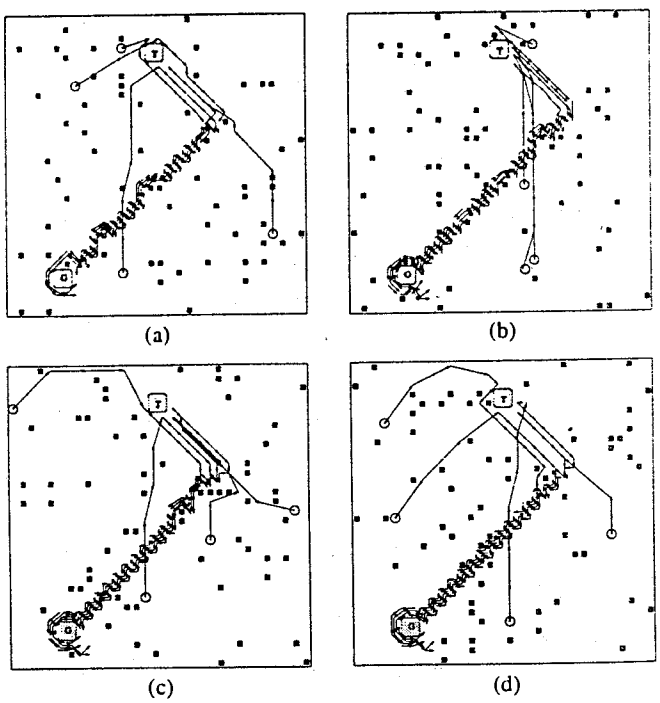


Figure 9: Trajectory of robots running the evolved program in the test cases.

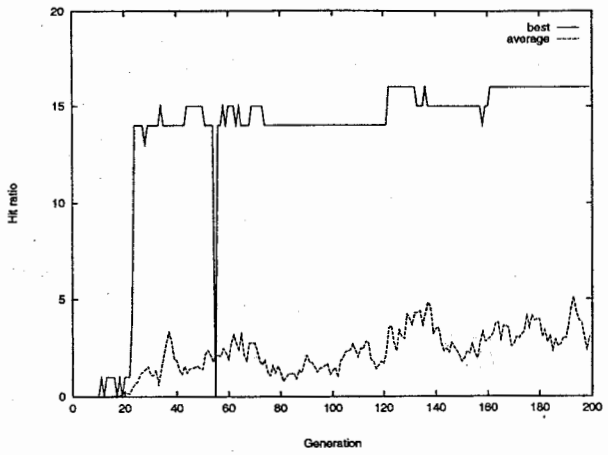


Figure 10: Number of hits vs. generation.

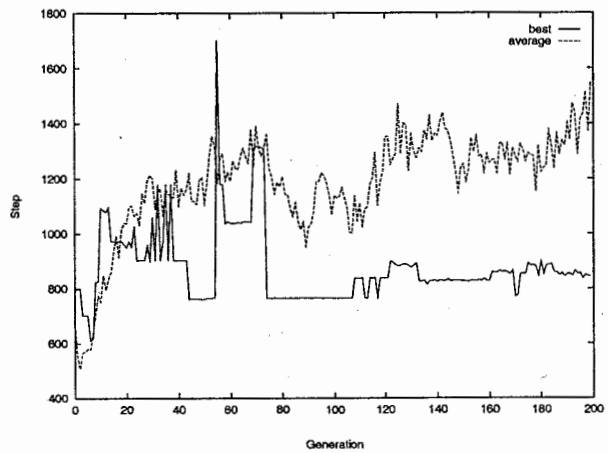


Figure 11: Average number of steps vs. generation.

switching in hit ratio training. The winner in generalization performance was the coevolutionary fitness switching.

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:

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

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. This performance can be separately evaluated by S_{train} and S_{test}. S_{train} is the average number of steps moved to achieve the goal for the training cases. S_{test} is the average step taken to reach the goal for the test cases.

Figure 11 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 also shows the size of programs evolved by each method. The program evolved by coevolutionary fitness switching was the most sparse (33 nodes), which are about half the size evolved by the other methods.

Method	Hit Ratio	
	Training (20)	Test (20)
Naive	0.00 (0)	0.00 (0)
Sequential	0.95 (19)	0.70 (14)
Coevolution	0.80 (16)	0.85 (17)

Table 3: Hit ratio in comparison: The values are derived from 20 fitness cases for training and test, respectively.

Method	#Nodes	Average Number of Steps	
		Training	Test
Naive	59	433.8	370.7
Sequential	67	1003.2	620.5
Coevolution	33	845.7	744.5

Table 4: Average number of steps in comparison: Each figure in the rightmost two columns is the number of steps taken by a group of four robots for table transportation, averaged over all training cases and test cases, respectively.

5 Conclusions

We presented a novel method for evolving composite cooperative behaviors of multiple robotic agents. 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.

Using the table transportation problem 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. Experi-

mental 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. We also observed that coevolutionary fitness switching produced in general more sparse solution trees than sequential evolution.

Acknowledgments

This research was supported in part by the Korea Science and Engineering Foundation (KOSEF) under grant 96-0102-13-01-3.

References

[1] Bennett III, Forrest H. 1996. Automatic creation of an efficient multi-agent architecture using genetic programming with architecture-altering operations. In Koza, John R., Goldberg, David E., Fogel, David B., and Riolo, Rick L. (editors). *Genetic Programming 1996: Proceedings of the First Annual Conference* Cambridge, MA: The MIT Press. Pages 30-38.

[2] Haynes, Thomas, Sen, Sandip, Schoenefeld, Dale, and Wainwright, Roger. 1995. Evolving a team. In *Proceedings AAAI-95 Fall Symposium on Genetic Programming* AAAI Press. Pages 23-30.

[3] Iba, Hitoshi. 1997. Multi-agent learning for a robot navigation task by genetic programming. In Koza, John R., Deb, Kalyanmoy, Dorigo, Marco, Fogel, David B., Garzon, Max, Iba, Hitoshi, and Riolo, Rick L. (editors). *Genetic Programming 1997: Proceedings of the Second Annual Conference* Morgan Kaufmann. Pages 195-200.

[4] Koza, John R. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* Cambridge, MA: The MIT Press.

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

[6] Luke, Sean and Spector, Lee. 1996. Evolving teamwork and coordination with genetic programming. In Koza, John R., Goldberg, David E., Fogel, David B., and Riolo, Rick L. (editors). *Genetic Programming 1996: Proceedings of the First Annual Conference* Cambridge, MA: The MIT Press. Pages 150-156.

[7] Mataric, Maja j. 1996. Learning in multi-robot systems. In Weiss, Gerhard and Sen, Sandip (editors). *Adaptation and Learning in Multi-Agent Systems*, LNCS 1042, Springer. Pages 150-156.

[8] Parker, L. E. 1995. The effect of action recognition and robot awareness in cooperative robotic teams. In *IEEE/RSJ International Conference on Intelligent Robots and Systems.* Pages 212-219.

[9] Versino C. and Gambardella, L. M. 1997. Learning real team solutions. In *Distributed Artificial Intelligence Meets Machine Learning.* Pages 40-61.

[10] Werner G. M. and Dyer M. G. 1993. Evolution of herding behavior in artificial animals. In *Proceedings of Second International Conference on Simulation of Adaptive Behavior.* Pages 393-399.

[11] Zhang, B. T. and Hong, Y. J. 1997. A multinet neural architecture for evolving collective robotic intelligence. In *Proceedings of International Conference on Neural Information Processing* Springer. Pages 971-974.

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