Genetic Programming-Based Alife Techniques for Evolving Collective Robotic Intelligence

Dong-Yeon Cho
Artificial Intelligence Lab (SCAI)
Dept. of Computer Engineering
Seoul National University
Seoul 151-742, Korea
Phone: +82-2-880-7302
dycho@scai.snu.ac.kr

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

Abstract

Control strategies for a multiple robot system should be adaptive and decentralized like those of social insects. To evolve this kind of control programs, we use genetic programming (GP). However, conventional GP methods are difficult to evolve complex coordinated behaviors and not powerful enough to solve the class of problems which require some emergent behaviors to be achieved in sequence. In a previous work, we presented a novel method called fitness switching. Here we extend the fitness switching method by introducing the concept of active data selection to further accelerate evolution speed of GP. Experimental results are reported on a table transport problem in which multiple autonomous mobile robots should cooperate to transport a large and heavy table.

Keywords: genetic programming, artificial life, multiagent learning, fitness switching, training data selection

1 Introduction

Many types of animals seek the company of others. There are several benefits to be gained by gathering into groups with others of the same species, including safety from predators, access to mates, and help in finding food [1].

Similarly, multiple robotic agents can perform some tasks more easily or effectively by cooperation with others. Most of these tasks, however, are so complex that human programming alone is not effective enough to take into account all the detail of real-world situations. In this case, genetic programming provides a way to search for the most fit computer program that solves the problem given by training examples [2]. GP starts with an initial population of randomly generated computer programs. The computer programs are usually represented as trees composed of function and terminal symbols appropriate to the problem domain. They are evolved to better programs using the selection and genetic operations. The ability of the program to solve the problem is measured as its fitness value.

Due to this powerful expressiveness, a number of attempts have been made to evolve cooperative behavior of a group of simple robotic agents using genetic programming. Koza and Bennett [2, 3] used genetic programming to evolve a common program that causes foraging of foods by an ant colony. Luke et al. [4] explored various strategies to develop cooperation in predator-prey environments. Iba [5] studied three different breeding strategies (homogeneous, heterogeneous, and coevolutionary) for cooperative robot navigation. Luke [6] used GP to evolve soccer behaviors of software robots in the RoboCup Soccer Server.

However conventional GP methods are difficult to solve the problems which require some emergent behaviors to be achieved in sequence. Moreover, it requires enormous computational time to evolve complex programs. For these reasons, we used a novel method called fitness switching presented in a previous work [7] and active data selection introduced by one of the authors for efficient training of neural networks [8].

The paper is organized as follows. Section 2 describes the task. Section 3 presents the general framework for fitness switching. Section 4 describes the GP approach with active data selection. Section 5 shows experimental results of the presented methods. Section 6 discusses the result and suggests further work.

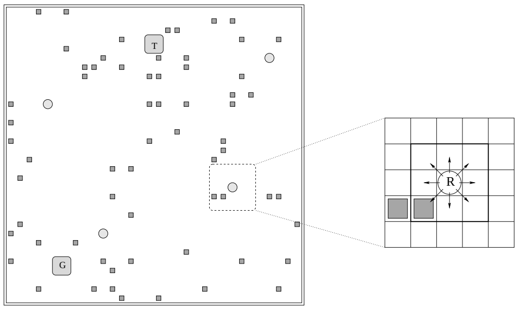

Figure 1: The environment for multiagent learning.

2 The Table Transport Problem

The table transport problem we consider in this paper is 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. A fixed number of obstacles are also placed in the grid. 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 i ($i = 1, .., N_{robots}$) is equipped with a same control program A_i. 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, 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.

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 Evolving Cooperative Behaviors by Fitness Switching

The problem described in Section 2 requires two different behaviors homing and herding, that is, the robots need first to get together around the table and then transport it in team to the destination. In fitness switching method, different parts of a genetic tree are responsible for different behaviors and for each of the subtrees a basis fitness function is defined. Thus, our genetic program tree can be considered as a composition of two subtrees S_1, S_2, where S_1 is responsible for homing and S_2 is for herding.

In the experiments we have used the following fitness functions for S_1 and S_2:

$$\begin{aligned} f_1 &= \sum_{r=1}^{4}\{c_1 \max(X_r, Y_r) + c_2 S_r \\ &\quad + c_3 C_r - c_4 M_r + K\} \qquad (1) \\ f_2 &= \sum_{r=1}^{4}\{c_1 \max(X_r, Y_r) + c_2 S_r \\ &\quad + c_3 C_r - c_4 M_r + c_5 A_r + K\} \qquad (2) \end{aligned}$$

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

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. The following procedure gives more detailed description of this method.

```
For g = 1 to MAX_GEN
    For all T in Pop      // T = (S1,S2)
        Fit(T) = 0;
        f1 = Execute(S1);
        f2 = Execute(S2);
        Fit(T) = f1 + f2;
    New_Pop = { };
    For i = 1 to POP_SIZE/2
        Select P1 and P2 from Pop by Fit;
        Perform Crossover;
        Perform Mutation;
        Insert P1 and P2 into New_Pop;
    Pop = New_Pop;
Tbest = Best(Pop);
```

Since we desire smaller program [9], a complexity term was used in all experiments to penalize large trees:

$$F = F + \beta C \qquad (3)$$

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.

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

4 Genetic Programming with Data Selection

The fitness, $F_i(g)$, of program i at generation g is measured as the average of its fitness values $f_{ij}(g)$ for the cases j in the training set.

$$F_i(g) = \frac{1}{S}\sum_{j=1}^{S} f_{ij}(g) \tag{4}$$

where S is the number of fitness cases.

For the initial population, a small subset of fitness cases, D_0 is chosen from the base training set $D^{(N)}$ of size N

$$D_0 \subset D^{(N)}, \quad |D_0| = n_0. \tag{5}$$

After individuals are evolved by the usual evolutionary process (fitness evaluation, selection, and mating to generate offsprings), a portion of training set, d_g, is chosen randomly from the previous candidate set C_{g-1}

$$d_g \subset C_{g-1}, \quad |d_g| = \lambda, \tag{6}$$

where $C_{g-1} = D^{(N)} - D_{g-1}$. And it is mixed with the previous training set to make a new training set D_g for the next generation

$$D_g = D_{g-1} \cup d_g, \quad D_{g-1} \cap d_g = \{\}. \tag{7}$$

That is, the sequence of training sets for GP active is

$$D_0 \subset D_1 \subset D_2 \subset ... \subset D_G = D^{(N)}, \tag{8}$$

where G is the number of max generation.

Since GP with active data selection uses only a subset of given data accumulated at each generation to evolve programs, while GP standard use all the given training data repeatedly, the number of fitness cases at each generation is

$$S_g = \begin{cases} N, & \text{for GP standard} \\ n_0 + \lambda g & \text{for GP active.} \end{cases} \tag{9}$$

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	100 training worlds, 100 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.

5 Experiments and Results

Table 2 summarizes the experimental setup for genetic programming. The terminal and function set consists of six primitives, respectively. 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 100 training cases are used for evolving the programs for standard GP. The GP run with active data selection use $10 + 3g$ examples out of the given data set, i.e. $n_0 = 10, \lambda = 3$, for fitness evaluation. For both methods, a total of 100 independent worlds were used for evaluating the generalization performance of evolved programs.

Figure 2 shows the result of fitness switching method for evolving composite cooperative behaviors. Figure 3 shows the fitness of two methods with repect to the number of evaluations calculated as a product of the population size and the data size for each generation. The GP active achieved a speed-up factor of approximately two compared with that of the standart GP. The results are summarized in Table 3. Though the GP with active data selection used a smaller set of fitness cases, its training and test performance were slightly better than those of the standard GP.

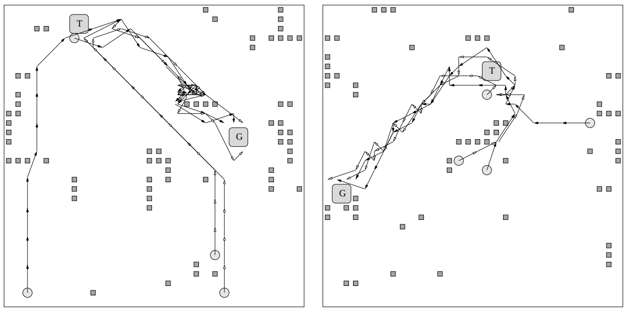

Figure 2: Trajectory of robots running the evolved program in the training and test cases.

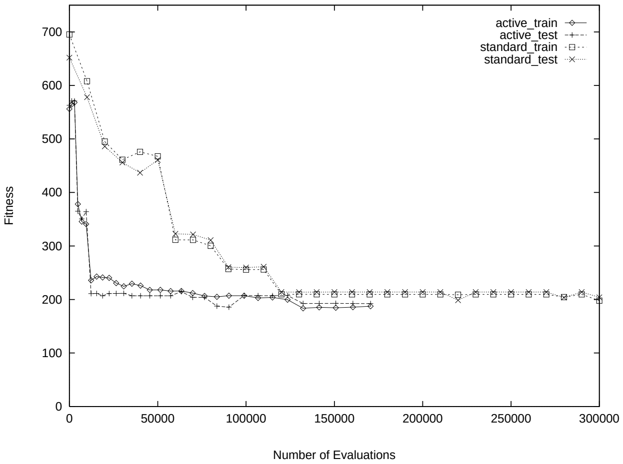

Figure 3: Comparison of fitness values as a function of the number of function evaluations.

Method	Time	Average Number of Steps	
		Training	Test
GP standard	150000	209.562	213.839
GP active	76500	206.654	203.996

Table 3: Comparison of time and average fitness values for the standard GP and the GP with active data selection.

6 Conclusions

Using the table transport problem we have experimentally shown that the fitness switching method can evolve composite cooperative behaviors of multiple robotic agents. Experimental results also show that by reducing the effective number of fitness cases through active data selection the evolution speed of GP can be enhanced without loss of generality of evolved programs.

Acknowledgments

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

References

[1] 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.

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

[3] Bennett III, Forrest H. 1996. Automatic creation of an efficient multi-agent architecture using genetic programming with architecture-altering operations. In J.R. Koza *et al.* (eds.), *Proceedings of the First Annual Conference on Genetic Programming.* The MIT Press. Pages 30-38.

[4] Luke, Sean and Spector, Lee. 1996. Evolving teamwork and coordination with genetic programming. In J.R. Koza *et al.* (eds.), *Proceedings of the First Annual Conference on Genetic Programming.* The MIT Press. Pages 150-156.

[5] Iba, Hitoshi. 1997. Multi-agent learning for a robot navigation task by genetic programming. In J.R. Koza *et al.* (eds.), *Proceedings of the Second Annual Conference on Genetic Programming.* Morgan Kaufmann. Pages 195-200.

[6] Luke, Sean. 1998. Genetic Programming Produced Competitive Soccer Softbot Teams for RoboCuy97. In J.R. Koza *et al.* (eds.), *Proceedings of the Third Annual Conference on Genetic Programming.* Morgan Kaufmann. Pages 214-222.

[7] Zhang, B. T. and Cho, D. Y. 1998. Fitness Switching: Evolving Complex Group Behaviors Using Genetic Programming. In J.R. Koza *et al.* (eds.), *Proceedings of the Third Annual Conference on Genetic Programming.* Morgan Kaufmann. Pages 431-438.

[8] Zhang, B. T. and Veenker, G. 1991. Focused incremental learning for improved generalization with reduced training sets, *Proc. Int. Conf. Artificial Neural Networks*, Kohonen, T. *et al.* (eds.) North-Holland, Pages 227-232.

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