Generating Smart Robot Controllers Through Co-evolution

Kouichi Sakamoto and Qiangfu Zhao

The University of Aizu Aizuwakamatsu, 965-8580, Japan
`{d8061104, qf-zhao}@u-aizu.ac.jp`

Abstract. To evolve robot controllers that generalize well, we should evaluate the controllers using as many environment patterns (evaluation patterns) as possible. However, to evolve the controllers faster, we should use as few evaluation patterns as possible in evaluation. It is difficult to know in advance what patterns can produce good controllers. To solve this problem, this paper studies co-evolution of the robot controllers and the evaluation patterns. To improve the effectiveness of co-evolution, we introduce fitness sharing in the population of evaluation patterns, and the inter-generation fitness in selecting good controllers. Simulation results show that the proposed method can get much better robot controllers than standard co-evolutionary algorithm.

1 Introduction

Neural network is a good model for robot control because it can acquire the control rules automatically through learning. In many cases, the rules cannot be given beforehand, because the environment may change frequently. In such cases, the robot must be smart enough to acquire the rules by itself. Since the teacher signals are often not available, supervised learning cannot be used. It is known that evolutionary learning or reinforcement learning is more efficient. In reinforcement learning, when the robot takes a certain action in the current state, there is some feedback (reward or penalty) from the environment, and the robot can learn based on the feedback. In evolutionary learning, only the final result is evaluated. In general, evolutionary learning uses much less information in learning. In this research, we consider evolutionary learning only.

There is one dilemma in evolving smart robot controllers. On the one hand, the evolution may become unstable if the environment of evolution is not fixed, and good controllers cannot be obtained. On the other hand, if the environment for evolution is fixed, the controller might be good only for that environment. That is, the robot is just a lucky-guy that cannot generalize well. One method for resolving this dilemma is to choose some typical evaluation patterns (environment patterns). The individual robot controllers can be evaluated using these patterns during evolution. Using this method, we can reduce the chance of obtaining lucky-guys because the robot controller should be good for different evaluation patterns. Also, because the evaluation patterns are fixed, the evolution can be stable. The problem is that in general it is difficult to choose the evaluation patterns that can generate good controllers.

In this paper, we try to resolve the above dilemma through co-evolution between the evaluation patterns and the robot controllers. We start from a simple problem first here. The problem considered is to evolve a mobile robot that can reach a given goal from any start point in an environment containing obstacles. The layout of the environment is not changed during evolution. The evaluation patterns in this case are the start points and the orientation of the robot. If we can solve this problem well, we can get some important hints for solving more complex problems in the next step.

When we use co-evolution, it is necessary to consider the following points:

1. Because we would like to obtain different evaluation patterns, fitness sharing is necessary for the population of the evaluation patterns. If we do not use fitness sharing, all evaluation patterns in the population will be very similar after evolution, and the robot so evolved will have a good chance to be the lucky-guy.
2. Since the robots and the evaluation patterns evolve together, when the evaluation patterns change, the criteria for evaluating the robots also change. If the robot individuals are selected based only on the fitness in the current generation, many good individuals can be selected against. This is a problem seen in dinosaur extinction. To solve this problem, we introduce the concept of inter-generation fitness, which is the sum or average fitness of an individual over several generations. The robot individuals are selected based on their inter-generation fitness.

In this paper, we will confirm the effectiveness of the above ideas through simulation experiments. This paper is organized as follows. In the next section, some preliminaries related to this work are provided. In Section 3, we give a short review on the GA (genetic algorithm) based evolution of the robot controllers. In Section 4, we describe the co-evolution in detail. Section 5 provides the simulation results, and Section 6 is the conclusion.

2 Preliminaries

2.1 The Robot Used in This Study

In this study, we use the Khepera robot which is a well-known mini mobile robot used by many researchers for studying intelligent robots. However, we do not use the real robot in our experiments because that will be very time consuming. We use the free software Khepera simulator 2.0. The simulation environment is a squared map of 1000×1000 pixels. The layout of the robot(s), the obstacles, and the goal(s) can be defined by the user.

2.2 The Controller Model

The multilayer perceptron (MLP) is used as the robot controller in this study. The network has one input layer, one hidden layer and one output layer. The input layer has 17 inputs (8 infrared sensors, and 8 light sensors and one bias). For convenience, the number of hidden neurons is also fixed to 17 without fine-tuning. Two output neurons are used to encode four actions:

1. Move forward: both outputs are larger than 0.5.
2. Move back: both outputs are less than 0.5.
3. Turn right: the first output is larger than 0.5, and the second output is less than 0.5.
4. Turn left: the first output is less than 0.5, and the second output is larger than 0.5.

3 Evolving the Robot Controllers Based on Standard GA

The genetic algorithm (GA) can be used for evolving the robot controllers. For this purpose, we need to define the fitness and genotype of the individuals. The genotype of a neural network robot controller is simply the list of all connection weights represented in real numbers. Since the problem considered here is to obtain robot controllers that can drive the robot to a given goal (light source) from any start point, we can define the fitness based on the distance between the robot and the goal after the robot moves for a certain number of steps. If the robot can reach the goal, we should prefer those that can reach the goal quickly. Specifically, the fitness of a robot controller can be calculated as follows:

1. The robot is put to the start points.
2. Let the robot move in the environment based on the sensor inputs and the decisions made by the controller. The robot stops when the number of moves reaches to 2,000, or when it reaches the goal.
3. If the robot cannot reach the goal, the fitness is defined as follows:

$$fitness = A \times (B - distance) \tag{1}$$

where A and B are constants, and distance is the distance between the robot and the goal when the robot stops. In this paper, we defined A=0.1 and B=700. If the robot reaches the goal within 2,000 steps, the bonus given below is added to the fitness.

$$Bonus = [(2000 - n)^2 \times C + D] \tag{2}$$

Here, C=0.000016 and D=14. The parameters A, B, C and D are so chosen that when the robot reaches the goal with 1,000 steps, the fitness is 100. If the robot reaches the goal with 2,000 steps, the fitness becomes 84. The fitness is still higher than the fitness when the robot cannot reach the goal.

4 Co-evolution of Robot Controllers and Evaluation Patterns

4.1 General Considerations

To evolve the robot controller using GA, the selection of start points is very important. If we fix one start point, the robot can reach the goal from this start points, but may not be able to reach goal if we put it to other start point. If we evaluate the robot using start points randomly generated in each generation, the generalization ability may become higher, but the evolution process may not be stable because the evaluation criterion is not constant. To solve this problem, we study the co-evolutionary approach here. The subjects to be co-evolved are the robot controllers and the start points (the evaluation patterns). The Individual of an evaluation pattern includes the x and y coordinates of the start point, and the orientation of robot at that point.

4.2 Standard Co-evolution

First, let us consider the standard co-evolution. Fig. 1 shows the flowchart of co-evolution. It can be described as follows:

– Step 0: Generate two populations, one for the robot controllers, and another for the evaluation patterns. All individuals are initialized using random numbers.
– Step 1: Evaluate the evaluation patterns using all robots. Specifically, for a given evaluation pattern, put all robots, one by one, to the position and orientation defined by this pattern, and let the robot move towards the goal. For each robot, we get a fitness value as defined by (1) and (2). The fitness of the evaluation pattern is defined as the sum of the fitness values of all robots. Note that fitness of an evaluation pattern should be as small as possible. That is, we should find such patterns from which the robots cannot reach the goal easily.
– Step 2: Evolve the evaluation patterns using standard GA defined in the previous section.
– Step 3: Evaluate the robots using all the evaluation patterns. For each robot, the fitness of the robot for one evaluation pattern is defined by (1) and (2). Its total fitness is the sum of the fitness values for all evaluation patterns. The fitness of the robots should be as high as possible.
– Step 4: Evolve the robots using standard GA.
– Step 5: If the terminating condition is satisfied, stop; otherwise, return to Step 1. In our experiment, the terminating condition is very simple. We just restrict the number of generations to 100.

4.3 Modified Co-evolution

In the standard co-evolution, each individual in one population must be evaluated by all individuals in another population. This is very time consuming. To

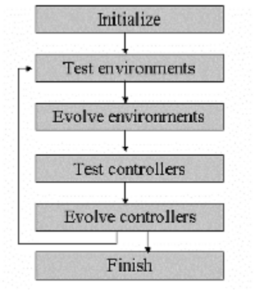

Fig. 1. Flowchart of standard co-evolution

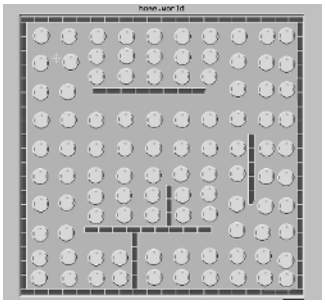

Fig. 2. Start points for testing the robot

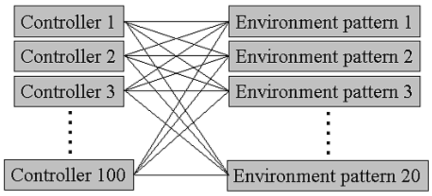

Fig. 3. Evaluation of the individuals in standard co-evolution

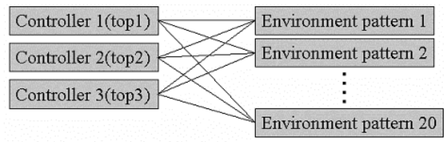

Fig. 4. Evaluation of the environment patterns in the modified co-evolutionary algorithm

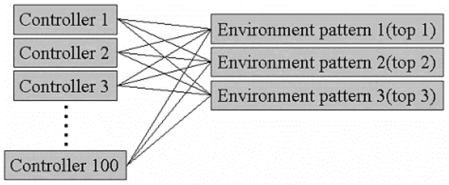
Controller 1
Controller 2
Controller 3
Controller 100
Environment pattern 1(top 1)
Environment pattern 2(top 2)
Environment pattern 3(top 3)

Fig. 5. Evaluation of the robots in the modified co-evolutionary algorithm

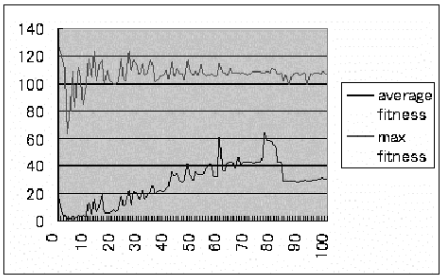

Fig. 6. Evolution curve of co-evolution with fitness sharing

speed-up the evolution process, we consider a modified co-evolution here. In the modified version, each individual is evaluated using several top individuals in another population. In this study, we just use the top 3 individuals in a population to evaluate the individuals in another population (Fig. 4 and Fig. 5). Note that in evaluating the robots, each environment pattern is used twice to make the evaluation results more reliable. Since there are some noises in the sensor inputs, the result for each evaluation is different.

In this study, the size of the robot population is 100, and that of the environment pattern population is 20. Therefore, the total number of evaluations in one generation in the standard co-evolution is $100\times20 + 100\times20\times2 = 6{,}000$. In

Table 1. Result of GA

	Method 1	Method 2
Controller1	33	78
Controller2	52	53
Controller3	43	44
Controller4	48	71
Controller5	26	43
Controller6	32	79
Controller7	32	71
Controller8	26	73
Controller9	19	63
Controller10	52	74
Average	36.3	64.9

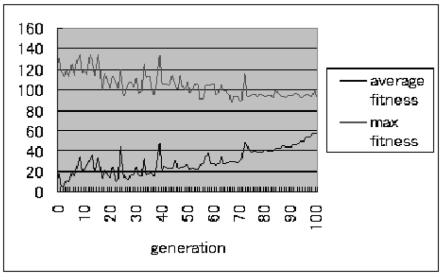

Fig. 7. Evolution curve of co-evolution with fitness sharing and inter-generation fitness

Table 2. Result of standard Co-evolution

	Method 3	Method 4	Method 5
Controller1	61	75	56
Controller2	59	60	61
Controller3	55	57	56
Controller4	52	57	61
Controller5	56	84	87
Controller6	53	77	86
Controller7	52	75	84
Controller8	64	70	78
Controller9	61	59	59
Controller10	51	59	57
Average	56.4	67.3	70.1

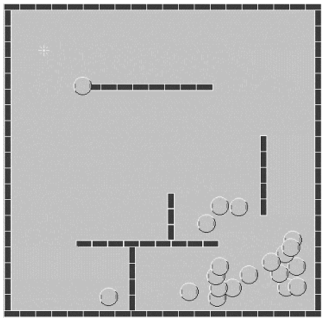

Fig. 8. Evaluation patterns obtained by the standard co-evolution after 100 generations

Table 3. Result of modified Co-evolution

	Method 6	Method 7	Method 8
Controller1	49	64	76
Controller2	51	63	77
Controller3	74	72	74
Controller4	67	67	85
Controller5	59	71	74
Controller6	63	69	72
Controller7	68	65	74
Controller8	63	65	78
Controller9	69	86	73
Controller10	84	82	77
Average	64.7	70.4	76.0

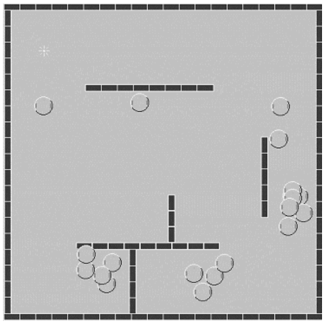

Fig. 9. Evaluation patterns obtained by the co-evolution with fitness sharing after 100 generations

the modified co-evolution, the number of evaluations is $100\times2\times3 + 20\times3 = 660$. The speed-up ratio is about 9.

4.4 Fitness Sharing

There is an important problem in the above co-evolutionary algorithm. In fact, the evaluation patterns after evolution tend to be close to one point in the environment. Usually, this point is the most difficult start point for the robot to reach the goal. Note that our purpose is to evolve robots that can reach the goal from ANY start point. Using only one point (although it is the most difficult one) cannot result in robot that generalizes well.

To solve the above problem, we introduce fitness sharing in the population of the environment patterns. For each individual, we first find its neighbors in a certain range. If there are N neighbors, its fitness is divided by N. In this study, we just adopt a hard limit for defining the neighborhood. For two points p_1 and p_2, if their distance is less than R, we say that they are neighbors. In our simulations, $R = 250$. With fitness sharing, different environment patterns can be obtained through evolution.

4.5 Inter-generation Fitness

Since our purpose is to find one robot that can go from any start point to the goal, we do not use fitness sharing in the robot population. However, there is another problem in evolving the robots. Because the evaluation patterns evolve together with the robots, good robots may suddenly become not so good or even bad in the next generation. This results in an unstable evolution. To solve this problem, we propose to use the inter-generation fitness in selecting the robot individuals. By inter-generation fitness we mean the total or average fitness of the robot over many generations. By so doing, robots that are good for different evaluation patterns generated in different generations can be preserved.

5 Simulation Results

To verify the ideas given in the previous section, we conducted several experiments. Results of the following methods are used for comparison:

– Method 1: Fix the start point to the center of the environment, and evolve the robot using GA;
– Method 2: Generate a new start point at random in each new generation, and evolve the robot using GA;
– Method 3: Evolve the robot and the evaluation pattern (the start point) together via standard co-evolution;
– Method 4: Standard co-evolution with fitness sharing;
– Method 5: Standard co-evolution with fitness sharing and inter-generation fitness based evaluation;
– Method 6: Modified (simplified) co-evolution;
– Method 7: Modified co-evolution with fitness sharing;
– Method 8: Modified co-evolution with fitness sharing and inter-generation fitness based evaluation.

To test the performance (generalization ability) of the robot controllers, we count the number of times the robot can reach the given goal from 100 start points as shown in Fig. 2. The number of successes divided by the total number of test cases is called the success rate. To make the results more reliable, we evolved 10 robot controllers, and evaluated each controller 10 times.

The success rates are used to compare the effectiveness of the methods. The parameters used in the simulations are 1) The number of generations is 100 or 1,000; 2) The size of the population for robots is 100; 3) The size of the population for the environment patterns is 20; 4) The number of individuals (of another population) used for evaluation is all or 3; 5) Weight-by-weight mutation is used, with the mutation rate=0.002; 6) Two-point crossover with the crossover rate=0.8; and 7) Truncation selection with the selection rate=0.2.

Tables 1-3 are the experimental results. The numbers shown in the tables are the success rates (in %) of the robots. From these tables we can see that co-evolution without fitness sharing cannot generate good robot controllers. Method 2 is actually better. Fitness sharing can improve the generalization ability of both standard co-evolution and the modified one. However, from Fig. 6 we can see that fitness sharing alone is not enough to stabilize the evolutionary process. The (average) fitness of the population can drop sharply in some generation. Using inter-generation fitness, the evolution can be more stable (see Fig. 7).

Note that from Fig. 7 we can also see that the fitness of the best robot controller did not increase through evolution. This is because that the evaluation patterns are becoming more and more difficult. This is actually a relative measure. In this sense, the success rate shown in Tables 1-2 is the absolute measure.

One interesting fact is that the modified co-evolutionary algorithm is better than the standard one, although its computation cost is lower. We need to do more experiments to see if this is generally true or not.

Fig. 8 and Fig. 9 show the evaluation patterns obtained by the standard co-evolution and the co-evolution with fitness sharing. Clearly, co-evolution with fitness sharing can get different evaluation patterns, while standard co-evolution tends to provide evaluation patterns in the same location.

6 Conclusion

In this paper, we have investigated different co-evolutionary algorithms for evolving robot controllers. We found that fitness sharing and inter-generation fitness are very useful for improving the performance of the evolved robot controllers. However, the success rates are still not high enough. Further improvement is required to get better controllers.

In the future, we would like to use parallel computation to speed-up the evolution process first. Using parallel computation, we can increase the number of generations as well as the population sizes, and hopefully we can get better robot controllers. We will also try to solve more difficult problems such as evolving robots that can approach to any goal from any start point, and for any given environment layout . In addition, we would like to transform the learned results into understandable and re-usable rules.

References

1. T. Endo and Q. F. Zhao, "Generation of comprehensible decision trees through evolution of training data," Proc. IEEE Congress on Evolutionary Computation (CEC'2002), pp. 1221-1225, 2002.
2. M. Ikarashi "Acquisition of Robot's Moving Strategies through Co-evolution" Master Thesis, The University of Aizu, March 2003.
3. L. M. Fu, "Rule generation from neural networks," IEEE Trans. System, Man, and Cybernetics, Vol. 24, No. 8, pp. 114-124, 1994.
4. Genetic algorithm with co-evolution mechanisms http://www.symlab.sys.i.kyoto-u.ac.jp/research/coevga/coevga.htm
5. K. Hirasawa, K. Nakanishi, T. Eguchi, & J. Hu "Multi Agent Systems with Symbiotic Learning and Evolution -Masbiole- and Its Application" IEEJ Transactions on Electronics, Information and Systems, Vol. 123-C, No.1, pp. 67-74, 2003.
6. T. Eguchi, K. Hirasawa, J. Hu, & J. Murata "Multi Agent Systems with Symbiotic Learning and Evolution using GNP" IEEJ Transactions on Electronics, Information and Systems, Vol. 123-C, No.3, pp. 517-526, 2003.
7. "Evolving controllers for a homogeneous system of physical robots:structured co-operation with minimal sensors" Matt Quinn,Lincoln Smith,Giles Mayley,Phil Husbands
8. K. Sakamoto, T. Takeda and Q. F. Zhao, "Generation of good training data for extracting DTs from evolved NN robot controllers," Proc. IEEE International Conference on Neural Networks and Signal Processing (ICNNSP03), pp. 33-36, Dec. 2003.