International Journal of Fuzzy Logic and Intelligent Systems, vol. 5, no. 4, December 2005 pp. 302-309

Evolution of the Behavioral Knowledge for a Virtual Robot

Su-Chul Hwang, Kyung-Dal Cho

Inha Tech. College
Inchon, 402-752 Rep. of Korea

Abstract

We have studied a model and application that evolves the behavioral knowledge of a virtual robot. The knowledge is represented in classification rules and a neural network, and is learned by a genetic algorithm. The model consists of a virtual robot with behavior knowledge, an environment that it moves in, and an *evolution performer* that includes a genetic algorithm. We have also applied our model to an environment where the robots gather food into a nest. When comparing our model with the conventional method on various test cases, our model showed superior overall learning.

Key words : Classification Rule, Neural Network, Genetic algorithms, Artificial life

1. Introduction

The use of robots in order to perform tasks under a dynamic and an informal environment has grown rapidly. At the same time, many researchers have studied artificial life in order to apply characteristics of ant behavior to control robots or software agents[1][10]. AI methods exist for representing the knowledge of a robot's behavior, such as evolving neural networks[4] and genetic programming techniques[8]. But if the knowledge of a robot is contained in rules or in a semantic network, a robot's response speed may suffer because the inference process may be complicated. If the behavior of a robot is controlled only by a neural network with a genetic algorithm, learning speed may drop.

In this paper, we suggest a model that evolves the virtual robot's behavior to accomplish a task more efficiently and speedily than that of a conventional evolving neural network. For this work, we combine the classification rule with neural network, evolved using a genetic algorithm. We construct a system to apply our model and evaluate it, consisting of virtual robots that have behavior knowledge represented by the classification rule and neural network, the environment that robots move in, and the evolution performer that includes the genetic algorithm. In the virtual environment, robots with intelligent behavior knowledge avoid obstacles and gather foods into a nest. We compare our method with a conventional neural network and genetic algorithm approach using the same conditions and fitness measures.

The next section briefly reviews artificial life, genetic algorithm, the classification rule, and an evolving neural network approach related to our work. Section III describes the suggested model and section IV introduces the implementation of an environment for application and reports on an experimental evaluation. Finally, section V offers conclusions and future work.

Manuscript received Oct. 26, 2005; revised Nov. 21, 2005

2. Related Work

2.1 Artificial Life

Many researchers have studied the field of artificial life, with the intention of interpreting characteristics of life and applying them to engineering applications[2][3][7][14]. Artificial life uses a bottom-up method, which is opposite that of conventional artificial intelligence. It generates complex creative behaviors from simple behavior factors of a lower level [9], and is the approach on which our paper is based.

2.2 Genetic Algorithm

Genetic algorithms are a search method that can be used both for solving problems and for modeling evolutionary systems [5]. The basic idea of a genetic algorithm is very simple. A population of candidate solutions is created, and then the population is evolved with use of various operators (such as mutation, selection and crossover). Natural selection is utilized through an appropriate measure of fitness. There are many ways of implementing this simple idea. We use the genetic algorithm to evolve a neural network and classification rules.

2.3 Classification Rule

In the late 1970's, the classification system was introduced in which classification rules were learned using a genetic algorithm [6]. Each string in the population is a set of rules in this system. Each rule is generated with the classifier in the condition of the rule and the message in the conclusion. There are two approaches; the Michigan[6] and Pittsburgh methods

[13]. Our work is closest to the Pittsburgh method because we generate a set of rules and a neural network from chromosomes, and use these as the behavior knowledge of a virtual robot. In the Pittsburgh method, a robot's entity is characterized not with a single a rule, but with a set of rules. Thus, this approach doesn't evaluate each rule independently, but instead produces sets of rules using the genetic algorithm, and then calculates the fitness for each set. The details of the classification rule will be shown in section 3.2.

2.4 Evolving Neural Network

Evolving a neural network with a genetic algorithm is an effective method for adaptive modeling individuals in the field of artificial life. Both neural networks and genetic algorithms are themselves models to imitate and abstract the principles of organisms, and can solve problems applicable to machine learning and to adaptation. As search methods, genetic algorithms utilize global search that explores surroundings from several nodes in the search space, while neural networks use local search to investigate only around a specific location. The two methods can have complementary effects in search.

There are three ways that a genetic algorithm can be used to evolve a neural network: (1) by evolving the weights between nodes[12][15], (2) by generating the structure of the neural network [11], and (3) to do both[4]. We utilize the third method, on which the learning of the link weights and the generation structure is mixed within the same chromosome. In a neural network, the connection between nodes is represented by a connection descriptor that consists of both the linking weights and the structure. For example, if a connection descriptor has a value '000000011111', then the first '0000' and the next '0001' represent unit number '0' for the beginning node, and '1' for the ending node respectively. Then, the final '1111' represents '-7' as the linking weight between the nodes. Thus the '000000011111' implies a link from node 0 and node 1 with weight value '-7' in a neural network.

3. An Evolving Model for a Virtual Robot

In this chapter, we suggest an evolving model of behavior for virtual robots, and describe the structure of the model and the function of its components. We will also show how to represent the knowledge for a robot's behavior, and how to train the robot.

3.1 Overview of Evolving Model

A structure of our model for evolving a robot's behavior is shown in Fig. 1. It uses the method of machine learning in which a human doesn't provide prior knowledge of the problem domain. The classification rule and the neural network represent the knowledge of a robot's behavior. The knowledge is learned using the genetic algorithm so that over time robots perform better on their assigned task. The *rule descriptor* for classification rules and the *link descriptor* for the neural network are both represented as binary strings.

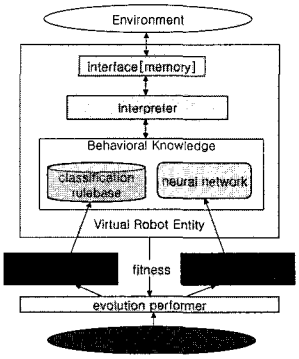


Fig. 1 The structure of the behavior evolution model

An overview of the algorithm for evolutionary learning that we suggest is shown in Fig. 2. The algorithm creates a set of genes (initially random) composed of classification rules and neural networks, which are analyzed by the interpreter and then applied to an environment. The virtual robot then is executed for some time within the environment, and its performance at achieving the goal is evaluated. The genes which adapt best to the environment are selected according to their fitness for the next generation, producing new genes with potentially better performance. As a result, after these steps are repeated for several generations, virtual robots will acquire behavior knowledge that enhances their ability to achieve the goal.

```
BehaviorEvolution()
{
   Initialize 2D population with random bitstrings
   for each generation
   {
      Select two pairs of genes
      generate new environment
      for each pair
      {
            build rules and NN from genes
            place copies of robots into environment
            run environment and determine fitnesses
      }
      Crossover highest fitness genes
      Mutation
      new children replace weakest genes
   }
}
```

Fig. 2 Algorithm of behavior evolution

3.2 The Components of the Model

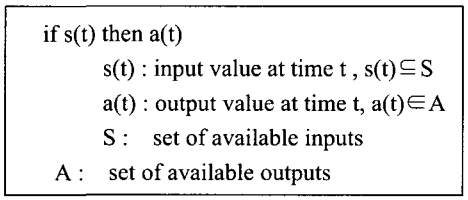
3.2.1 Evolution Performer

Our evolution performer includes the genetic algorithm, which generates and evolves a robot's characteristics. The chromosome of the gene consists of the part of the rule descriptor related with classification rules, the link descriptor for the neural network, and meta-data including the number of rules and the size of network.

A steady-state genetic algorithm, shown in Fig. 2, is used for the evolution of a robot's knowledge in our model, because the exact fitness of the strings is unknown, and can only be estimated by testing the virtual robots. The steady-state method replaces some - not all - individuals of the current gene pool in order to produce the next generation. That is, it initially creates many genes, and then chooses excellent ones of those. A 2-dimensional local tournament selection method is used for selecting superior genes, in which a winner is chosen by competing two neighboring random genes. After crossover and mutation are applied and two new genes are produced, they are substituted for losing genes from another similar tournament.

3.2.2 Classification Rule and Rulebase

Some of the chromosomes generated by the evolution performer are classification rules, and are stored in the rule base that contains knowledge for the robot to utilize input signals from the environment. Each rule is represented by if-then as follows. There is a condition part consisting of {0,1,#}, and a conclusion part consisting of {0,1} - where pound symbol,#, means "don't care". If an input s(t) $\subseteq$S is matched with a rule in the rule base, the rule is fired and a consequence a(t)$\in$A is run. All input values are in S$\in\{0,1,\#\}^L$, such that each member is described by a bit string of length L.



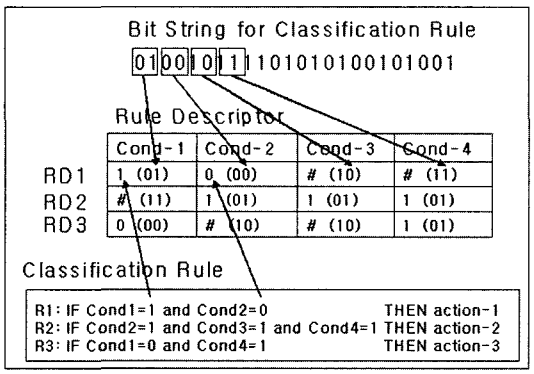


Fig. 3 An example of a rule descriptor

The rule descriptor is used to build classification rules from bit strings, or to match an input signal with the condition part of rules in the rule base. An example of transforming a bit string into a rule descriptor is shown in Fig. 3.

In Fig. 1 the process by which the rule is fired is as follows. First the input signal is compared by bit unit with the conditions of the rules in the rule base. If a match is found, then the interpreter outputs 1, otherwise 0. For example, if input value is "01001011" in Fig. 3, the result is "100" because only the first rule was matched. The interpreter processes this result, in turn outputting the conclusion part of any applicable rule, in order to produce the robot's behavior.

3.2.3 Neural net and its Construction

The neural network is generated by the genetic algorithm and is initially random. For our virtual robots, the network computes its output based on the information from environment via input unit, the result of fired rule, and the content of memory. In our model, the neural network's genetic encoding, as described earlier, consists of three parts: "from" for start node, "to" for end node, and "weight" for link strength. In this way, the descriptor also represents the state of links between nodes in a neural network. If two links happen to contain identical 'from' and 'to' nodes, their weights are added. Fig. 4 shows an example of a neural network represented as a bit string using link descriptors.

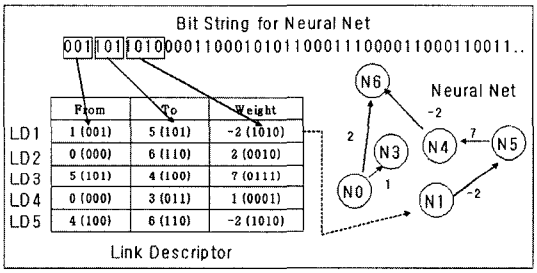


Fig. 4 An example of link descriptors

The particular neural network evolved in our application is shown in Fig. 5. There are 44 input units, corresponding to: sensor inputs 1 through 20 (1 bit each), results from rulebase (10 rules, 1 bit each), and 14 random inputs. There are then 2 hidden layers of 6 units each, and 7 output units for generating the resulting output behavior, described in section 4. (note: since there are a total of 44+6+6+7=63 units in the neural network, the link descriptors in our robot application require 6 bits each, rather than the 3 bits shown in the previous smaller example of Fig. 4.)

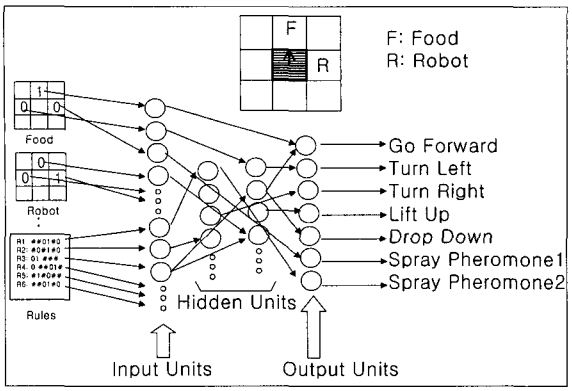


Fig. 5 Neural network architecture for robot

3.2.4 Interpreter and Virtual Robot's Behavior

The interpreter provides some knowledge for robots including the group of virtual robots **R**, rulebase **Rbase**, the **neuralNetwork** and Unit values (**I**: Input layer, **H**: Hidden layer, **O**: Output layer) describing a particular neural network, matches values from each robot with rules in rulebase, then sends the fired rule and the input value together to the neural network. The resulting behavior information is used by the virtual robots to accomplish their task efficiently. Fig. 6 shows an algorithm for the interpreter.

```
Interpret(R,Rbase,neuralNetwork,I,H,O)
{
    for each robot r ∈ R {
  inputString =  sense()
  resultString = CRuleInterpret(InputString,Rbase)
result=neuralNetwork(inputString∪ esultString,neuralNetwork,I,H,O)
  perform(result,0)
}
```

Fig. 6 Interpreter algorithm

4. Application and Evaluation

In order to show the efficacy of the suggested model, we have implemented and evaluated a system with Borland C++ Builder on Windows XP. The details are shown in the following subsection.

4.1 Virtual Environment and Robot Entity

The virtual environment for robot's task is a grid, in which the length of each square is 1. Also on the grid is Nest (robot's nest), Food (robot's food), and Block (obstacles). In this space, robots perform their task, which consists of gathering Food into Nest, using behavior primitives shown in Table 1. The 7 behavior primitives correspond to the 7 outputs from the neural network. Note that two pheromone primitives are included for quickly locating Food under the assumption that there is more food on the paths robots pass through frequently. Some limits are applied to the virtual environment and to the robot's behavior, as follows:

(1) It is impossible for two objects (such as Food and Block) to occupy the same grid location at the same time.
(2) Block can't be moved to any other square.
(3) Only robots can change location of Food.
(4) Robot can only drop down Food into Nest. Putting down Food to places other than Nest is not allowed.

Fig. 7 shows our virtual robot with sensors of three direction and arms.

Table 1. Robot's behavior primitives

Behavior primitives	Meaning
Go Forward	Move one grid step
Turn Left	Turn left 90°
Turn Right	Turn right 90°
Lift Up	Lift up Food
Drop Down	Drop down Food into Nest
Pheromone1	Spray P. on current location
Pheromone2	Spray P. on current location

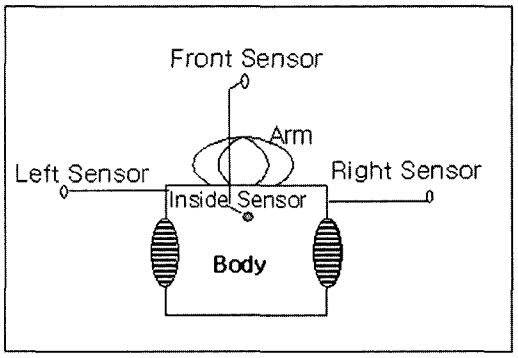


Fig. 7 The virtual robot

4.2 Rulebase and Neural Network

The condition part of each rule in the rulebase has 12 bit string values, because there are twelve binary values coming from the sensors of each robot. The neural network consists of an input layer that receives the sensor information (such as a location or the existence of Food, Block, and Robot) in addition to the result of the fired rule, then two hidden layers with 6 nodes each and an output layer that provides robots with commands to move, turn, lift up and drop down.

4.3 Fitness

Fitness plays an important role in selecting good genes for producing the next generation. For fitness, we use the sum of compensational values, which a robot acquires through acting

in the environment for a given time according to the system clock and is expressed as follows:

(1) If a robot moves and there is Food in front adjacent cell, the robot is given 1 point.

(2) If a robot lifts up Food, the robot is given 1 point.

(3) If a robot drops down Food into the Nest, the robot is given 1000 points.

The fitness for the corresponding gene is then calculated as the sum of the points acquired by each of the n identical robots R_i:

$$\text{Fitness}(gene) = \sum_{i=1}^{n} \text{points}(R_i)$$

4.4 Experimental Evaluation

Parameters used for simulation are shown in Table 2. Robot Group Size indicates the number of robots acting within the environment, and Food Amount is the number of food objects to be maintained. Active Time Unit indicates the total number of input and behavior cycles from each robot. Number of Block Wall indicates the number of obstacles in the environment.

Table 2. Parameters for simulation

Parameters	Values #1	Values #2	Values #3
1. *Number of Block*	**0, 2 or 4**	4	4
2. *Robot Group Size*	20	**10 or 15**	20
3. *Food Amount*	30	30	**10 or 20**
Crossover Ratio	90%	90%	90%
Environment Size	20 * 20	20*20	20*20
Population Size	100	100	100
Nest Size	2*2	2*2	2*2
Rulebase Size	10	10	10
Number of Sense	12	12	12
Active Time Unit	100	100	100
Mutation Ratio	0.05%	0.05%	0.05%

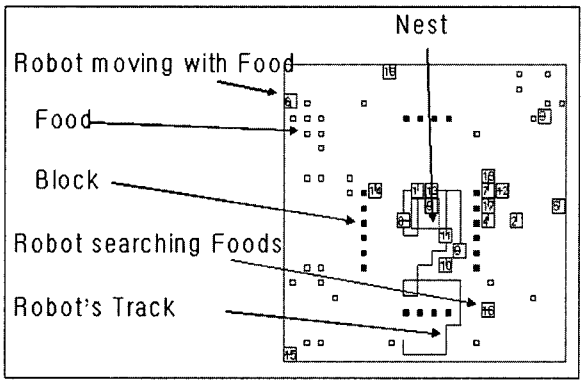


Fig. 8 An example simulation scenario

We have evaluated various values for particular parameters: Group Size, Food Amount and Number of Blocks. Fig. 8 shows a screen snapshot utilizing one set of values. By using the fitness according to each generation under the same conditions, our method is compared against the more traditional method of evolving neural network (without a rulebase). As a result, we have found that on average our fitness values are better. This means that our method evolves robot behavior that is more efficient than that produced by the simple evolving neural network.

5. Test Cases

(1) Various numbers of Blocks (Values #1 in Table 2)

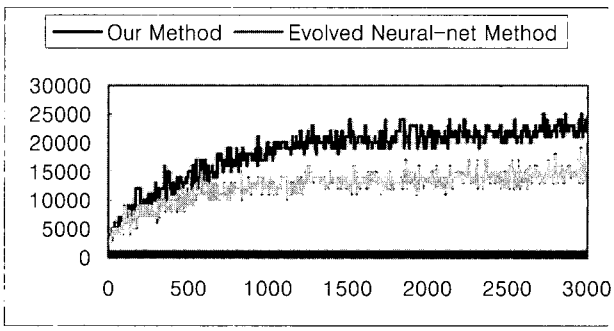
Fig. 9 shows the maximum fitness for each generation when the number of obstacles is changed. In Fig. 9(a) our method's fitness goes up suddenly at around 300 generations, and begins to converge on 30000 in about 600 generations while the simple method's fitness initially converges on 15000 near generation 250. Fig 9(b) and Fig. 9(c) also show that our method converges with higher fitness values throughout the experiment.

(2) Various Robot Group Sizes (Values #2 in Table 2)

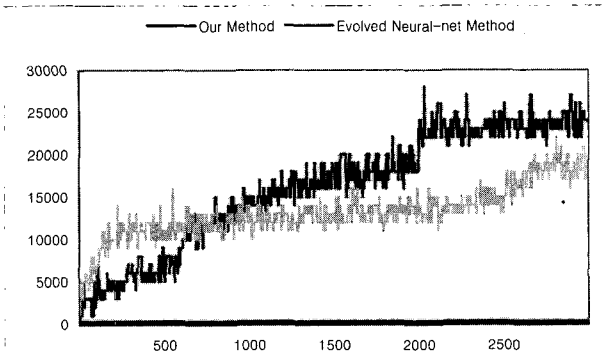
Fig. 10 indicates the maximum fitness for generation in the case of changing the number of robots-10 and 15, respectively - under an environment with four obstacles. Fig. 10(a) shows that when the number of robots is 10, our fitness converges around 12000 and the simple method around 9000. Fig. 10(b) also show that our method achieves higher fitness when the number of robots is increased to 15.

(3) Various Food Amounts (Values #3 in Table 2)

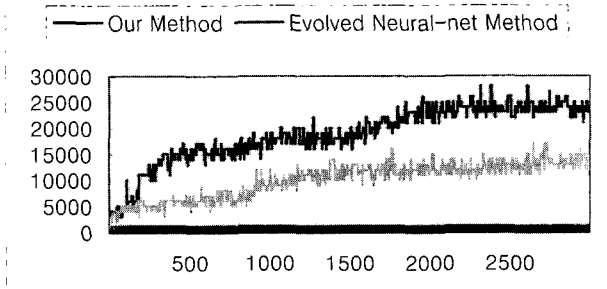
Fig. 11 indicates the fitness by generations in case of changing Food Amount-20 and 10 respectively, under an environment with four obstacles. Our method converges with a few higher values although in fig 11(b) the fitness values for both methods are similar.



(a) Case of no blocks

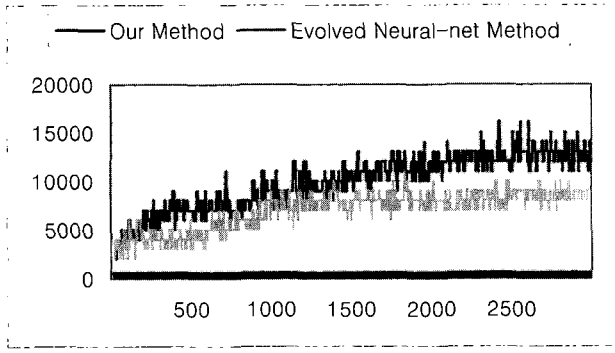


(b) Case of two blocks

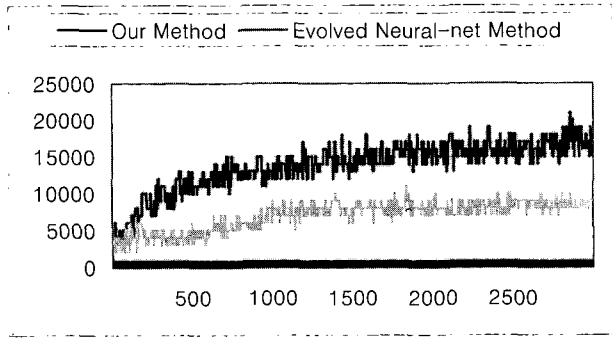


(c) Case of four blocks

Fig. 9 Fitness for various numbers of blocks

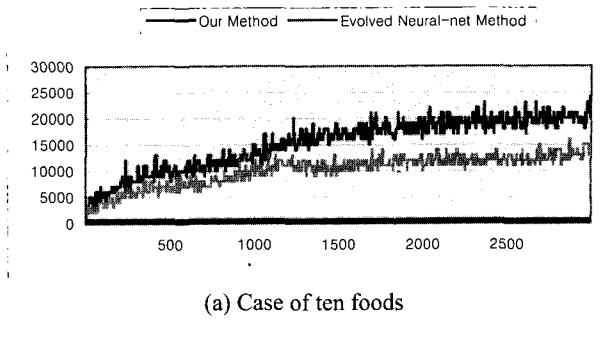


(a) Case of ten robots

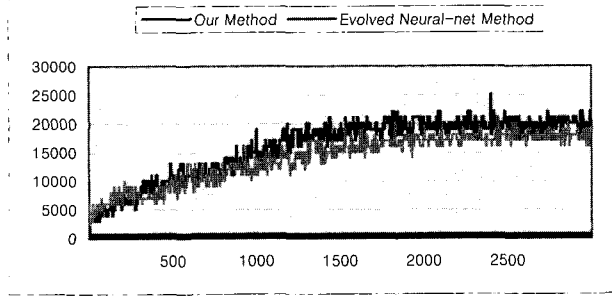


(b) Case of fifteen robots

Fig. 10 Fitness for various numbers of robots



(a) Case of ten foods



(b) Case of twenty foods

Fig. 11 Fitness for various food amounts

6. Discussion on with rules vs. without rules

Fig.12 shows the average fitness and the maximum fitness of gene pool in case of using rule and not using rule through the follow method.

1. Evolve two sets of genes, one *with rules* and one *without rules.*
2. Choose one gene (robot) randomly from each result set
3. Run 15 copies of each robot with environment parameter values #2 in Table 2.
4. Obtain maximum and average points accumulated by each type of robot.

Here we can know that robot's fitness is higher by using the rule. In our implementation of the traditional evolutionary neural network, we used all of the bits for generating the neural network, instead of using some of them for creating rules. Then we found that, despite the fact that our model has fewer bits available for neural network construction, the average fitness of the gene pool is higher than for the traditional approach.

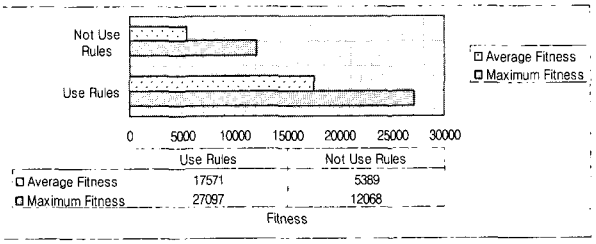


Fig.12 Fitness in case of using rule and not using rule

Fig.13 is a result of another experiment. In this test, we could verify to acquire higher fitness if we use classification rule for robot's knowledge.

1. Evolve two sets of genes, one *with rules* and one *without rules*.
2. Retrieve the maximum fitness gene from each result set
3. Run 15 copies of each robot with environment parameter values #2 in Table 2.
4. Obtain average fitness over 10 runs.

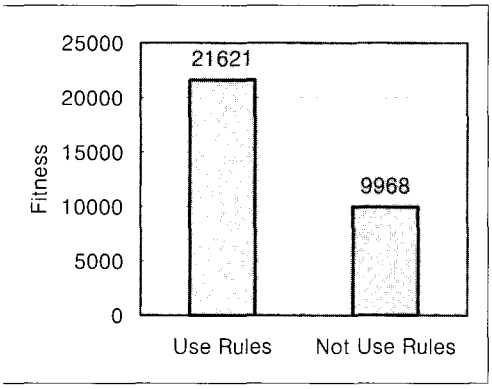


Fig. 13 Average fitness in case of using rule and not rule

7. Conclusions

In the field of virtual robot, the robot's intelligent behavior is important in processing this task. Thus, we considered the knowledge representation for our robot's behavior using a neural network and classification rules. Next, we suggested a model that evolves the robot's knowledge using a genetic algorithm, and implemented a system to apply our idea. By simulating our idea in a given environment, we verified that our model learned more quickly than the conventional method of evolving neural network, for various cases. The learning speed and quality of our robots were superior and accomplished more efficiently their task of gathering food, presumably because of the way in which they stored their knowledge of intelligent behavior. Our virtual robot works well for a variety of situations regardless of the number of robots, blocks, or food in our given environment

Currently we are analyzing the result of our system more closely. For example, we are examining the evolved rules of the best robots to determine which features (sensors, etc.) are the most useful. We are also studying the nature of the moving paths of the robots. In the future, our objective is to find out the learning model for the heterogeneous structure of the robot as well as the homogeneous one.

References

[1] Adami, C., "Introduction to Artificial Life", Springer-Verlag, 1998.

[2] Brown, M., Smith, R., "Effective Use of Directional Information in Multi-Objective Evolutionary Computation", Proc. of GECCO-2003, 2003.

[3] Cho, K.D., "Learning of Emergent Behaviors in Collective Virtual Robots using ANN and Genetic Algorithm", International Journal of Fuzzy Logic and Intelligent System, Vol. 4 Number 4, December, 2004.

[4] Collins, R.J., "Studies in Artificial Evolution", Phd Thesis, Dept. of Computer Science, Univ. of California, Los Angeles, 1992.

[5] Forrest, S., "Genetic Algorithms: Principles of Natural Selection Applied to Computation", Science, Aug 13, 1993.

[6] Holland, J.H., Reitman, J.S., "Cognitive Systems Based on Adaptive Algorithms", Pattern-Directed Inference Systems, Academic Press, NY, 1978.

[7] Kamio, S., "Integration of Genetic Programming and Reinforcement Learning for Real Robots", Proc. of GECCO-2003, 2003.

[8] Koza, J.R., "Genetic Programming: On the Programming of Computers by Means of Natural Selection", MIT Press, 1992.

[9] Langton, C., "Artificial Life II", Addison-Wesley, 1989.

[10] Langton, C., "Artificial Life: An Introduction", MIT Press, 1995.

[11] Miller, G., Todd, P., Hedge, S., "Designing Neural Networks Using Genetic Algorithms," Proc. of IJCAI, 1989.

[12] Montana, D., Davis, L., "Training Feedforward Neural Networks Using Genetic Algorithms", Proc. of IJCAI, 1989.

[13] Smith, S.F., "A Learning System Based on Genetic Adaptive Algorithms", Ph.D. Thesis, Univ. of Pittsburgh, 1980.

[14] Spector, L., Klein, J., Perry, C., Feinstein, M., "Emergence of Collective Behavior in Evolving Populations of Flying Agents", Proc. of GECCO-2003, 2003.

[15] Whitley, D., Hanson, T., "Optimizing Neural Networks Using Faster, More Accurate Genetic Search", Proc. of ICGA-89, 1989.

Su-chul Hwang

He received the BS., MS., and Ph.D. degree in department of Computer Science from Chung-Ang University, Korea, in 1986, 1988, and 1993 respectively. He is currently a professor in the department of Computer System, Inha Tech. College, Korea. His research interests include artificial life, neural network, genetic algorithm, expert system, and AI game.

Phone : 82-32-870-2331
E-mail : schwang@inhatc.ac.kr

Kyung-dal Cho

He received his B.S. degree in in Computer Science from Kyonggi University, Kyonggi, Korea, in 1990. He received the M.S and Ph. D. degree in Computer Science and Engineering from Choong-Ang University, Seoul, Korea, in 1992 and 2004, respectively. His research interests include fuzzy system, artificial life, machine learning, and artificial neural networks.

Phone : 82-32-870-2330
E-mail : kdcho88@hanmail.net