Available online at www.sciencedirect.com
ScienceDirect
Neurocomputing 69 (2006) 2193–2207
NEUROCOMPUTING
www.elsevier.com/locate/neucom

Evolved neural networks based on cellular automata for sensory-motor controller

Kyung-Joong Kim*, Sung-Bae Cho*

Department of Computer Science, Yonsei University, 134 Shinchon-dong, Sudaemoon-ku, Seoul 120-749, South Korea

Received 17 April 2004; received in revised form 15 July 2005; accepted 15 July 2005
Available online 7 February 2006
Communicated by W. Yu

Abstract

Constructing the controller of a mobile robot has several issues to be addressed: how to automate behavior generation procedure, how to insert available domain knowledge effectively, and how to hybrid these methods in an integrated manner. There has been extensive work to construct an optimal neural network for controlling a mobile robot by evolutionary approaches such as genetic algorithm, genetic programming, and so on. However, evolutionary approaches have a difficulty to design the controller that conducts complex behaviors. In order to overcome this shortcoming, we propose an incremental evolution method for neural networks based on cellular automata and a method of combining several evolved modules by a rule-based approach. The incremental evolution method evolves the neural network by starting with simple environment and gradually making it more complex. The multi-modules integration method can make complex behaviors by combining several modules evolved or programmed to do simple behaviors. Simulation results show the potential of the incremental evolution and multi-module integration methods as sophisticated techniques to make the evolved neural network to do complex behaviors. In this paper, we attempt to investigate the applicability of cellular automata-based neural networks and propose sophisticated techniques for the generation of high-level behaviors.

Keywords: Evolutionary neural network; Incremental evolution; Multi-module integration; Cellular automata; Mobile robot control

1. Introduction

How can we minimize human's effort to design robot controller? There are some difficulties to construct a sensory-motor controller for an autonomous mobile robot such as coordinating the mechanics and control system parts of the robot, and managing interaction with external environments [16]. There have been many publications of constructing a mobile robot controller by evolutionary approaches, such as evolving neural network (NN) by genetic algorithm, using genetic programming, and combining fuzzy controller with genetic algorithm to avoid the obstacles, because they have the possibility of automatic construction of the controller without explicit design. In particular, evolving NNs has attracted much interest as a promising way for this problem: NNs can easily exploit various forms of learning and they are resistant to noise that is massively presented in interaction between robot and environments [7].

There are more than 100 publications that discuss evolutionary design methods applied to NNs [27]. One of the important advantages of evolutionary NNs is their adaptability to a dynamic environment, and this adaptive process is achieved through the evolution of connection weights, architectures and learning rules. Designing the optimal architecture of NNs can be formulated as searching in the space where each point represents an architecture. The performance level of all the architectures forms a surface in the space. There are several characteristics in such a surface which make evolutionary algorithms promising candidates over conventional learning models.

Cellular automata (CA) can represent complicated structures and functions with simple rules as a biological brain composed of billions of simple nerve cells does. This

*Corresponding authors.
E-mail addresses: kjkim@cs.yonsei.ac.kr (K.-J. Kim), sbcho@cs.yonsei.ac.kr (S.-B. Cho).

0925-2312/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.neucom.2005.07.013

indicates that the CA might be a good platform of complex NNs developed based on simple rules. Moreover, due to the features (massive parallelism and affinity to VLSI implementations) of CA it is possible to evolve enormous NNs very quickly on parallel hardware. There are special purpose hardwares called CAM (cellular automata machine) for the quick update of the states of CA [23].

In this paper, the NNs based on CA [9] are used to control an autonomous mobile robot. The model develops NNs composed of neurons, axons and dendrites on CA. The mechanism used to evolve NNs on CAM in this paper is the same as the original work which attempts to apply the model to many problems and show the possibility of the approach [3,4]. The applications include 2-D pattern recognizer [5], bit string detection module, multiple timer module and life-like robot called Robokoneko which shows the possibility of the robot behavior such as jumping, catching a ball, and walking [8].

However, the traditional evolutionary method cannot be easily used to design the controller for complex and general behaviors which have multiple goals. We propose two methods for a sensory-motor controller to do complex behaviors with NNs based on CA. One is incrementally evolving the controller by starting with simple environment and gradually making it more complex and general for complex behaviors [11]. Experimental results of this method show the potential as a technique for improving efficiency of evolutionary process. The other is combining several modules evolved or programmed to do a simple behavior by a rule-based approach [21]. Simulation results of this method show the feasibility of integration of multi-module evolved NNs based on CA incrementally.

The rest of this paper illustrates on the sophisticated techniques for NNs based on CA, and simulation results from applying them to controlling a mobile robot in detail. The neural network is used to evolve basic behaviors incrementally and rule-based integration of them is adopted for high-level behaviors. Section 2 offers a brief explanation on backgrounds such as a behavior-based robot, Khepera, used in our simulations, and related works. Section 3 describes the NNs based on CA in detail. Sections 4–6 explain the integrated approach for generating high-level behavior and multi-module integration methods and present the simulation results.

2. Backgrounds

2.1. Behavior-based robot: Khepera

Khepera (see Fig. 1) was originally designed for research and education in the framework of a Swiss Research Priority Program [13]. It allows confrontation to the real world of algorithms developed in simulation for trajectory execution, obstacle avoidance, pre-processing of sensory information, and hypothesis test on behavior processing.

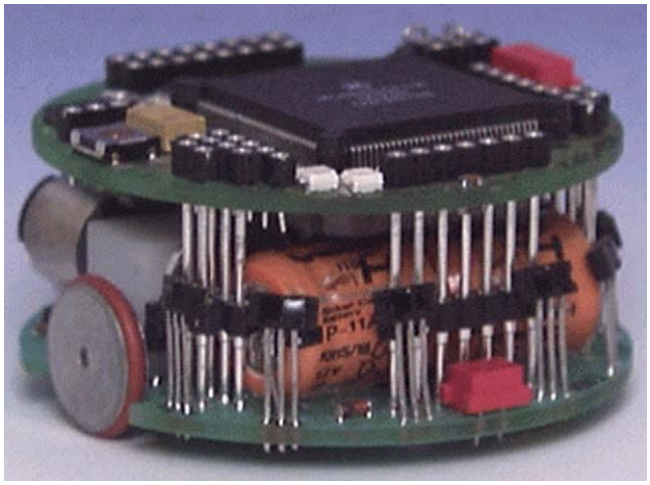

Fig. 1. Mobile robot, Khepera.

2.2. Related works

As mentioned previously, there are some difficulties to construct a sensory-motor controller of an autonomous mobile robot: It is very difficult to coordinate mechanics and control system parts of the robot because it is hard to predict the interaction between these two levels of Khepera robot, because the autonomous robot interacts with external environments. Therefore, building robots is a tough job because the designer has to predict the interactions between the robot and the environment [22]. It is reasonable to use an automatic procedure such as evolutionary computation because this makes it possible to construct a controller without human intervention and we can automatically design its internal structure in detail with less restriction.

Evolving NNs by using genetic algorithm is a useful method among many evolutionary approaches. Smith et al. proposed the evolution of GasNet NN controllers for a robotic visual discrimination problem [20]. Santos et al. presented different aspects of the use of evolution for the successful generation of artificial NN controllers for real robot [19]. In their work, several parameters of an evolutionary/genetic algorithm (GA) and the way they influence the evolution of artificial neural network (ANN) behavioral controllers for real robots were contemplated. Floreano and Mondada [6] have evolved recurrent NNs for a real mobile robot controller. Their aim is to show that more complex behaviors can emerge by reducing the constraints imposed by the fitness function and by increasing the affordabilities of the environment. For this, they have evolved a real mobile robot controller to navigate and avoid bumping against obstacles by GA and battery-recharging behavior by using the simple fitness function. Walker and Miglino [26] have presented that GA is successfully used to evolve ANNs networks for efficient exploration strategies in a population of software simulated Khepera robots.

Barlow et al. developed autonomous navigation controllers for fixed wing unmanned aerial vehicle (UAV) applications using incremental evolution with multi-objective genetic programming [1]. Walker described the effect of mutating every individual in the seed populations of incremental evolution [25] and compared the performance of incremental evolution to that of direct evolution in multi-variable symbolic regression problem [24]. Parker et al. studied the incremental evolution of NNs to control hexapod robot locomotion [17].

3. Neural networks evolved on cellular automata

NNs based on CA finally aims at developing an artificial brain. In particular, due to the features (massive parallelism and affinity to VLSI implementations) of CA the architecture of NNs composed of millions of neurons can be evolved very quickly on parallel hardware. It is already proved that it can be implemented on CAM-brain machine (CBM) with field programmable gate array (FPGA). It is basically designed to create and evolve CA-based NN modules in real time [3].

However, there are only a few simulation results of this model in some problems such as binary comparator, timer, sine curve generator, and so on. Recently, they are attempting to make NN modules evolved to control a lifesize kitten robot, Robokoneko, for showing off the capacities of the model [8].

3.1. Chromosome representation

The model makes NNs based on CA with its own chromosome that has information about the cell structure and state of CA. Therefore, it is possible to evolve and adapt the structure of the NN to a specific task by GA. Fig. 2 shows the evolution procedure of this model. In general, GA generates the population of individuals and evolves them with genetic operators such as mutation and crossover [10]. We have used the GA to search for the optimal NN. At first, a half of the population that has better fitness value is selected to produce new population. Two individuals in the new population are randomly selected and parts of them are exchanged by one-point crossover. Mutation is operated in the segment of chromosome (if there is n-bit chromosome, n segments exist and 1 bit means 1 segment). Mutation changes the value of segment as 0 if the value is 1 (vice versa). The GA generates a new population from the fittest individuals for the given problem.

CA consists of a number of cells with states that change according to a set of rules, based on the states of its surrounding cells. The state of a cell which is located at (x, y, z) at time $t+1$ in three dimensional CA is as follows:

$$s^{t+1}_{(x,y,z)} = f(s^t_{(x+1,y,z)}, s^t_{(x,y+1,z)}, s^t_{(x,y,z+1)}, s^t_{(x,y,z)}, s^t_{(x-1,y,z)}, s^t_{(x,y-1,z)}, s^t_{(x,y,z-1)}), \quad (1)$$

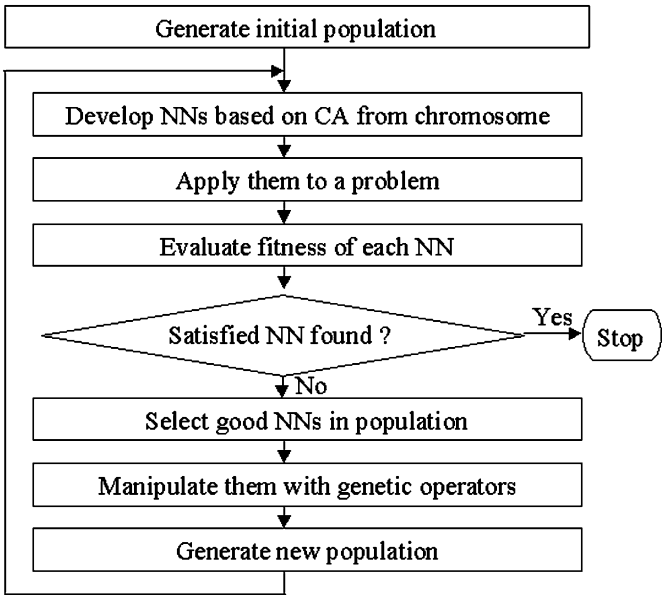

Fig. 2. The procedure of evolving neural networks based on cellular automata.

where $s^t_{(x,y,z)}$ denotes the state of cell at time t, $f()$ denotes the transition function that defines the rule governing state change of a cell. NN structure composed of blank, neuron, axon and dendrite is grown inside 3-D CA space encoded by chromosome. Roles of each cell are as follows:

- Blank: If cell state is blank, it represents empty space and cannot transmit any signals.
- Neuron: It collects signals from surrounding dendrite cells which are accumulated. If the sum of collected signals is greater than a threshold, neuron cells send them to surrounding axon cells.
- Axon: It sends signals received from neurons to the neighborhood cells.
- Dendrite: It collects signals from neighborhood cells and passes them to a connected neuron in the end.

Neighborhood cells of one cell mean surrounding cells (north, south, west and east in 2-D CA space and top and bottom added to them in 3-D CA space). The states of each cell and program (or rules) deciding it with those of neighbors are determined by chromosome (see Fig. 3). The information encoded in a chromosome determines a NN architecture. To represent the whole structure of a NN, chromosome has the same number of segments with the cells in CA space and each segment has information of each cell. A segment can change blank cell to neuron cell (NS bit of Fig. 3), and decide the directions of sending received signals to neighborhood cells (N, S, E, W, T and B bits of Fig. 3). The signal can be only sent to the direction where the bit corresponds to 1.

3.2. Growth phase

The growth phase organizes neural structure and makes the signal trails among neurons. Neurons are seeded in CA

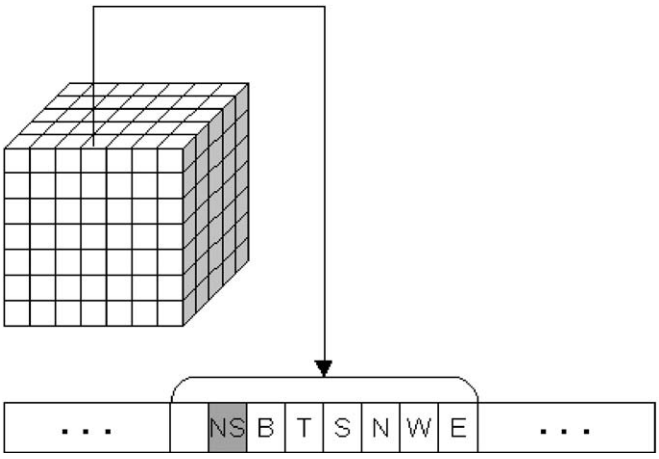

Fig. 3. Information encoded in chromosome.

Table 1
State transition rules

Old cell state				New cell state
Type	Neuron seed	The number of axon signal	The number of dendrite signal	Type
00	0	1	*X*	11
00	0	*X*	1	10
00	0	0 or >1	0 or >1	00
00	1	*X*	*X*	01

X Means "don't care"; 00: Empty; 01: Neuron; 10: Dendrite; 11: Axon.

space by chromosome. The NN structure grows by sending two kinds of growth signals (axon and dendrite) to neighborhood cells. A neuron sends axon growth signal to two opposite directions determined by chromosome and dendrite growth signal to the remaining four directions. Table 1 explains state transition rules. The number of axonal or dendrite signals is dependent on surrounding cells. The detailed procedure is as follows:

Step 1: A chromosome is randomly made and the states of all cells are initialized as blank. At this point some of the cells are specified as neuron with some probability.

Step 2: A neuron cell sends axon and dendrite growth signals to the direction determined by chromosome. Axon growth signal is sent to two opposite directions and dendrite growth signal is sent to the remaining directions.

Step 3: The blank cell received growth signal changes to axon or dendrite cell according to the type of growth signal. It sends the signals received from other cells to the direction determined by chromosome.

Step 4: Every blank cell goes through Step 3. Repeating this process, the final NN is obtained when the state of every cell changes no longer.

Fig. 4 shows the growing process in 4×4 2-D CA space. In this figure, the cell which is filled with oblique lines is blank cell, and the black arrows show the direction of signaling determined by chromosome. Fig. 4(a) shows the process of seeding a neuron in blank cells, where a neuron is located in (x_2, y_2). Fig. 4(b) shows that the neuron cell sends growth signal to surrounding cells. Fig. 4(c) shows Step 3 of the above procedure, and Fig. 4(d) shows that blank cells grow into axon or dendrite.

3.3. Signaling phase

Signaling phase transmits the signal from input to output cells continuously. The trails of signaling are performed with the structure evolved at the growth phase. Each cell plays a different role according to the cell type. If the cell type is neuron, it gets the signal from connected dendrite cells and gives the signal to connected axon cells when the sum of signals is greater than a threshold. If the cell type is dendrite, it collects signals from the connected cells and eventually passes them to the neuron body. If the cell type is axon, it distributes signals originating from the neuron body.

The location of input and output cells in CA space is determined in advance. During signaling phase, the fitness evaluation is executed. The detailed procedure of signaling is as follows:

Step 1: If the state of input cell is neuron, it receives signals from outside and accumulates them. It implies that no signaling occurs when the input cell is not a neuron.

Step 2: If the sum of signals from outside is greater than a threshold, it sends $+1$ to excitatory axon and -1 to inhibitory axon.

Step 3: Axon cell that received signals from neuron sends the signal to the surrounding cells except the cell that sends the signal. Repeating this process, axon cell distributes the signal to neighborhood cells continuously.

Step 4: When dendrite cells belonging to another neuron receive the signals, they collect these signals and send them to a neuron.

Step 5: A neuron cell that received signals from dendrite cell goes through Step 2 and it sends the signals to the surrounding cells. Repeating this process, the signal from input cell is passed to the neuron cells and finally arrives at output neuron.

Fitness is evaluated by the output value in this process. Depending on the task, several methods can be used such as the number of activated cells, Hamming distance of the target and output vectors, and some function to evaluate the fitness. Fig. 5 shows the directions of signals after neuron, axon and dendrite are made. In this figure, neuron sends excitatory signal $(+1)$ to the neighborhood cell that has grown into excitatory axon, and inhibitory signal (-1) to the neighborhood cell that has grown into inhibitory axon. Dendrite cell collects signals from neighborhood cells and sends them to a neuron, and axon cell distributes the signals originated from neuron to neighborhood cells.

4. Mobile robot control

Applying CA based NN to a mobile robot control requires the following process. NN structure is made at

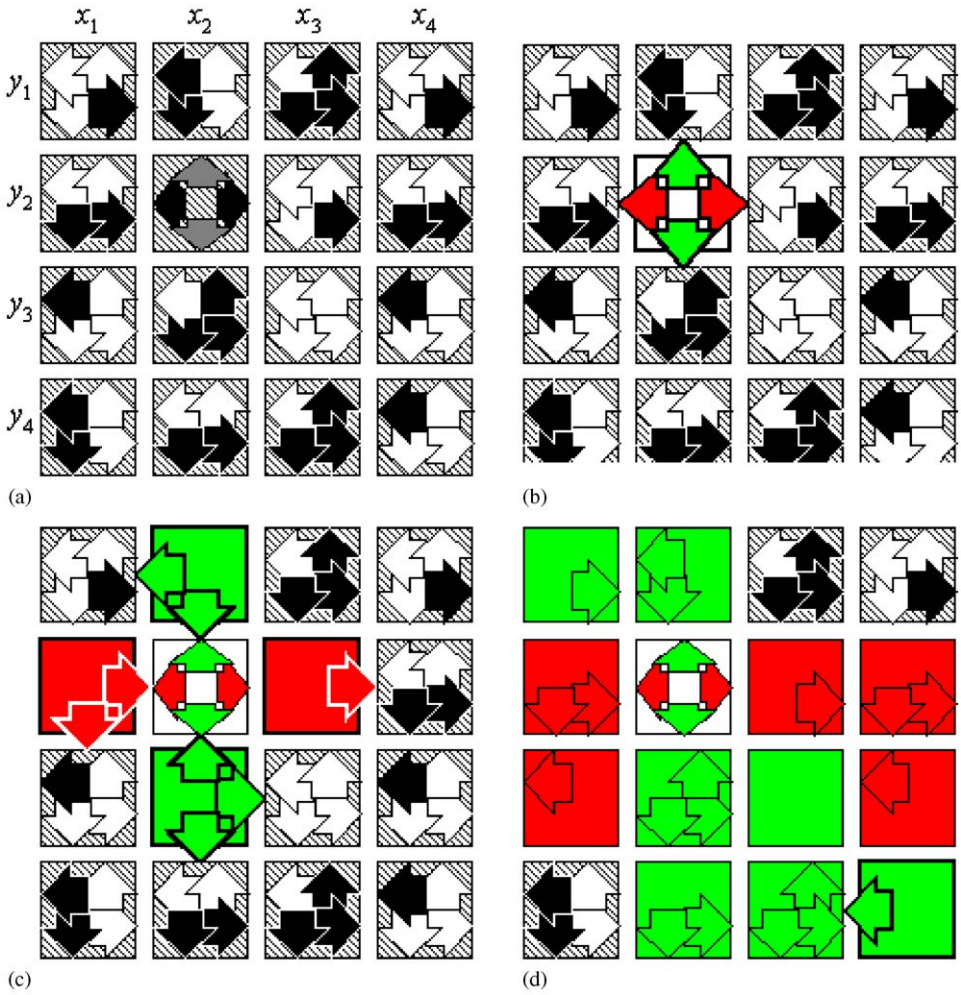

Fig. 4. The process of growth of neural network structure: (a) black arrow represents signaling direction determined by chromosome, and a neuron is located in (x_2, y_2); (b) the neuron sends growth signals; (c) the cell state is determined according to the type of growth signals; (d) propagating growth signals, blank cells become axon or dendrite.

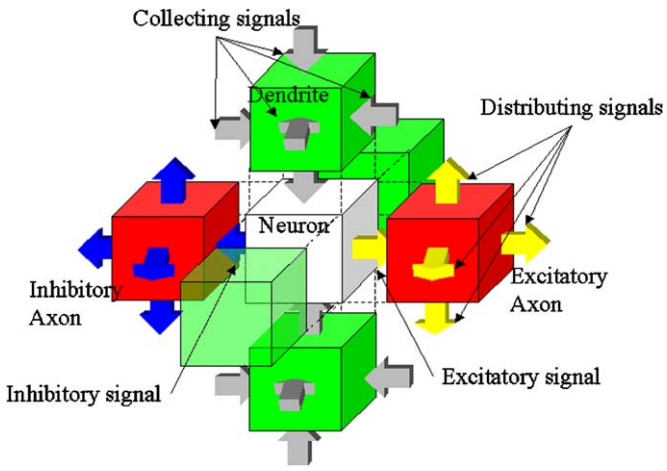

Fig. 5. The process of signaling of neural network. Neuron cell in CA-based neural network does a similar job with that in feed-forward neural network: it collects signals and accumulates them until the signals exceed the threshold.

first, and then sensor values from Khepera simulator are used as inputs of the input cells of the NN which transmits signal from input to output cells. As the output values of the network are inputted to Khepera simulator, the robot moves. When the robot bumps against the obstacle or reaches the goal, its fitness is calculated. Chromosomes are reproduced according to it.

4.1. Method

There are many problems in applying the NNs based on CA to control a mobile robot. Because the network cannot perfectly utilize activation values of robot sensors the activated range of input neurons is made differently according to the value. The higher the value of sensor, the more the number of neurons that gets the input signals. In addition, because delay time is needed until the network makes output value, dummy signaling phase should be executed for some duration until the signals started from

Table 2
Environmental variables

The size of CA space	$5 \times 5 \times 5$, $10 \times 10 \times 10$
Population size	100
The proportion of neurons in cellular automata space	5%
Crossover rate	0.3
Mutation rate	0.05
Threshold values of neuron	$-16, 15$

input cells arrive at output cells. These make the robot to react timely in situation.

Khepera robot simulator has been programmed with C++, and the simulation has been performed on Sun UltraSparc. The population size is 100, and the maximum step of sensor sampling for the evaluation of the individual is 30 000. We use $5 \times 5 \times 5$ and $10 \times 10 \times 10$ CA space to solve the problem ($5 \times 5 \times 5$ CA is used to analyze its structure). Table 2 shows the parameters used in our simulation. Among sensors of the robot, light sensors are not used and 0, 2, 3, 5 distance sensors are used for evolving avoiding obstacle behavior. Only four sensors are used in this simulation and each input cell is in the center region of four faces of hexahedrons in CA space. Output cells are in the top and bottom faces of hexahedrons, and produce the speed of left and right motors, respectively.

The number of bumps in navigation is used for fitness evaluation. If the robot goes round the simulation space completely without bumping, the fitness becomes 1.0 and the robot stops moving. If robot crashes to the wall it stops movement and the fitness is decreased. Even if we do not give any knowledge about the movements, such as "turning right," "turning left" and "avoiding obstacles," the evolution guides NNs naturally to solve the problems.

"Following Light"-module is evolved to go to the light source. Input values of NN module are light sensor values of the robot. The fitness value is defined as follows:

$$Fitness = S * (1 - \sqrt{V}) * (D), \quad (2)$$

where S, is the average speed of the two wheels, V, the difference between the velocities of two wheels, D, the normalized value of light sensors.

The fitness evaluation causes the robot to go straight quickly and reach the nearby light source. Population size is 50 and the fitness of individual is computed by the average of four runs. Of course, the direction of the robot changes in each run.

4.2. Simulation results

Fig. 6 shows the change of fitness with generation. At the 10th generation, better solution emerges suddenly and it struggles to survive. The fluctuation from the 14th generation to the 20th generation exists because of no eliticism. It seems to make evolution search for more robust solution by exploring extensively. After the 21st

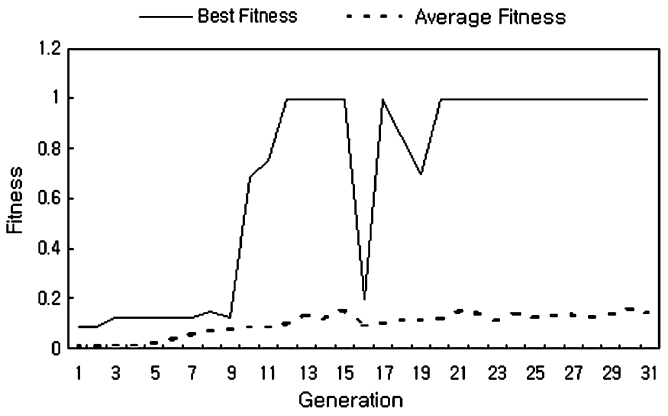

Fig. 6. The change of the best and average fitnesses when applied to mobile robot control ($5 \times 5 \times 5$ cellular automata space).

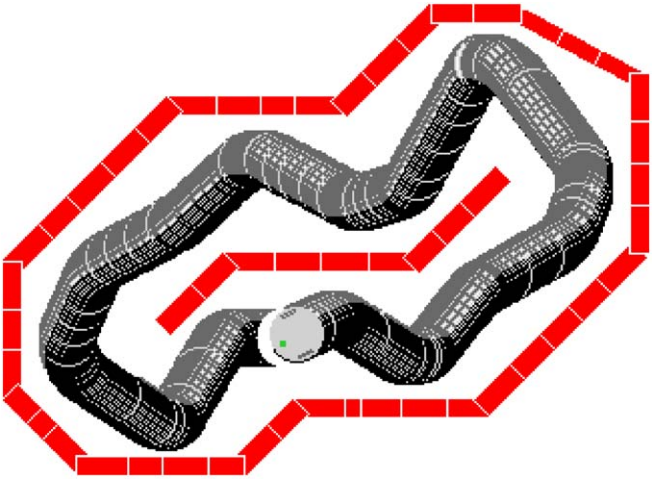

Fig. 7. The trajectory of a successful individual evolved in $5 \times 5 \times 5$ cellular automata space.

generation, individuals that have fitness value of one keep on appearing. Fig. 7 shows the trajectory of a successful robot. This is less smooth than that obtained in our previous work [2], but this robot controller has smaller number of neurons, which makes the analysis easier.

Fig. 8 shows the architecture of the NN evolved. Dotted arrow represents inhibitory connection, and lined arrow represents excitatory connection. This has been obtained by tracing the activation values of each neuron. The number of neurons is 12, but neurons 8, 11 and 12 are inaccessible because these neurons are not in the path of input to output neurons. Neurons 2 and 10 are output neurons to produce the velocity of left and right wheels, and neurons 3, 4, 5 and 6 are input neurons. Neuron 3 is for the front sensor of the robot, neurons 5 and 6 are for the left sensor, and neuron 4 is for the right sensor. The output neuron for left wheel has only one input from the right sensors. The output neuron for right wheel has two inputs from left sensors and hidden neurons. The link between the output neuron and the neuron for left sensors is positive but the link with the hidden neuron is negative.

The input from the left sensors can accelerate the velocity of right wheel but inputs from left, right and front sensors can deaccelerate the right wheel with some delay.

Fig. 8 shows the interpreted neural architecture of the 3-D NN based on CA. The procedure of transforming the CA-based NN into such an understandable NN architecture is as follows. At first, each neuron cell is indexed and their relationships are analyzed by tracing the neural trail formed at the growth phase. Input and output neurons are predefined at designing stage of controller. The remaining neurons are all hidden neurons. Some of them are ignored because they are isolated from the input–output signal stream. Based on the information of the link and the direction of signal, the edge is visualized. Each neuron is represented as a circle. This interpreted NN makes someone understand the structure of CA-based NN. However, the visualization cannot reflect the different time-delaying property of each connection.

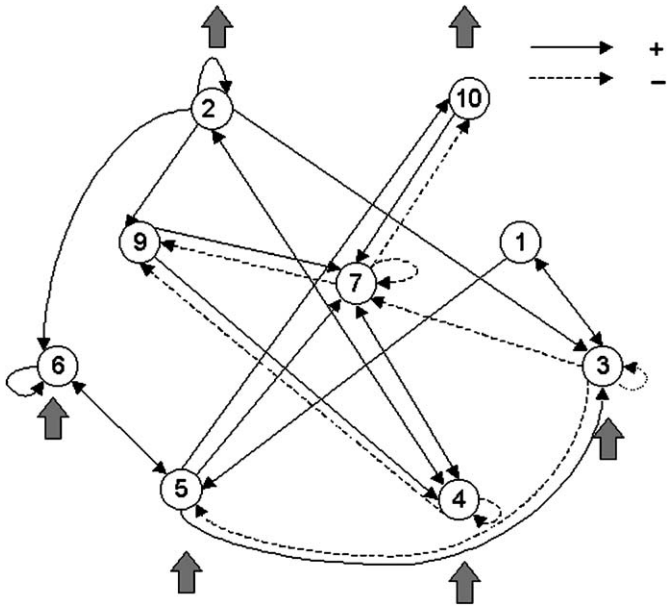

Fig. 8. The architecture of the neural networks evolved in $5 \times 5 \times 5$ cellular automata space.

Fig. 9 shows the change of the best and average fitnesses for evolving $10 \times 10 \times 10$ CA for following light behavior. It shows evolutionary change of 200 generations. The best solution has emerged at the 51st generation but it disappears because of no eliticism. However, there is steady increase in average fitnesses. It shows that evolving $10 \times 10 \times 10$ CA needs more generations for convergence than evolving $5 \times 5 \times 5$ CA. Fig. 10(a)–(d) shows the trajectory of the evolved robot when the angles are $0°, 90°, 180°$ and $270°$, respectively. The robot can be oriented in the intended direction at the initial position. $0°$ means that robot faces the right side of the environment. The robot changes its orientation to the counterclockwise rotation, if the angle value increases. In evolutionary procedure, the average success of four trials (0, 90, 180, and $270°$) is used and the figure shows the successful one at the last generation.

5. Incremental evolution

An evolutionary method suffers from the inefficiency to solve complex problem because it is difficult to find proper solution in huge search space. Therefore, it is more feasible to solve a complex problem by evolving the controller with simpler environments and gradually developing it with more general and complex environments [11]. Evaluation tasks $t_1, t_2, t_3, \ldots, t_n$ are derived by transforming a goal task in incremental evolution, where n is the number of tasks and t_n is the goal task. In this set, t_i is easier than t_{i+1} for all i: $0 < i \leqslant n$. Thus, population is evaluated in task t_i and then task t_{i+1} and it finally does in the goal task, t_n [11]. Fig. 11 shows the process of this method. It is expected to produce a controller doing complex and general behaviors.

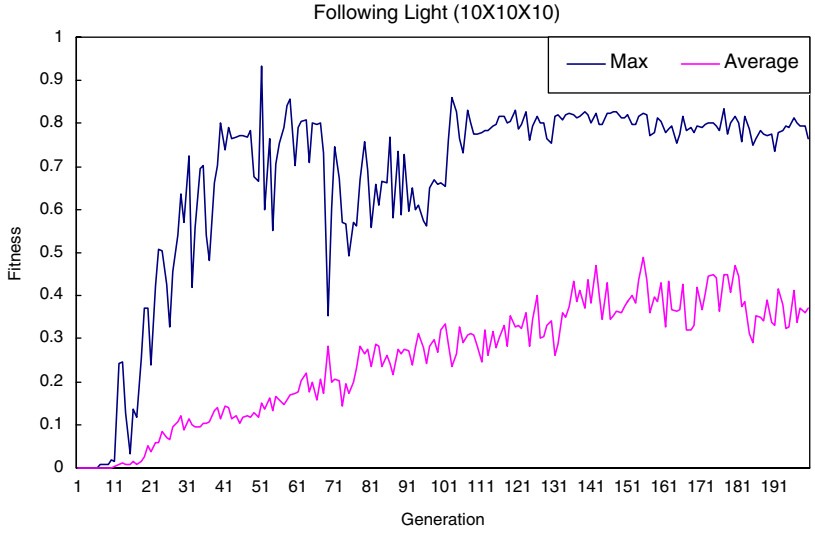

Fig. 9. The change of the best and average fitnesses ($10 \times 10 \times 10$ cellular automata).

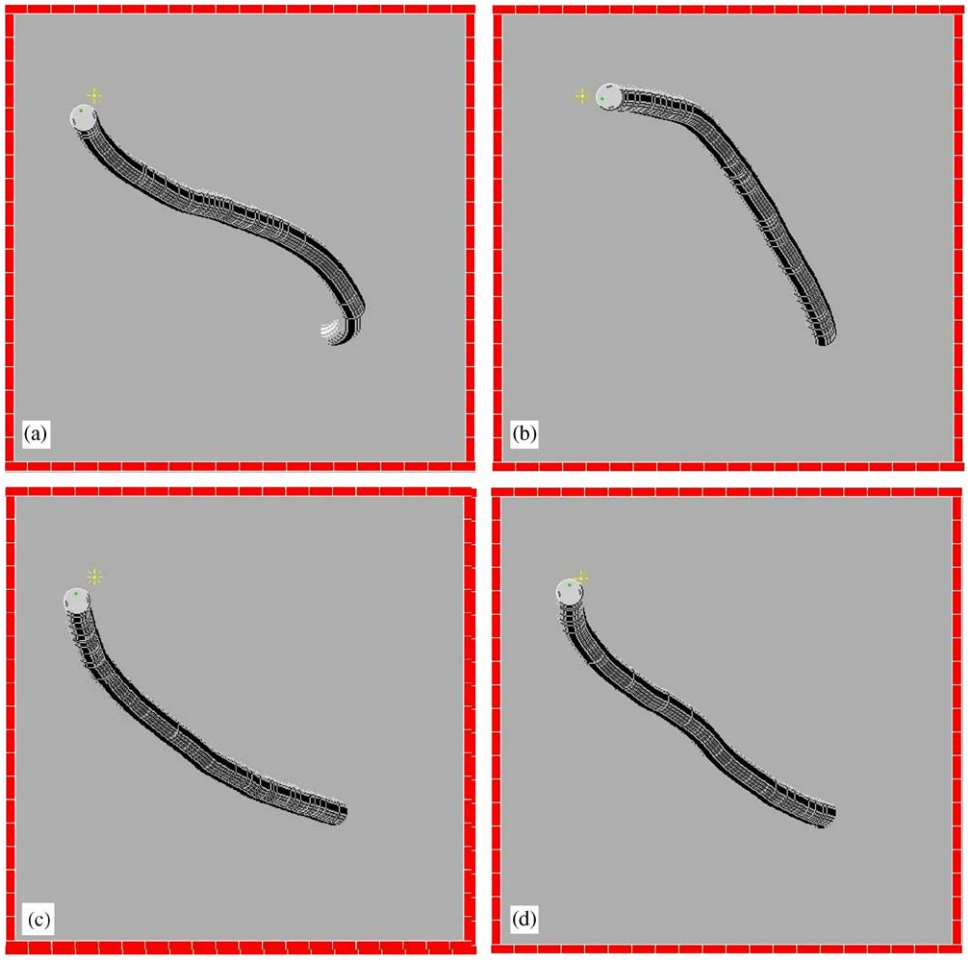

Fig. 10. The trajectory of a successful robot: (a) 0°, (b) 90°, (c) 180° and (d) 270°.

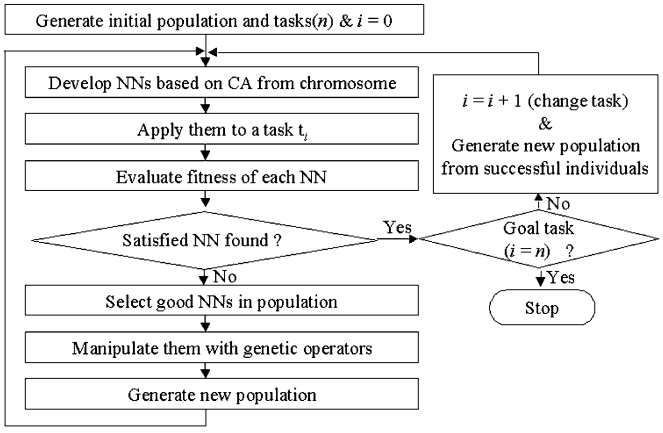

Fig. 11. The incremental evolution process of neural networks based on cellular automata.

5.1. Method

In this paper, we attempt to incrementally evolve a mobile robot controller to avoid bumping against obstacles. After determining environments, we evaluate population of the NNs in each environment. It is important to devise a nice method to generate new population for the next environment, while maintaining the behaviors learned at the previous environments. The simplest way is to copy a successful individual (this method is called manual selection) obtained in the previous step into all the individuals in the next population and then mutate them. This can maintain good features of successful individuals in the previous step but new individuals tend to heavily depend on it.

Gomez and Mikkulainen [11] have solved it by applying delta-coding [15] to escape local minima. Delta coding is an iterative genetic search strategy that sustains search by periodically re-initializing the population. In manual selection, we choose the best individual for the seed of the next population. In this paper, we select several successful individuals according to the fitness value (this method is called automatic selection) and then generate new population by mutating them with a modified delta-coding method. The number of mutations in each individual is decided by Cauchy distribution to distribute new individual properly. The values that represent differences from the best solution need to be generated using a probability density function that concentrates most

of the values in the local search space while occasionally permitting values of much larger magnitude. Cauchy distribution is defined as follows:

$$f(x) = \frac{\alpha}{\pi(\alpha^2 + x^2)}. \quad (3)$$

With this distribution, the value of f will fall within the interval $\pm\alpha$ with 50% probability. This makes many individuals similar to the individual obtained in the previous step and a few individuals very different to them, due to the shape of Cauchy probability density function. In the simulations, this distribution is used to decide the number of mutations on the current population.

Because the robot behavior is composed of going straight and turning left and right, the controller can be incrementally evolved to do those behaviors step by step. The environments in Fig. 12(a) and (b) make the robot to go straight and those in Fig. 12(c)–(e) to turn right, and those in Fig. 12 (f) and (g) to turn left. Fitness is evaluated based on the velocity of the robot and the number of straight movements

$$Fitness = 50 \times \left(\frac{1}{S} \sum_{i=0}^{S} V_i \right), \quad (4)$$

where S, is the number of movements until stopping and V_i, the value determined by the velocity in the ith step.

This fitness function causes the robot to arrive at goal position quickly without bumping against walls. Population size is 50 and the fitness of individual is computed by the average of four runs.

5.2. Simulation results

Fig. 13(a) shows the change of the best fitness when evolving in the environment of Fig. 12(g). As can be seen, the direct evolution has difficulty producing a successful controller for this task easily. Because all the individuals are poor in performance for the difficult task, simple GA gets trapped in an unpromising region of the solution space. An easier task might induce good individuals which leads to a promising region of the space. In this simulation, successful controller did not emerge after 100 generations. Fig. 13(b) shows the result of the incremental evolution with manual selection of the best individual in each step. In the figure, the goal of evolution is to find the successful solution for the environment in Fig. 12(g). At first, six generations are passed to generate the successful solution for the environment in Fig. 12(a). If the average fitnesses of the population is larger than the pre-defined threshold, it stops. The next generation for more difficult problem is determined manually. For Fig. 12(c), (e), and (f), it takes only one generation because the first population is enough for the problem. To achieve the final goal, more than 100 generations are needed.

Fig. 13(c) shows the result of the incremental evolution with automatic selection of the best individuals and mutations using Cauchy distribution. In the figure, the distribution of the next population for the more difficult problem is determined automatically. The stop criterion is the same as that of manual selection. It needs only 41 generations for achieving the final goal. In these figures, we can see the efficiency of incremental evolution for solving a complex problem. Fig. 14 shows the architectures of the evolved NNs in each step. Fig. 14(a)–(d) shows only the NN architectures evolved in the period of (a), (b), (e) and (g) in Fig. 12, respectively, because Fig. 12(c)–(e) can be solved by the successful individual obtained in the previous step. We can see that the architectures of NNs get complicated as the step goes on.

In Fig. 14(a), input from the front sensors (neuron labeled as "6") has a major role for the movement. In Fig. 14(c), the right wheel needs to decrease velocity and

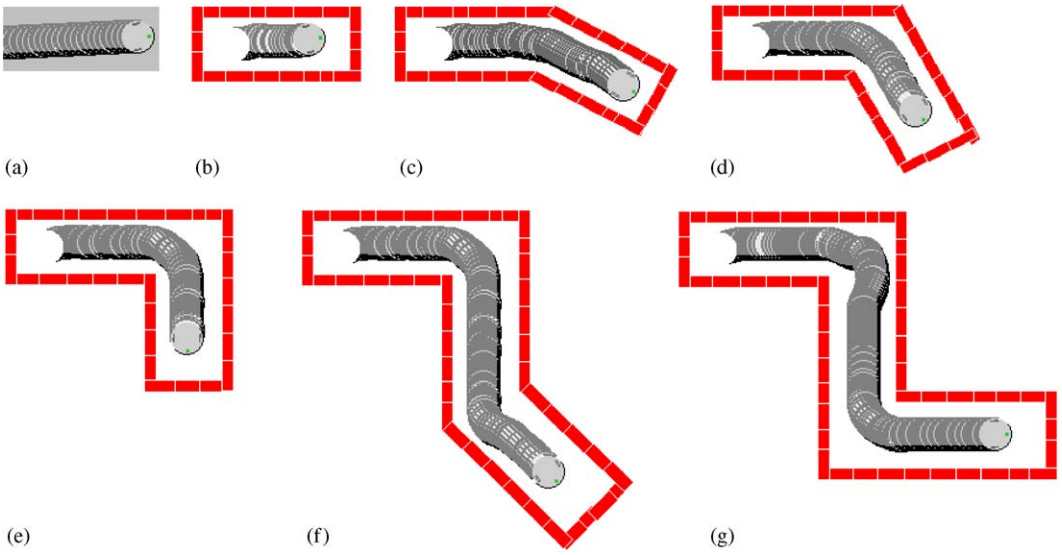

Fig. 12. Environments used in incremental evolution.

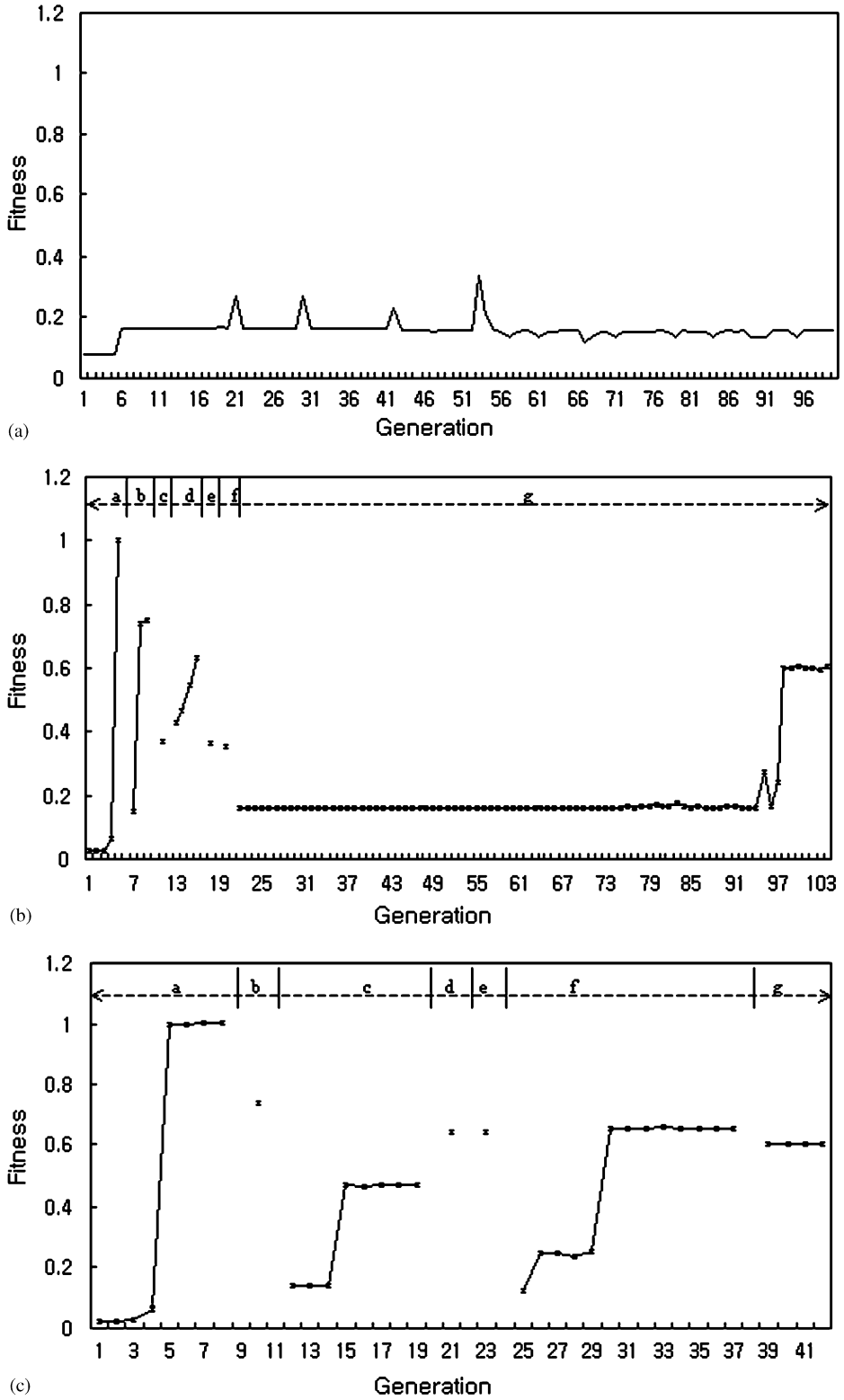

Fig. 13. (a) The change of the best fitness when evolving in Fig. 12(g) environment; (b) the change of the best fitness of the incremental evolution with manual selection of the best individuals in each step; (c) the change of the best fitness of the incremental evolution with automatic selection of the best individuals and mutations using Cauchy distribution.

the left wheel needs to increase for turning right. As a result, neuron for the right wheel has two inhibitory connections for decreasing speed but the neuron for the left wheel has one inhibitory link. In the case of Fig. 14(d), both turning left and turning right behaviors are needed to move correctly and the architecture of the network is difficult to understand because of the complexity of the architecture. Fig. 15 shows the results of applying the

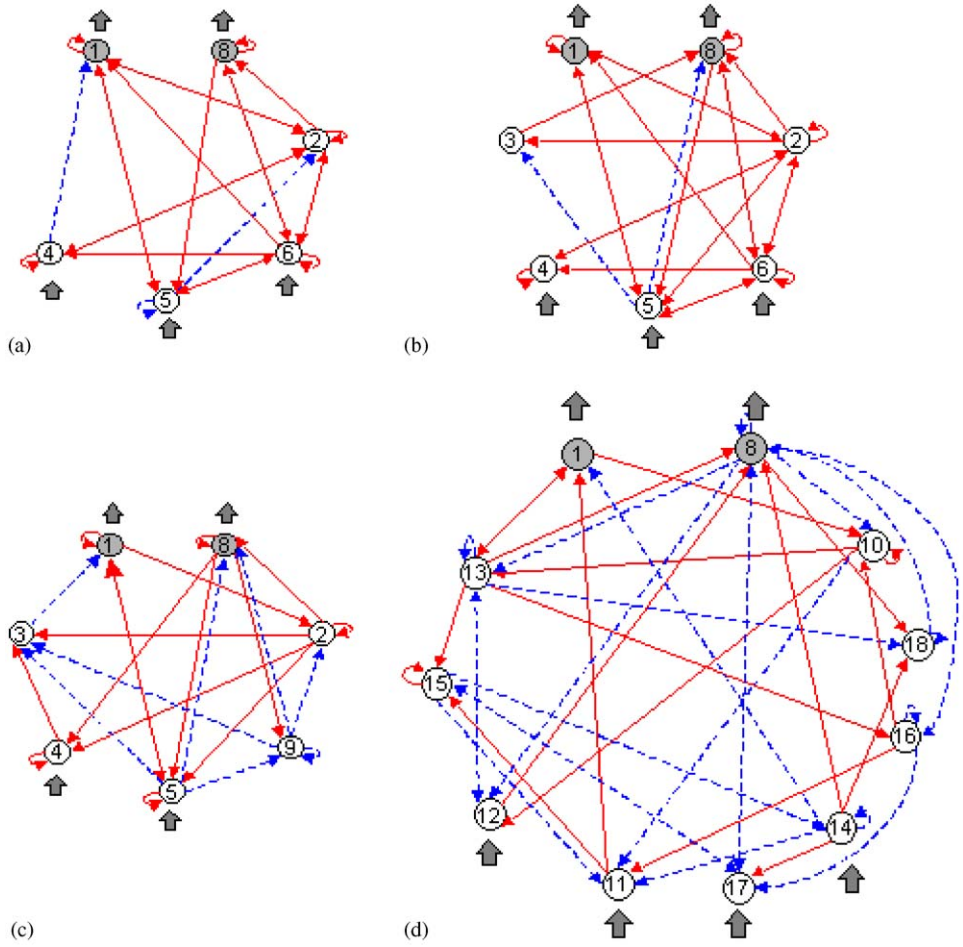

Fig. 14. The neural network architectures in each step (dotted line: inhibitory connection, solid line: excitatory connection).

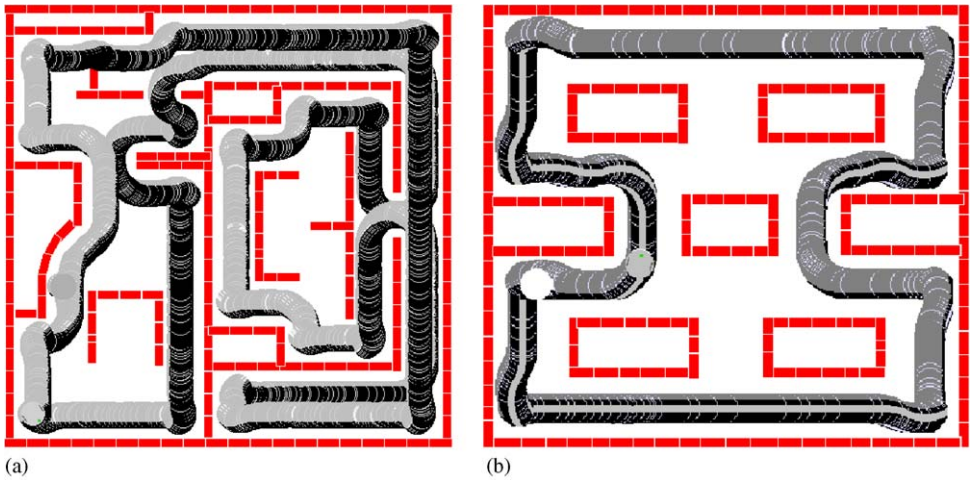

Fig. 15. Applying a successful robot to different environments.

individual obtained by the incremental evolution method to more complex environments.

Fig. 16 shows the simulation results for an improved following light behavior. In this simulation, the configuration of the environment is relatively more complex than that of "avoiding obstacle," and "following light." Like other simulations, the map information of the environment is not used and there is no exact information about which way is the right one (only light information is used as a clue). In this case, the initial population is generated

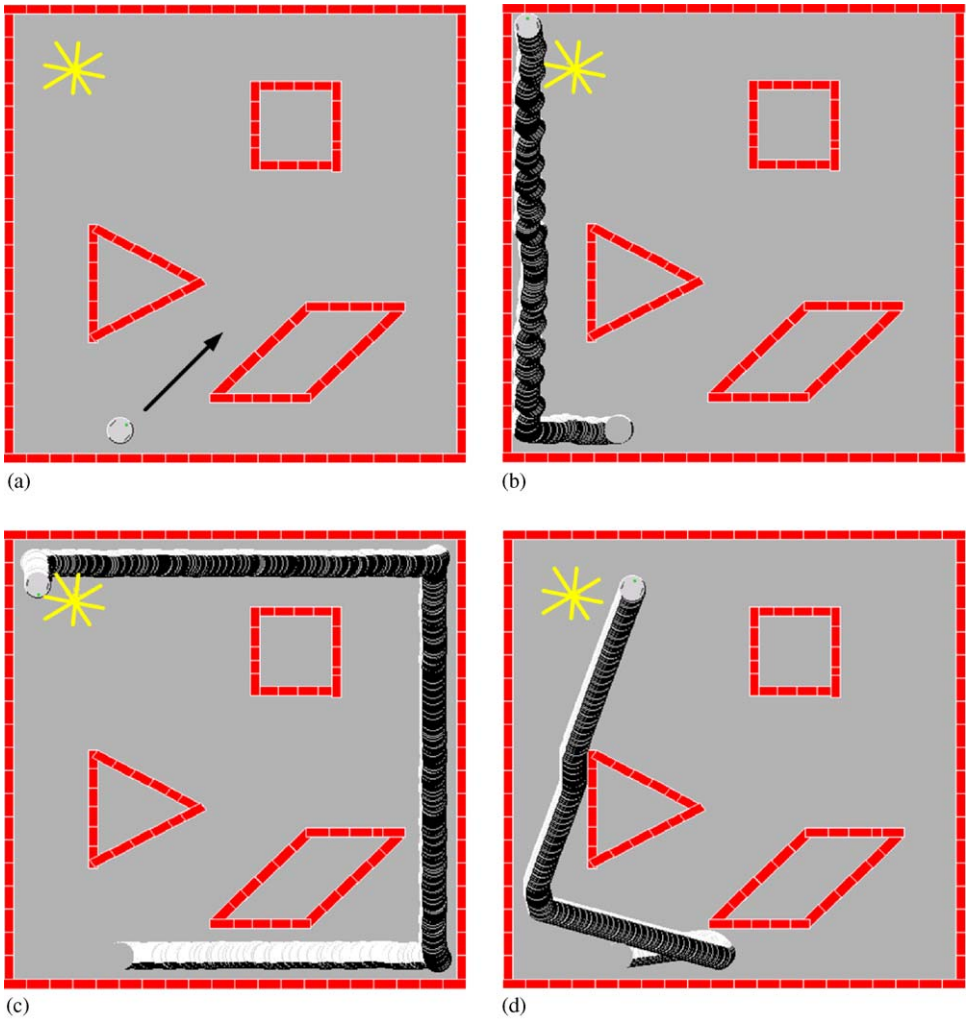

Fig. 16. Evolution of improved following light behavior for simple environment with relatively simple obstacles. The initial population for the evolution is automatically generated from the most successful individual for Fig. 12(g). The initial setting of the CA-based neural network is modified to input the light signal to the network because the previous behavior (avoiding obstacle) does not need the sensor values. The robot searches for the light source with reduced bumps using the behavior: (a) simulation environment (star mark represents the light source and the arc in (a) represents the initial orientation of the robot), (b) an individual in the initial population which causes many bumps to get the light source but takes a shortcut to the destination, (c) an individual in the initial population which takes long time, (d) successful one.

automatically using the most successful one for Fig. 12(g). It can only avoid obstacles well and has no information on following light behavior. Finally, the individual which finds shortcuts (robot can infer the way by using the light level information) with relatively less bumps emerges.

6. Multi-module integration

The controller composed of only one NN module has a difficulty in making the robot perform complex behaviors. Some researchers attempt to overcome this shortcoming by combining several modules evolved or programmed to do simple behaviors such as "going straight," "avoiding obstacles," "seeking object," and so on [14,18]. They expect the controller combined with several modules to do complex behaviors.

6.1. Method

Combining low and simple behaviors makes high-level complex behaviors. In this section, basis behaviors and the IF–THEN rules for combining them are presented. Four basis behaviors used in this paper are defined as follows:

- Battery recharge: If a robot arrives at battery recharge area, battery is recharged. This module enables the robot to operate as long as possible.
- Following light: The robot goes to stronger light. It must operate this module to go to the 'battery recharge' area because the light source exists in that area.
- Avoiding obstacles: If the obstacles exist around the robot, it avoids obstacles without bumping against

them. This module enables it to go to 'Battery Recharge' area safely by avoiding obstacles.
- Going ahead: If there is nothing around the robot, it goes ahead: This module makes it move continuously without stopping.

These basis behaviors are needed to adapt to a given environment shown in Fig. 17. The robot must avoid bumping against obstacles and go to the "Battery Recharge" area for recharging battery occasionally in order to be alive for a long time in a given environment.

We use IF–THEN rules for combining the four basis behaviors properly. We can determine an operating module according to the situation which is judged by sensor values of the robot using IF–THEN rules. We expect that this method adapts to a given environment if the rules are defined properly. The rules used in this paper are as follows:

IF ("Battery Recharge" area)
 THEN Execute Battery Recharge module
 ELSE IF (Battery sensor $<\alpha$) AND (Minimum value of light sensors $\leqslant\gamma$)
  THEN IF (Maximum value of distance sensors $\leqslant\beta_1$)
   THEN Execute Following Light module
   ELSE Execute Avoiding Obstacles module
 ELSE IF (Maximum value of distance sensors $\leqslant\beta_2$)
  THEN Execute Going Ahead module
  ELSE Execute Avoiding Obstacles module

The robot moves differently according to the constant values (α, β, γ). In these rules, constants are defined as follows:

- α: If battery sensor value is less than α, battery is needed recharging.
- β_1 and β_2: If the maximum value of distance sensors is greater than β_1 and β_2, the robot perceives obstacles.
- γ: If the minimum value of light sensors is less than γ, the robot perceives light.

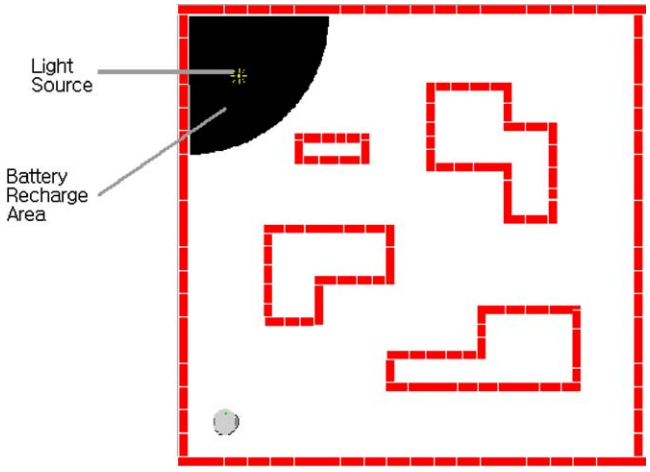

Fig. 17. Simulation environment.

The robot moves differently according to these constant values. For example, the robot moves around "Battery Recharge" area if α is large, while the robot consumes the battery before reaching the "Battery Recharge" area if it is small. Also, if β is large, the robot bumps against the obstacles frequently and if γ is very small, the robot far from light source cannot perceive "Battery Recharge" area.

6.2. Simulation results

This approach is simulated in a modified Khepera simulator. There are two kinds of modules. One is programmed modules such as "Battery Recharge" and "Going Ahead," and the other is evolved on CA such as "Following Light" and "Avoiding Obstacles."

"Battery Recharge" module is programmed because this behavior is very simple. Algorithm of this module is that the robot recharges battery if it is in the "Battery Recharge" area shown in Fig. 17.

The operating module is selected by the rules explained previously and sensor values of the robot. Fig. 18 shows the trajectory of the robot by combined modules in a given environment. According to starting position of the robot, the trajectory and moving behavior are different. Battery decreases as the robot moves and battery becomes 2500 when it is recharged: the robot can move without recharging for 2500 times. α is $\frac{2}{3}$ of the maximum battery, β_1 is 200, and β_2 is 250. The value of the distance sensor ranges from 0 to 1024, and the larger the value is, the closer the obstacles are. The β_1 and β_2 must be between 0 and 1024. For the safety of the robot, avoiding obstacle module is allowed to be selected for most cases but if the value of the sensor indicates that the robot is too far from the obstacles, another behavior can be selected. The definition of "too far" is empirically determined from many simulations and the values from 200 to 250 (approximately $\frac{1}{5}$ of the maximum value) are used. The β_1 and β_2 can be determined from the range.

Fig. 19 shows the trajectory of the robot by combined modules in a more difficult environment. The purpose of this simulation is to show the adaptivity and usefulness of the proposed method. In chaotic environment, there are many small obstacles but the basic structure of the environment and the goal of the robot are not changed. This implies that the pre-defined rules can be applied to the similar but more complex environments easily.

7. Concluding remarks

This paper has attempted to evolve neural networks based on cellular automata to control an autonomous mobile robot. In particular, in order to remedy the shortcoming of conventional evolutionary neural networks and obtain the sensory-motor controller doing complex and general behaviors, we have combined multi-modules evolved or incrementally to do simple behaviors using

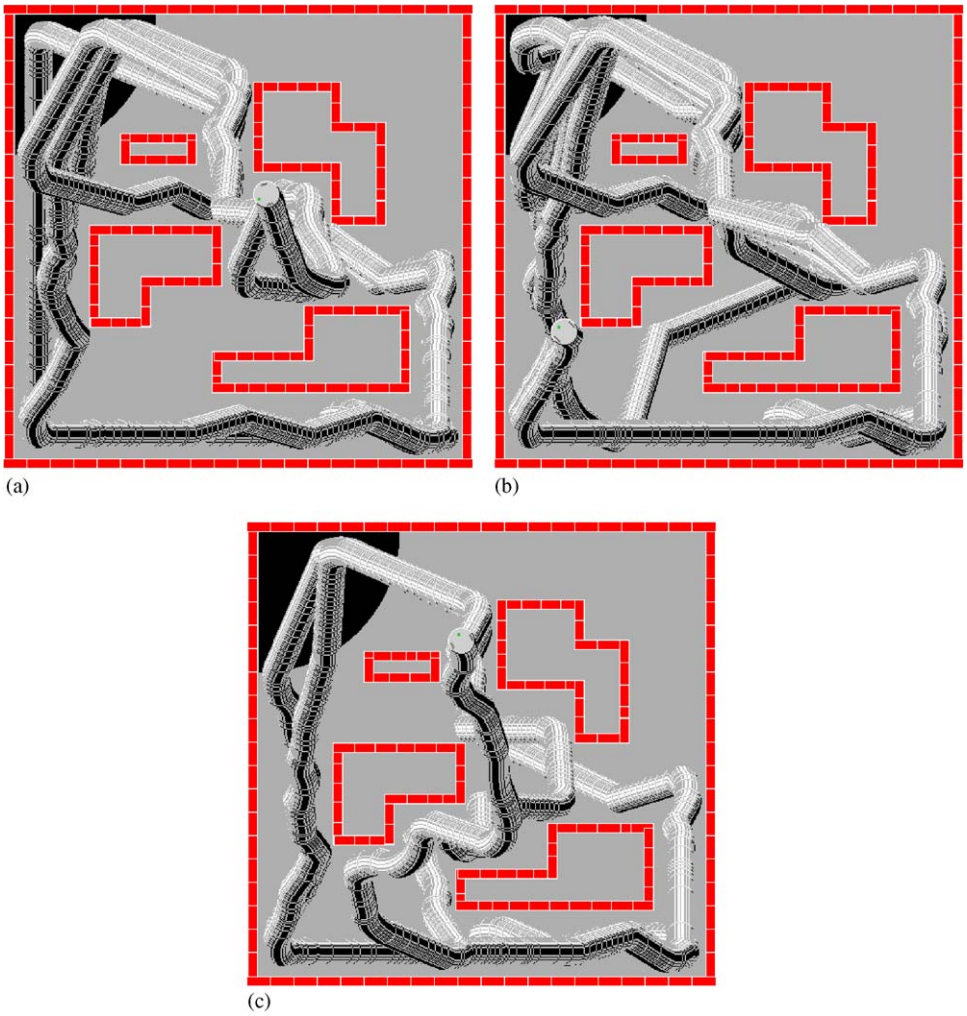

Fig. 18. The trajectories of the robot with multi-modules.

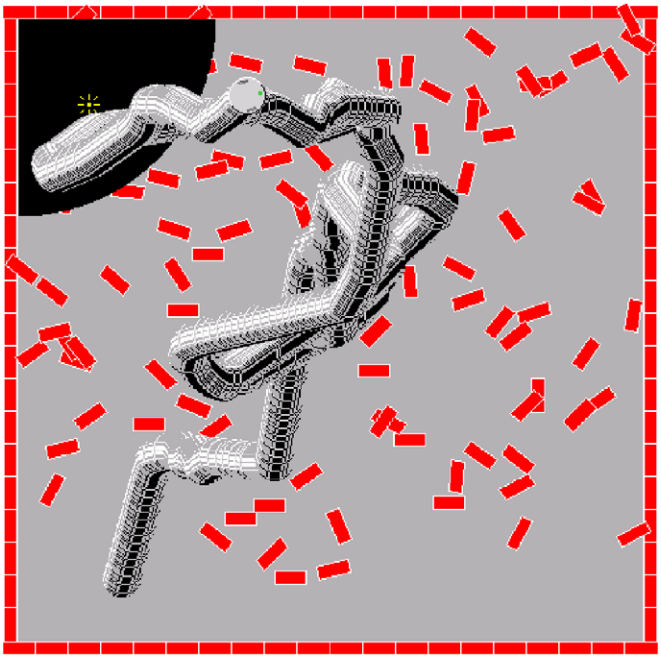

Fig. 19. The trajectory of the robot in chaos environment with multi-modules.

IF–THEN rules to generate complex behaviors. We hope that the sophisticated techniques proposed in this paper make the evolutionary algorithms to be used in larger variety of applications. For the scalability of the proposed method, automatic rule generation or sophisticated action selection mechanism is needed. Contribution of this paper lies in the investigation of applicability of cellular automata based neural network for the real-world applications. The proposed systematic procedure to generate high-level behavior can be used in the evolutionary robotics area.

This paper has reported ongoing efforts for generating complex high-level behavior using systematic approach in the context of CA-based NNs. Because it is at the initial stage of the research, the behaviors evolved or generated seem relatively simple and need to be improved. At this stage, we have attempted to show the potential of the proposed methods in relatively simple problems. The extension of the works using sophisticated module combination methods and evolving more complex behavior will be the next step. The increase of complexity for module combination using action selection mechanism is an ongoing research and a preliminary result is reported in another work [12].

Acknowledgement

This research was supported by Brain Science and Engineering Program sponsored by Korea Ministry of Commerce, Industry and Energy.

References

[1] G.J. Barlow, C.K. Oh, E. Grant, Incremental evolution of autonomous controllers for unmanned aerial vehicles using multi-objective genetic programming, Proceedings of the 2004 IEEE Conference on Cybernetics and Intelligent Systems, 2004, pp. 688–693.
[2] S.B. Cho, G.B. Song, Evolving CAM-brain to control a mobile robot, Appl. Math. Comput. 111 (2–3) (2000) 147–162.
[3] H. de Garis, M. Korkin, The CAM-brain machine (CBM) an FPGA based hardware tool which evolves a 1000 neuron net circuit module in seconds and updates a 75 million neuron artificial brain for real time robot control, Neurocomputing 42 (1–4) (2002) 35–68.
[4] H. de Garis, M. Korkin, G. Fehr, The CAM-brain machine (CBM): an FPGA based tool for evolving a 75 million neuron artificial brain to control a lifesized kitten robot, Auton. Robot. 10 (3) (2001) 235–249.
[5] H. de Garis, M. Korkin, P. Guttikonda, D. Cooley, Simulating the evolution of 2D pattern recognition on the CAM-Brain machine, an evolvable hardware tool for building a 75 million neuron artificial brain, Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks, vol. 6, 2000, pp. 606–609.
[6] D. Floreano, F. Mondada, Evolution of homing navigation in a real mobile robot, IEEE T. Syst. Man Cybern. 26 (3) (1996) 396–407.
[7] D. Floreano, F. Mondada, Evolutionary neurocontrollers for autonomous mobile robots, Neural Networks 11 (7–8) (1998) 1461–1478.
[8] GenoByte Inc. ROBOKONEKO (Kitten Robot), ⟨http://www.genobyte.com/robokoneko.html⟩.
[9] F. Gers, H. de Garis, M. Korkin, CoDi-1Bit: a simplified cellular automata based neural model, Proceedings of the Conference on Artificial Evolution, Nimes, France, October 1997, pp. 315–334.
[10] D.E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley, Reading, MA, 1989.
[11] F. Gomez, R. Mikkulainen, Incremental evolution of complex general behavior, Adapt. Behav. 5 (3–4) (1997) 317–342.
[12] K.-J. Kim, S.-B. Cho, Integration of multiple neural networks evolved on cellular automata by action selection mechanism, Seventh International Conference on Neural Information Processing vol. 2, 2000, pp. 687–692.
[13] K-Team, Khepera Simulator Version 5.02 User Manual, 1999.
[14] H.H. Lund, Modern artificial intelligence for human-robot interaction, Proc. IEEE 92 (11) (2004) 1821–1838.
[15] K.E. Mathias, L.D. Whitley, Initial performance comparisons for the delta coding algorithm, Proceedings of IEEE Conference on Evolutionary Computation, vol. 1, 1994, pp. 433–438.
[16] S. Nolfi, D. Floreano, Synthesis of autonomous robots through evolution, Trends Cog. Sci. 6 (1) (2002) 31–37.
[17] G.B. Parker, L. Zhiyi, Evolving neural networks for hexapod leg controllers, Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems, vol. 2, 2003, pp. 1376–1381.
[18] P. Rusu, E.M. Petriu, T.E. Whalen, A. Cornell, H.J.W. Spoelder, Behavior-based neuro-fuzzy controller for mobile robot navigation, IEEE Trans. Instrum. Meas. 52 (4) (2003) 1335–1340.
[19] J. Santos, R.J. Duro, J.A. Becerra, J.L. Crespo, F. Bellas, Considerations in the application of evolution to the generation of robot controllers, Inform. Sci. 133 (3–4) (2001) 127–148.
[20] T. Smith, P. Husbands, M. O'shea, Local evolvability of statistically neutral GasNet robot controllers, Biosystems 69 (2–3) (2003) 223–243.
[21] G.-B. Song, S.-B. Cho, Rule-based integration of multiple neural networks evolved based on cellular automata, FUZZ-IEEE'99, vol. 2, 1999 pp. 791–796.
[22] B. Ster, An integrated learning approach to environment modelling in mobile robot navigation, Neurocomputing 57 (2004) 215–238.
[23] T. Toffoli, N. Margolus, Cellular Automata Machine, MIT Press, Cambridge, MA, 1987.
[24] M. Walker, Comparing the performance of incremental evolution to direct evolution, Second International Conference on Autonomous Robots and Agents, 2004, pp. 119–124.
[25] M. Walker, Mutated seeds: a performance enhancer for incremental evolution, Proceedings of the First Postgraduate Conference of the Institute of Information and Mathematical Sciences, Massey University, 2004, pp. 95–99.
[26] R. Walker, O. Miglino, Simulating exploratory behavior in evolving artificial neural networks, Proceedings of the Genetic and Evolutionary Computation Conference 1999, pp. 122–1428.
[27] X. Yao, Evolving artificial neural networks, Proc. IEEE 87 (9) (1999) 1423–1447.

Kyung-Joong Kim (Student Member, IEEE) received the B.S. and M.S. degrees in computer science from Yonsei University, Seoul, Korea, in 2000 and 2002, respectively. Since 2002, he has been a Ph.D. student in the Department of Computer Science, Yonsei University. His research interests include evolutionary neural network, robot control, and agent architecture.

Sung-Bae Cho (Member, IEEE) received the B.S. degree in computer science from Yonsei University, Seoul, Korea, in 1988 and the M.S. and Ph.D. degrees in computer science from Korea Advanced Institute of Science and Technology (KAIST), Taejeon, Korea, in 1990 and 1993, respectively.

From 1991 to 1993, he worked as a Member of the Research Staff at the Center for Artificial Intelligence Research at KAIST. From 1993 to 1995, he was an Invited Researcher of Human Information Processing Research Laboratories at ATR (Advanced Telecommunications Research) Institute, Kyoto, Japan. In 1998, he was a Visiting Scholar at University of New South Wales, Canberra, Australia. Since 1995, he has been a Professor in the Department of Computer Science, Yonsei University. His research interests include neural networks, pattern recognition, intelligent man–machine interfaces, evolutionary computation, and artificial life.

Dr. Cho is a Member of the Korea Information Science Society, INNS, the IEEE Computer Society, and the IEEE Systems, Man and Cybernetics Society. He was awarded outstanding paper prizes from the IEEE Korea Section in 1989 and 1992, and another one from the Korea Information Science Society in 1990. In 1993, he also received the Richard E. Merwin prize from the IEEE Computer Society. In 1994, he was listed in Who's Who in Pattern Recognition from the International Association for Pattern Recognition and received the best paper awards at International Conference on Soft Computing in 1996 and 1998. In 1998, he received the best paper award at World Automation Congress. He was listed in Marquis Who's Who in Science and Engineering in 2000 and in Marquis Who's Who in the World in 2001.