Appl Intell (2006) 25:253–268
DOI 10.1007/s10489-006-0106-z

A unified architecture for agent behaviors with selection of evolved neural network modules

Kyung-Joong Kim · Sung-Bae Cho

Abstract To model complex systems for agent behaviors, genetic algorithms have been used to evolve neural networks which are based on cellular automata. These neural networks are popular tools in the artificial life community. This hybrid architecture aims at achieving synergy between the cellular automata and the powerful generalization capabilities of the neural networks. Evolutionary algorithms provide useful ways to learn about the structure of these neural networks, but the use of direct evolution in more difficult and complicated problems often fails to achieve satisfactory solutions. A more promising solution is to employ incremental evolution that reuses the solutions of easy tasks and applies these solutions to more difficult ones. Moreover, because the human brain can be divided into many behaviors with specific functionalities and because human beings can integrate these behaviors for high-level tasks, a biologically-inspired behavior selection mechanism is useful when combining these incrementally evolving basic behaviors. In this paper, an architecture based on cellular automata, neural networks, evolutionary algorithms, incremental evolution and a behavior selection mechanism is proposed to generate high-level behaviors for mobile robots. Experimental results with several simulations show the possibilities of the proposed architecture.

Keywords Cellular automata · Neural networks · Evolutionary algorithm · Incremental evolution · Behavior selection mechanism · Mobile robot control

K.-J. Kim · S.-B. Cho (✉)
Department of Computer Science, Yonsei University,
134 Shinchon-dong, Sudaemoon-ku, Seoul 120-749, Korea
email: sbcho@cs.yonsei.ac.kr

K.-J. Kim
e-mail: kjkim@cs.yonsei.ac.kr

1 Introduction

The study of evolving cellular automata with emergent properties is one of the most interesting research areas in the artificial life community [1]. Cellular automata are said to be emergent because they are based on basic cell interaction instead of on the regulation of global behavior. If we know the rules for specific global behaviors, we can generate very complex patterns easily. In the field of artificial life, life is regarded as a series of patterns that emerge from the local interaction of cells. However, in many cases, there are a large number of these candidate rules for specific behaviors. It is impossible to enumerate all of these exhaustively. One solution to this problem is genetic learning of the cellular automata rules.

Crutchfield has evolved cellular automata for basic computational functions such as density classification and synchronization [2]. His work shows the possibility of using cellular automata for computations in scientific applications, but it is questionable that these results can be confidently used in engineering applications. If the cellular automata are able to represent a general computational model (like a neural network), the power of the computation can be increased and there might be some practical applications. De Garis's work is an example of how to represent neural networks in cellular automata [3]. He uses a special type of hardware called CAM-Brain to update states very quickly and generates a brain-like system with millions of different behaviors. His research group developed ways to use the machine for multiple timers, pattern detectors, and motion detectors, but these were tested on very simple tasks and did not show significant practicality [4].

To solve the engineering problems of designing agent behaviors in hardware and software, a systematic integration with several competitive techniques is needed. An extension of the basic hybrid model of cellular automata and neural networks can lead to a computational architecture for high-level tasks. If the given search space is extremely large, evolutionary algorithms often fail to find satisfactory solutions in a reasonable time. In real-world problems, the computational cost of fitness evaluation in evolution is very high. This can lead to heavy losses of time and computational resources. Evolving only one kind of behavior for the performance of a complex task is inefficient and sometimes even impossible.

An incremental approach to evolution is more promising than a direct one. This system has more opportunities to become adaptable because it evolves through many levels of problems. In our proposed architecture, human beings decompose high-level tasks into multiple small tasks and each behavior is learned incrementally. In many cases, the sequencing of these basic behaviors cannot be defined explicitly. This sequencing requires a flexible behavior selection mechanism with which humans can deal. In the proposed architecture, a behavior selection network [5] that continuously changes the activation level of behaviors through the selection procedure is adopted for flexibility. Figure 1 shows the proposed method composed of cellular automata, neural networks, incremental evolution and behavior selection. In this paper, the proposed architecture was tested in relation to robot control.

This paper is organized as follows. In Section 2, related works are briefly described. Section 3 describes the basic components of the proposed architecture. In Section 4, results that were gathered in experiments with a mobile robot are presented.

2 Related works

Computational structures for agent behaviors are divided into four categories: reactive control, deliberative control, behavior-based control and hybrid control [6]. Mali reviewed the foundations, limitations, and achievements of a number of autonomous agent architectures including reactive control [7]. In the deliberative control category, constructing action plans for an agent operating in an environment containing uncertain and dynamic events is a difficult task, and requires a plan representation that contains both the rich

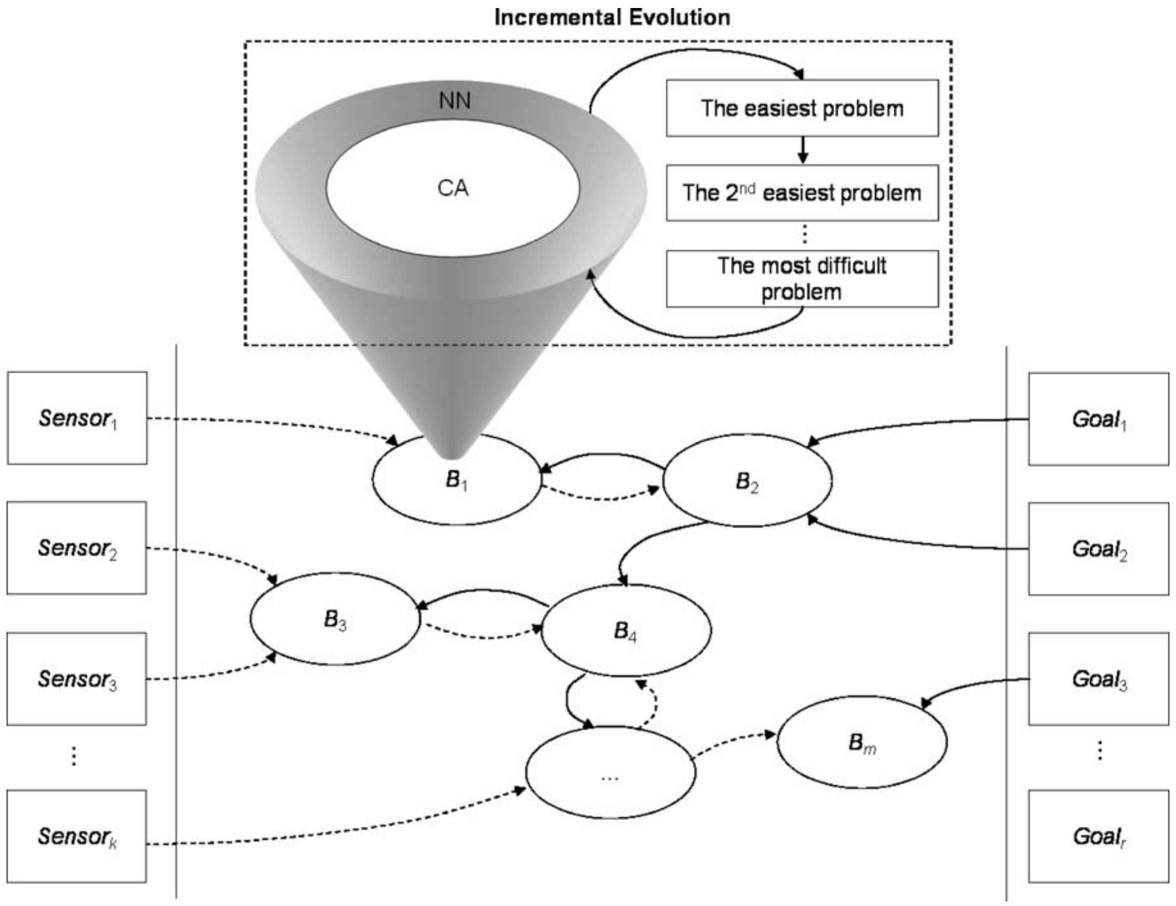


Fig. 1 An overview of the proposed method (*k*: the number of sensors, *r*: the number of goals, and *m*: the number of behaviors) (CA: Cellular automata, NN: Neural networks)

vocabulary of loops, conditionals, sensory requests and concurrency necessary to represent flexible agent control programs [8]. In the behavior-based control category, high-level behavior emerges from the interaction of primitive behaviors that are reactive to sensory inputs. Primitive behaviors with different goals may produce conflicts, and the formulation of effective mechanisms for coordination of these behaviors' activities is a major control issue. Numerous behavior selection mechanisms have been proposed over the last decade [9]. Because both of these (reactive and plan-based) approaches have their specific pros and cons, the hybrid agent deliberative-reactive architecture has been developed with the positive properties of each of the earlier methods.

In an imitation of the cognitive architecture of the human brain, a new type of agent architecture was proposed and a robot soccer team underlying this framework was established [10]. To model complex systems, software agents need to combine cognitive abilities and reactive abilities. Guessoum proposed an operational hybrid agent model which mixes well-known paradigms (objects, actors, production rules and ATN) and real-time performances [11]. There are some works related to these kinds of hybrid agent architectures which use the combination of reactive control and deliberative planning [12–14].

The evolutionary approach is the attempt to develop agent behaviors through a self-organized process based on artificial evolution [15]. Lee proposed decomposing the overall task to fit in with the behavior-based control architecture, and then evolving the separate behaviors and arbitrators with an evolutionary approach [16]. To develop the basic reactive behaviors of the robot, the fuzzy classifier system was adopted for evolutionary learning [17]. Floreano addressed the issue of incremental evolution in two different experiments from the perspectives of changing environments and robot morphologies [18].

3 Basic components of the proposed architecture

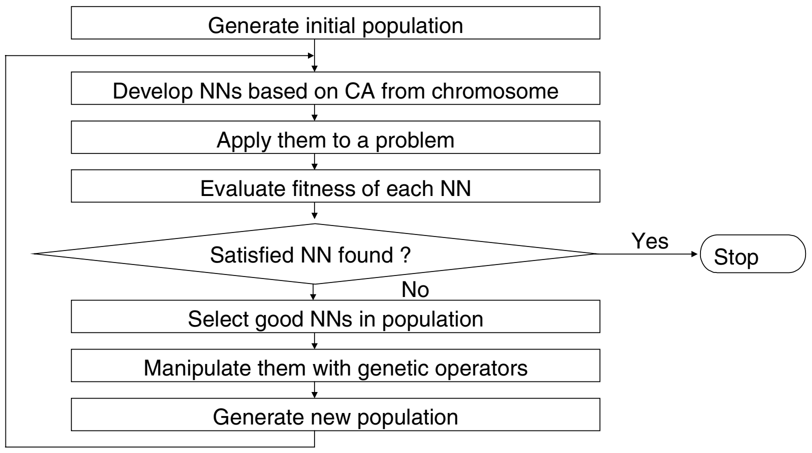
3.1 Neural networks based on cellular automata

Cellular Automata (CA) are populations of interacting cells. These cells are each computers (automatons) and can represent many kinds of complex behaviors by building appropriate rules [19]. Each cell has a state value and this value changes at each step. Change of state is based on the predefined rules and is also based on the current state of the cell and the conditions of the neighborhood cells. CA can model ecological systems or the behaviors of insects, and can be also used for image processing and the construction of neural networks.

A CA-based neural network structure composed of a blank, a neuron, an axon and a dendrite is grown inside a 2-D or 3-D CA-space by state, neighborhoods and rules encoded by the chromosome. If the cell state is blank, it represents an empty space and cannot transmit any signals. The neuron cell collects signals from the surrounding axon cells. The axon cell sends signals received from the neurons to the neighborhood cells. The dendrite cell collects signals from the neighborhood cells and passes them to the connected neuron in the end.

CA-based neural networks use evolutionary algorithms to optimize neural structures—one chromosome is mapped to one neural network. Therefore, with genetic algorithms working on this chromosome, it is possible to evolve and adapt the structure of the neural network for a specific task. Figure 2 shows the evolution process of a CA-based neural network.

Each chromosome leads to exactly one neural network. Figure 3 shows this chromosome representation. The model uses a distributing chromosome to encode its structure. This chromosome is initially distributed throughout the CA-space, so that every cell in the CA-space contains one chromosome



Generate initial population
Develop NNs based on CA from chromosome
Apply them to a problem
Evaluate fitness of each NN
Satisfied NN found ?
Yes
Stop
No
Select good NNs in population
Manipulate them with genetic operators
Generate new population

Fig. 2 Evolution process of a CA-based neural network

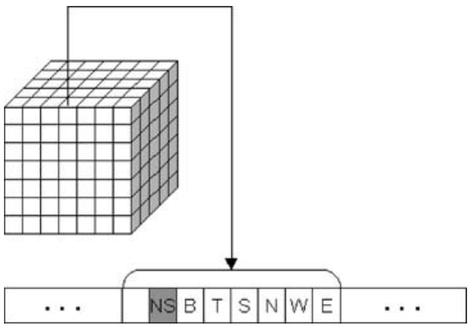


Fig. 3 Information encoded in the choromosome

instruction, i.e, one growth instruction, so that the chromosome belongs to the network as a whole. To represent the whole structure of a neural network, a chromosome has the same number of segments with the cells in the CA-space and each segment has information from each cell. A segment can change a blank cell to a neuron cell (NS bit), and decides the direction in which to send the received signals to the neighborhood cells (N, S, E, W, T and B bits). These signals can be only set to the direction in which the bit corresponds to 1. Each cell has the same form (logical representation) but has different functions which are defined by the characteristics of the cell components (the dendrite, the axon, the neuron and the empty cell.)

Each cell is represented as 7 bits (NS, B, T, S, N, W, E). If NS bit is 1, the cell is neuron. B (Bottom), T (Top), S (South), N (North), W (West), and E (East) represent the direction of growth signals. There are three different kinds of growth signals and they are excitatory axon, inhibitory axon, and dendrite growth signals. By the type of transmitted growth signals, the type of cell is determined. Only one bit among B, T, S, N, W, and E can be 1 and others are 0. The neighborhood cell in the direction (bit is 1) will get excitatory axon growth signal and the cell in the opposite direction will get inhibitory axon growth signal. Cells in other directions will get dendrite growth signals. If W bit is 1, a cell in the west direction is excitatory axon, a cell in the east direction is inhibitory axon and others are dendrite.

The growth phase organizes a neural structure and makes the signal trails among the neurons. First, a chromosome is randomly created and the states of all cells are initialized as blank. At this point, some of the cells are specified as neurons with some probability. A neuron cell sends axon and dendrite growth signals in the direction decided by the chromosome. An axon growth signal is sent in two directions (one is an excitatory axon signal and the other is an inhibitory axon signal), and a dendrite growth signal is sent in the remaining directions. Next, the blank cell receives a growth signal which changes to an axon or dendrite cell, according to the type of growth signal. This sends the signals received from the other cells in the direction that was determined by the chromosome. Finally, by repeating this process, the neural network is constructed. This means that the state of every cell changes no longer. Figure 4 shows the growing process in the 2-D CA-space.

The signaling phase transmits the signal from the input to output cells continuously. Signals are transmitted with the structure which was determined at the growth phase. First, the input cells produce the signal. This signal is sent to the faced axon cells which distribute it. Then, the neighborhood dendrite cells of other neurons collect and send the signal to the connected neurons. These neurons send the signal to the axon cells. Finally, the dendrite cells of the output neuron forward the signal to the output neurons. The output value can be obtained from the output neurons. The position of the input and output cells in the CA-space is decided in advance. Figure 5 shows the process of signaling after the neuron, axon and dendrite cells have been made.

3.2 The evolutionary algorithm

In general, a simple genetic algorithm generates a population of individuals and evolves them by using genetic operators such as selection, mutation, and crossover techniques [20]. We have used this genetic algorithm to search the optimal neural network. At first, half of the individuals that have better fitness value are selected to produce the new population. The genetic algorithm generates a new population from the fittest individuals. In the growth phase, the structure of the neural network is determined by the chromosome. During the signaling phase, fitness is evaluated by the output of the neural network. Two individuals in the new population are randomly selected and parts of them are exchanged by one-point crossover. Mutation is used in the segment of the chromosome.

However, when the problem is more difficult and the search space is very large, there is a need for modification of the basic evolutionary algorithm. The incremental approach is useful for two reasons: (1) learned behaviors have adaptive capabilities in a changing environment because they evolve from a simple environment to a complex one; and (2) when a solution cannot be found with direct evolution, incremental evolution can be an alternative.

If we use the movement of a robot as an example, the environments get more sophisticated when we compare straight movements to left and right turning movements. After the CA-based neural network module learns how to go straight, the successful chromosomes are copied to the next population. Then the robot evolves in the new environment to be able to go straight and also turn right. By repeating this process, the robot eventually evolves to the point of being able to go straight and also turn left and right. Efficient evolution is expected because of the reduced search space in each level of the incremental evolution.

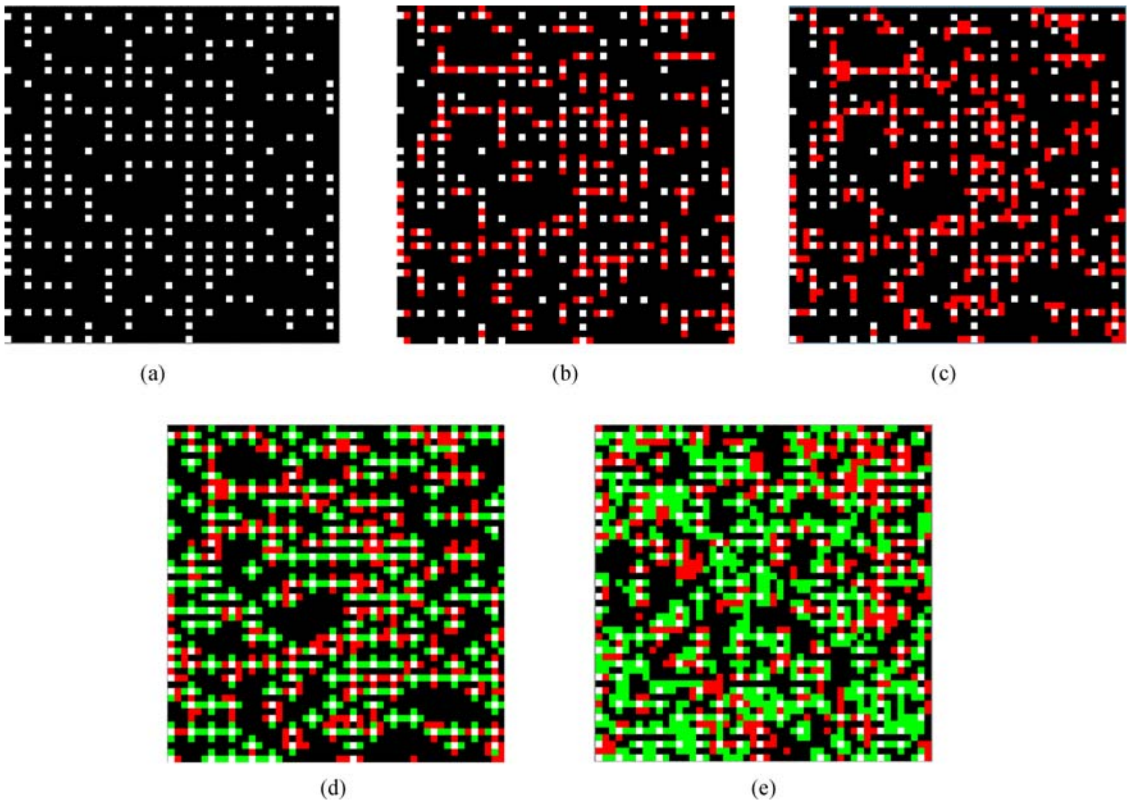


Fig. 4 The growth phase

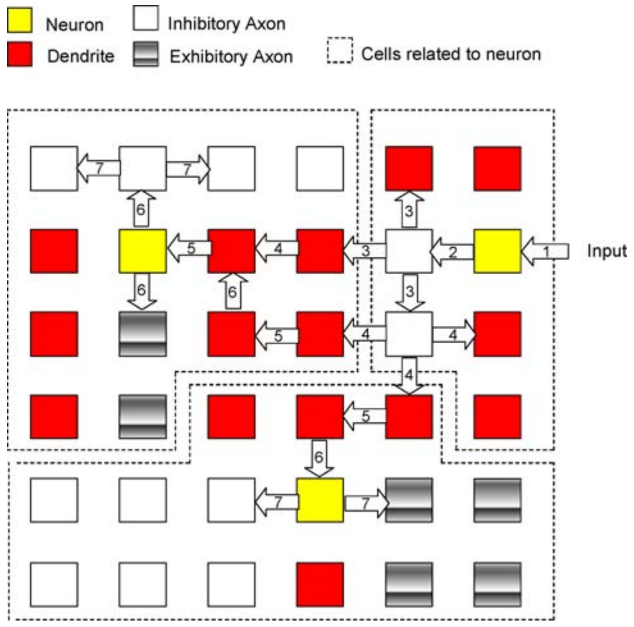


Fig. 5 The signaling phase

Incremental evolution is defined as follows. 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 an easier task than t_{i+1} for $0 < i \leq n$. Thus, the population evolves for task t_i and task t_{i+1}, and it also evolves for goal task, t_n, at the end [22]. It is expected to produce complex and general behaviors. In this process, the task becomes more difficult and the new population can be created from the most successful individuals using Cauchy distribution function [21] when the final candidate for the task is found. Figure 6 shows a procedure of incremental evolution for CA-based neural network.

3.3 The behavior selection mechanism

The behavior network is composed of nodes, environmental sensors, goals, and links. Each node has a set of pre-conditions (Fig. 7). These pre-conditions are logical conditions about the environment that are required to be true for the node to be executable. The "add" list consists of conditions about the environment that the node is likely to consider true. The "delete" list consists of conditions that are likely to be considered false. The final two components of the node are the activation level and the code that gets run after the node is executed. There are two types of links: internal links, which are used to connect nodes to other nodes, and external links, which are used to connect nodes to environmental sensors and goals.

The role of each element of the behavior network is as follows. Pre-conditions, post-conditions ("add" lists and "delete" lists) and executable codes regulate the properties of

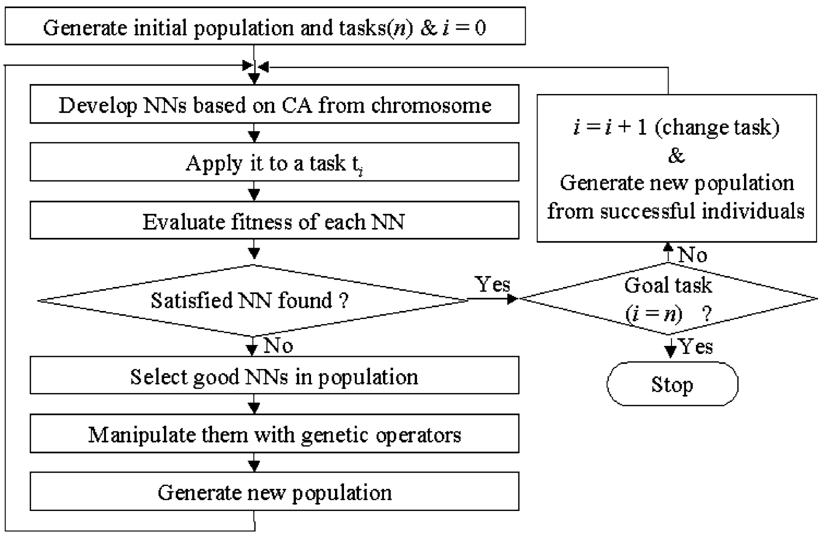

Generate initial population and tasks(n) & $i = 0$
Develop NNs based on CA from chromosome
Apply it to a task t_i
Evaluate fitness of each NN
Satisfied NN found ?
Yes
No
Select good NNs in population
Manipulate them with genetic operators
Generate new population
$i = i + 1$ (change task) & Generate new population from successful individuals
No
Goal task ($i = n$) ?
Yes
Stop

Fig. 6 Incremental evolution process of neural networks based on CA

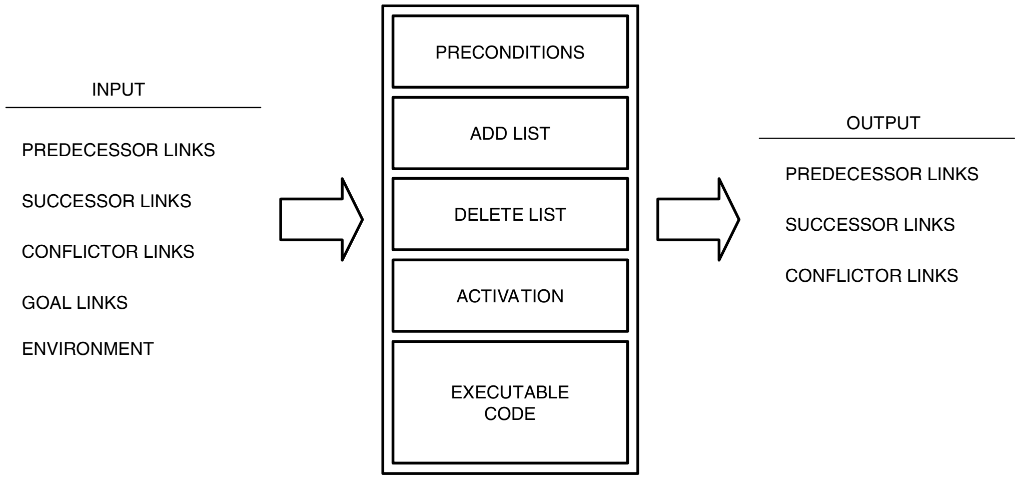


Fig. 7 Node structure in a behavior network

the behavior. To be executable, each behavior must fulfill all of the pre-conditions. For example, the "drinking water from a lake" behavior is meaningful when the lake is near. Such pre-conditions prevent the robot from executing the behavior in inappropriate situations. Because behaviors compete with each other to get the control of the robot, each behavior must estimate the effects of the execution of other kinds of behaviors. Environmental situations which exist after execution are defined as post-conditions and are used as a basis for establishing relationships between the behaviors. Executable codes define the specific control mechanisms of each behavior. The activation value indicates the level of applicability of the behavior for the current situation.

A node is executable if all its pre-conditions are true and its activation is greater than the global threshold (θ). If there are executable behaviors, the behavior with the highest activation is chosen. If there is no executable behavior, the global threshold is reduced by 10% and this process is repeated until an executable behavior is found. Several global parameters are used to tune the performance of the behavior selection mechanism to a particular environment. The mean activation value for each cycle (π) is used for normalization. The initial value of the global threshold (θ) is reduced by a given amount (e.g., 10%) for each cycle if no node is executable. A constant (ϕ) determines the weight of the environmental sensor inputs, as well as the weight of the successor links, and a constant (δ) determines the weights of the protected goal inputs and the conflictor links. The different inputs to a node are multiplied by the following weights (Fig. 8):

- environmental sensors by ϕ
- goals by γ
- protected goals by δ
- successor links by ϕ/γ
- predecessor links by ($\gamma/\gamma = 1$)
- conflictor links δ/γ

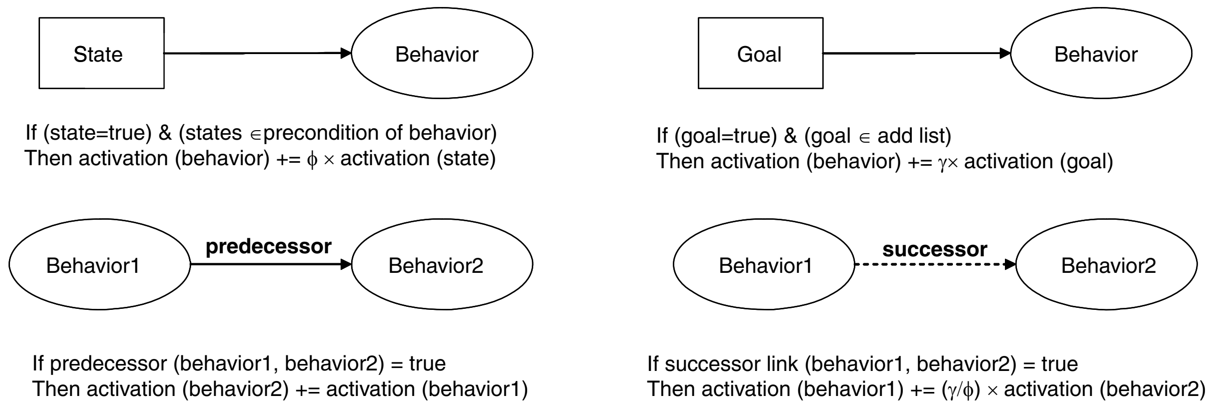

State
Behavior
If (state=true) & (states ∈precondition of behavior)
Then activation (behavior) += ϕ × activation (state)
Goal
Behavior
If (goal=true) & (goal ∈ add list)
Then activation (behavior) += γ× activation (goal)
Behavior1
predecessor
Behavior2
If predecessor (behavior1, behavior2) = true
Then activation (behavior2) += activation (behavior1)
Behavior1
successor
Behavior2
If successor link (behavior1, behavior2) = true
Then activation (behavior1) += (γ/ϕ) × activation (behavior2)

Fig. 8 Spreading activation

The weight can be adjusted based on the performance of the behavior network. Because the weight ϕ is multiplied to the value of environmental sensors, it can adjust the importance of the environmental conditions. Adjusting γ can control the effect of goals. Based on the ratio of ϕ/γ, the relevance of internal links are determined. If ϕ is increased, the behavior network is biased to the environmental sensors.

At the design stage, each behavior is represented as one node and a list of the nodes are defined using domain knowledge. Environmental sensors have abstract level states that can be observed in environments, and goals also have abstract level states that are related to the internal condition. Connections among behaviors are defined based on the information from the "add" list, the "delete" list, and the precondition. There are three types of internal connections—predecessor links, successor links, and conflictor links. As shown in Table 1, if an environmental sensor is in the precondition of nodes, there is a connection between the state and the node. If the execution of behavior is useful to achieve goals, there is a connection between the node and the goals. If the execution of behavior is not useful to a specific goal, there is a conflict connection between them.

The goal of the behavior network specifies the internal status of the robot. For humans and animals, body temperature, blood pressure, desire (motivation), or emotion can be examples of internal status. Because each behavior has no specific goal and is reactive to the environment, the long-term goal of the robot needs to be defined in the behavior network. In real robots, the battery level, speed of the wheel, emotional status (if it is modeled), or high-level definition of tasks can be described as goals of the network. In the model, there is no hierarchy of goals and they are all regarded as being on the same level. Many sub-goals are defined in the network and they can be conflicting.

The environmental sensor has a binary value (0 or 1) and the goals can have either binary or real values. The basic philosophy of behavior networks is competition among behaviors to take control of the agent. The activation level is also crucial to be executable. Each behavior propagates activation to other behaviors because its pre-condition is false and the state can be true by the execution of other behaviors (the state is in the "add" list of the behaviors). Each behavior can inhibit the execution of other behaviors by reducing activation levels. The procedure used to select a node and executed at each step is as follows and described in Figs. 8 and 9:

Table 1 Description of internal and external links

	Internal links
Predecessor link	If (proposition X is false) and (proposition X is a precondition of node A) and (proposition X is in the add list of node B), then there is an active predecessor link from A to B.
Successor link	If (proposition X is false) and (proposition X is in the add list of node A) and (proposition X is a precondition of node B) and (the node A is executable), then there is an active successor link from A to B.
Conflictor link	If (proposition X is true) and (proposition X is a precondition of node A) and (proposition X is in the delete list of node B), then there is an active conflictor link from A to B.
	External links
From sensors of the environment	If (proposition X about the environment is true) and (proposition $X$is a precondition of node A), then there is an active link from the sensor of the proposition X to node A.
From goals	If (goal Y has an activation greater than zero) and (goal Y is in the add list of node A), then there is an active link from the goal Y to node A.
From protected goals	If (goal Y has an activation greater than zero) and (goal Y is in the delete list of node A), then there is an active link from the goal Y to node A.

```
/* Select one action among candidates */
WHILE (1) {
    initialization();                              // clear candidate list
    spreading activation();                        // updates activation level
                                                   // 1st step: From environment sensors to behaviors
                                                   // 2nd step: From goals to behaviors
                                                   // 3rd step: Among behaviors
    normalization();                               // normalize activation level of behaviors
    FOR all behaviors {
      IF (all preconditions are true
            && activation (behavior) > threshold) {
                candidate (behavior);              // register to candidate list
      }
    }
    /* select one candidate behavior with the highest activation */
    IF (candidate () = NULL) {                     // there is no candidate behavior in the list
        threshold = 0.9 * threshold;               // decrease threshold
    }
    ELSE{
        select();
        break;
    }
}
```

Fig. 9 Pseudo code for the behavior selection mechanism

1. Calculate the excitation coming in from the environmental sensors and goals.
2. Propagate the activation level along the predecessor, successor, and conflictor links.
3. Normalize the node activations so that the average activation level becomes equal to the constant π.
4. Check to see if there are any executable nodes and if so, choose and execute the one with the highest activation level.
5. If there is no executable node, reduce the global threshold (θ) and return to step 1.

4 Experimental results

The proposed architecture can be applied to many practical problems such as pattern recognition, mobile robots, and virtual avatars. There are many intelligent models that address the problem of controlling mobile robots, because this is not an easy task. We have adopted the Khepera mobile robot because it has been frequently used [23]. This kind of robot has 8 light sensors and distance sensors with two wheels. Eight infrared proximity sensors were placed around the robot. These sensors embedded an infrared emitter and a receiver, which allowed us to make two measurements. The normal ambient light measurement was made using only the receiver part of the device, without emitting light. A new measurement was made every 20 ms. During these 20 ms, the sensors were read sequentially every 2.5 ms. Generally, the value ranges from 50 to 500 and the smaller the value, the brighter the location. The distance sensors range from 0 to 1024 and the higher the value, the closer the obstacle. In a previous work, Cho developed a robot controller for avoiding obstacles which used CA-based neural networks [24].

4.1 Evolving basic behaviors

In this paper, we list four basic behaviors which are defined as follows:

- Recharging Battery: If a robot arrives at a battery recharge area, the battery is recharged. This behavior enables the robot to operate for as long as possible.
- Following Light: The robot goes to a stronger light. This behavior can be used to make the robot go to the battery recharge area, because the light source is located in that area.
- Avoiding Obstacles: If there are obstacles around the robot, it avoids them without bumping against them.
- Going Straight: If there is nothing around the robot, it goes straight ahead. This behavior allows it to move continuously without stopping.

These basic behaviors evolve incrementally. In our experiments, we attempted to incrementally evolve a mobile robot to avoid bumping against obstacles. Because the robot behavior is composed of going straight and turning left and right, the robot can be incrementally evolved to achieve those behaviors step by step (Fig. 10). Fitness is evaluated based on the velocity of the robot and the number of movements.

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

S: The number of movements until stopping: V_i: The value determined by the velocity in the ith step.

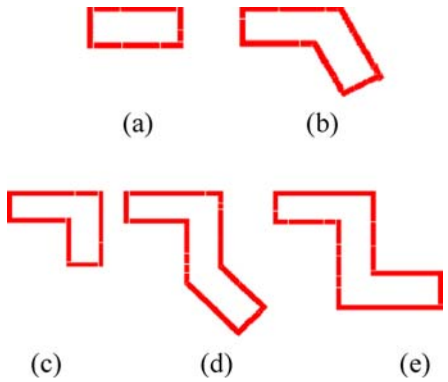


Fig. 10 Experimental environments for incremental evolution

The parameters of the evolution are summarized in Table 2. After sorting the individuals by their fitness values, half of them survive. The length of the chromosome is the multiplication of the number of the cells and 7 bits for each cell. The 1-point crossover is used and this crossover point is randomly selected from the number between 0 and the length of the chromosome. Mutation occurs as follows. For each cell, if the normalized random number is smaller than the predefined mutation rate, 6 bits of segments (all segments except 1 bit for the determination of the neuron cell) are replaced with the binary representation of the number randomly generated between 0 and 63. For the cell, if the normalized random number is smaller than the neuronal density (the proportion of neurons in the cells), the 1 bit for the determination of the neuron cell is set as 1.

Table 2 Parameters of evolution

The size of cellular automata space	$5 \times 5 \times 5$
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

Figure 11 shows the trajectories of successful robots in each environment. The environments move gradually from (a) to (f) and the behavior is incrementally improved step by step.

Figure 12 shows the simulation environment for the proposed architecture. The black fan-shaped area represents the battery recharge area—the robot can recharge batteries only in this area. The light source is also located here and this is what guides the robot to the black area. In the beginning stage, the battery level of the robot is predetermined. When a robot executes one behavior, the battery level is decreased by one. When the battery level reaches zero, the robot stops.

4.2 Structure of behavior network

Our environment can be represented by 5 different states. These states are defined as follows:

- “In the battery recharging area”: Check if the robot is in the battery recharging area.
- “Obstacles are near”: Check if the maximum value of the distance sensors is larger than 700.
- “Near battery recharge area”: Check if the distance from the robot to the light source is less than 800.
- “In shade area”: Check if the minimum value of the light sensors is larger than 400.
- “Nothing around the robot”: Check if the maximum value of the distance sensors is less than 700.

Because the robot’s battery decreases whenever the robot moves, the robot needs to maintain a high battery level to operate for longer time periods. We set two goals: “battery is not zero” and “battery is not below half”:

- “Battery is not zero”:

$$c = \frac{m - n}{m}$$

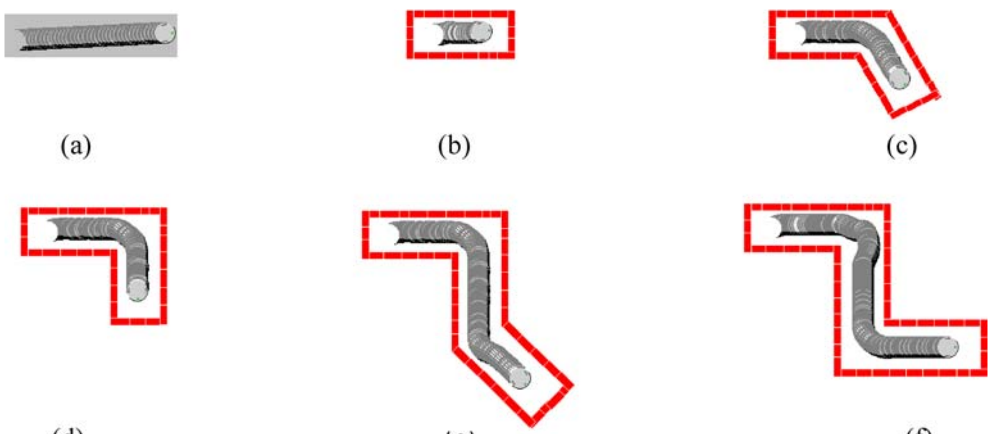


Fig. 11 Trajectories of successful robots in each environment with incremental evolution

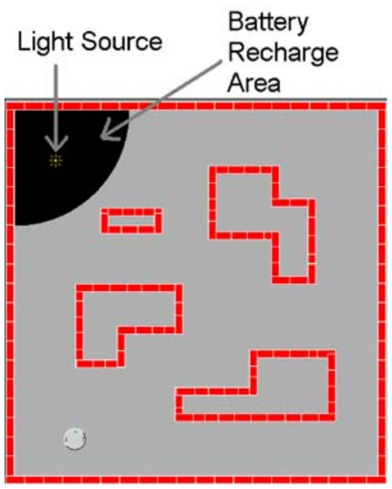


Fig. 12 Simulation environment (simple environment)

where c is the value of "battery is not zero," m is the maximum level of the battery, and n is the current level of the battery.

- "Battery is not below half": Check if the battery level is below the halfway level.

The five states are binary-values and the two goals are continuous values. The behavior network model is composed of four nodes, five states, two goals and their connections. Figure 13 shows the behavior network, where each circle represents the basic behavior and each rectangle represents the sensor or goal. Arcs represent the relationship among the components of the behavior selection mechanism.

Each node has pre-conditions and "add" lists. The node must satisfy all the pre-conditions in order to be executed. Table 3 describes the pre-conditions and "add" lists of the nodes, and Table 4 describes the relationships between basic behaviors. Each node has one or two pre-conditions. The relationships among the nodes are decided as one of the successor links or the predecessor links. We empirically set the values of π, θ, γ, ϕ, and δ in the behavior network as follows: $\pi = 4.5$, $\theta = 3.0$, $\gamma = 0.8$, $\phi = 1.2$, and $\delta = 1.0$.

4.3 Analysis of results

The initial battery level was set at 1500: this means that the robot could only move 1500 times without having to recharge the battery again. In other words, the maximum number of

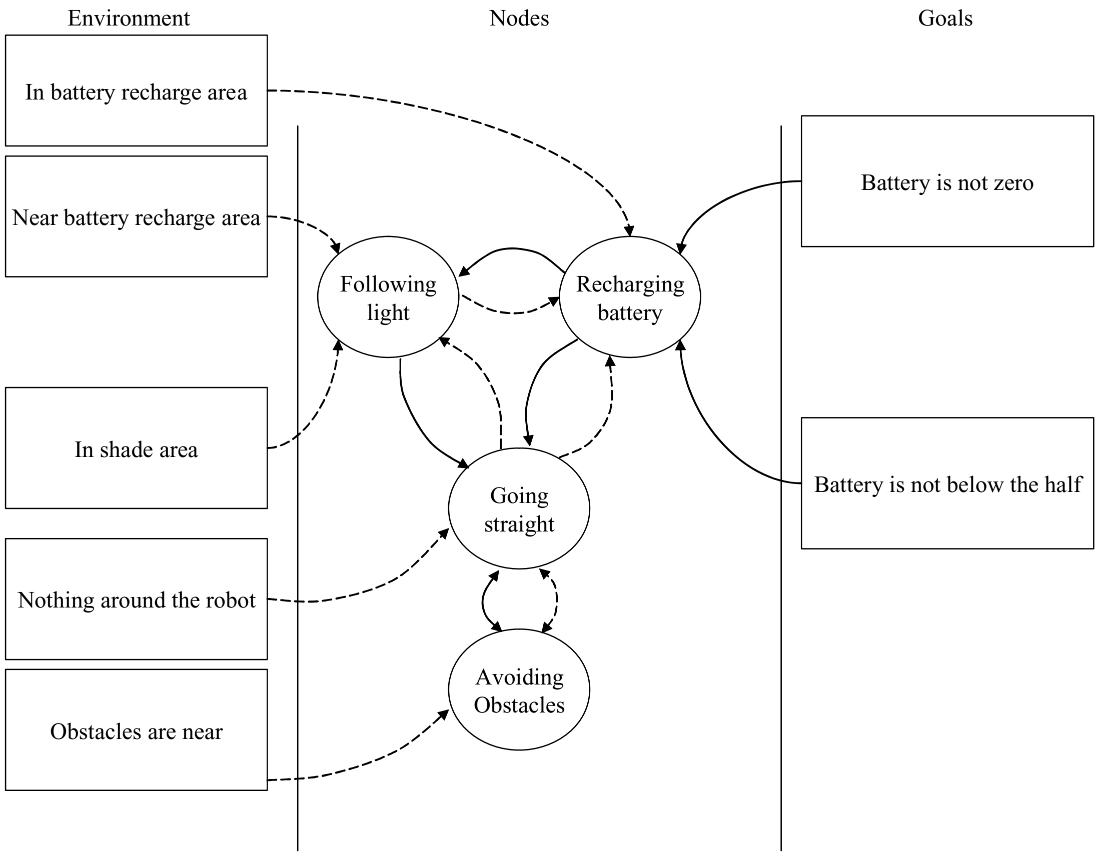


Fig. 13 The structure of the behavior network. Solid lines denote goal-behavior connections or predecessor connections, and dashed lines denote environment-behavior connections or successor connections.

Table 3 Pre-conditions and "add lists of nodes

Preconditions	
Recharging Battery	In battery recharge area
Following Light	In shade area, Near battery recharge area
Avoiding Obstacle	Obstacles are near
Going Straight	Nothing around the robot
Add lists	
Recharging Battery	Battery is not zero, Battery is not below the half
Following Light	In battery recharge area
Avoiding Obstacle	Nothing around the robot
Going Straight	Obstacles are near, In battery recharge area, Near battery recharge area

Table 4 Relationships between the nodes

Predecessor link	Successor link
Recharging Battery → Following Light	Following Light→ Recharging Battery
Recharging Battery → Going Straight	Going Straight → Following Light
Following Light → Going Straight	Going Straight → Recharging Battery
Going Straight → Avoiding Obstacles	Going Straight → Avoiding Obstacles
Avoiding Obstacles → Going Straight	Avoiding Obstacles → Going Straight

execution of basic behaviors is 1500. To operate for a longer time period, the robot had to go to the battery recharging area. In the experimental environment, there were obstacles to hinder the robot's navigation and attempts at battery recharging.

In the experiment, the robot moved a total of 9329 steps. This required many instances of autonomous battery recharging. Figure 14 shows the robot's trajectory in simple and chaotic environments which were composed of irregular particle obstacles. In the simple environment, the robot seemed to navigate without needing to recharge its battery until the level became too low. When the battery level did get too low, the robot's navigation was guided by the light source in the battery recharging area. (This procedure is not defined explicitly in the behavior network but the model can work only with an implicit definition of the goals and relationships among the behaviors. This can make designing high-level behaviors easier.)

In the chaotic environment, the robot behaved in the same manner as in the simple environment. It navigated until the battery level became too low, and then it chose to follow the light source for battery recharging. This behavior generation procedure can be easily illustrated as in Fig. 15. To achieve its goals, the robot combined two or more behaviors and generated new behavioral patterns. For example, to reach the battery recharging area, the robot selected the "avoiding obstacles" and "following light" behaviors in turn. This pattern was not designed but it emerged. By selecting two behaviors in turn, the robot was able to go to the battery recharging area without bumping against obstacles.

The same experiments are repeated by changing the initial starting points. Figure 16 shows four experimental results

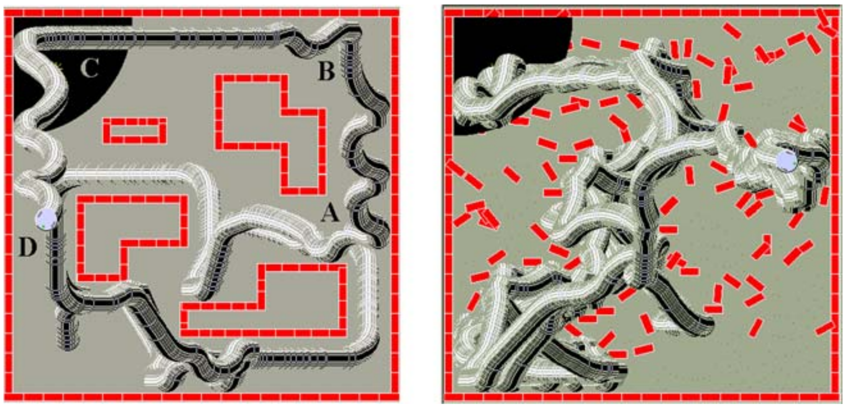


Fig. 14 The robot's trajectory in simple and chaotic environments. By recharging its battery autonomously several times, the robot can navigate the area as long as possible even though obstacles hinder the robot from approaching the battery recharging area

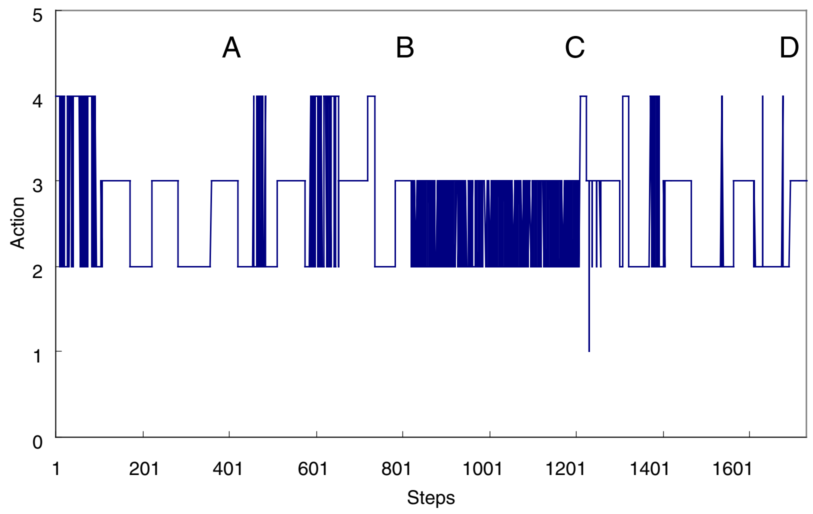


Fig. 15 Action selection of the robot (1 = Recharging Battery, 2 = Following Light, 3 = Avoiding Obstacles, and 4 = Going Straight) (A: Low battery level, B: Turning point to approach the battery recharging area, C: Inside the battery recharging area, and D: Navigation after recharging the battery). Between areas B and C, the robot selects the "avoiding obstacles" and "following light" behaviors in turn to achieve higher-level behaviors such as "going to the battery recharging area without bumping against obstacles." At the C point, the robot selects the battery recharging behavior autonomously

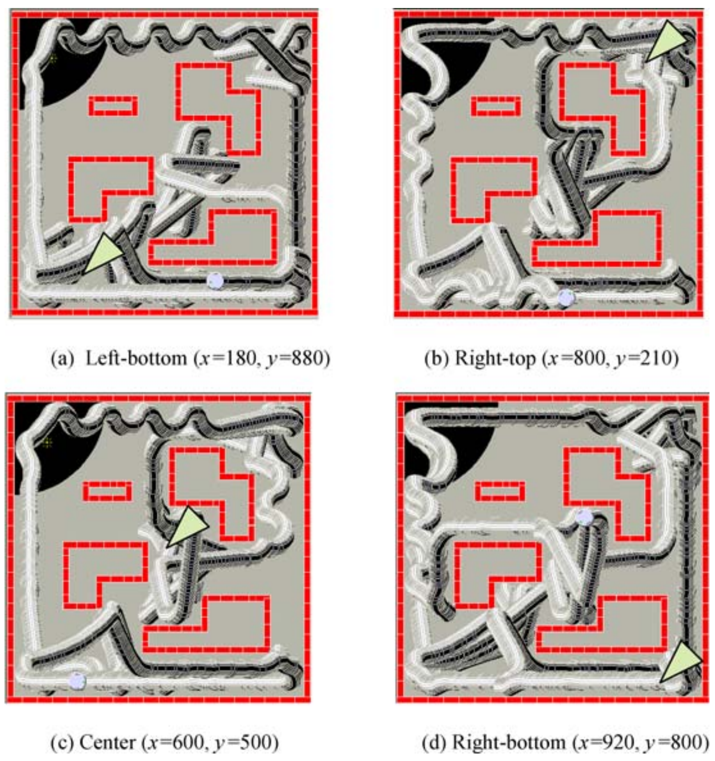
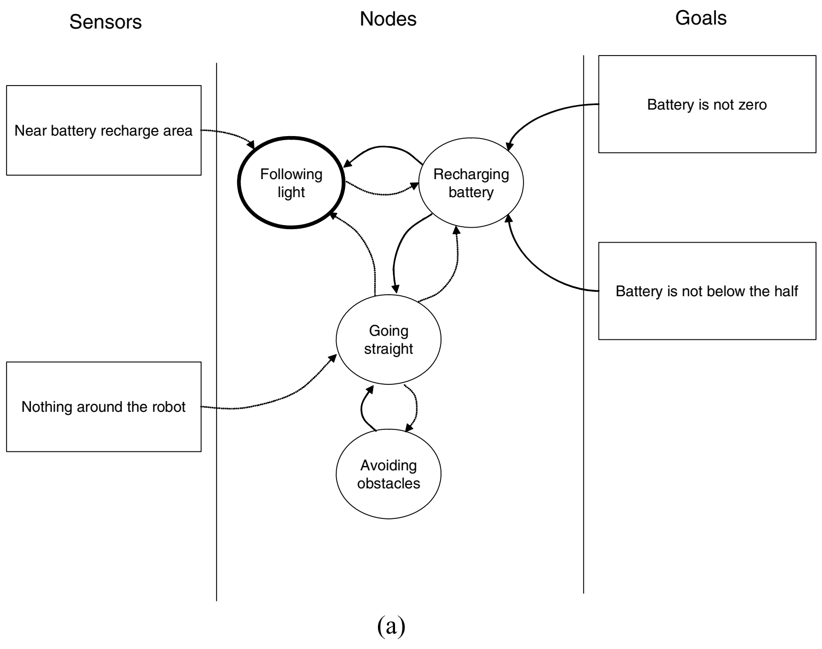

(a) Left-bottom (x=180, y=880)
(b) Right-top (x=800, y=210)
(c) Center (x=600, y=500)
(d) Right-bottom (x=920, y=800)

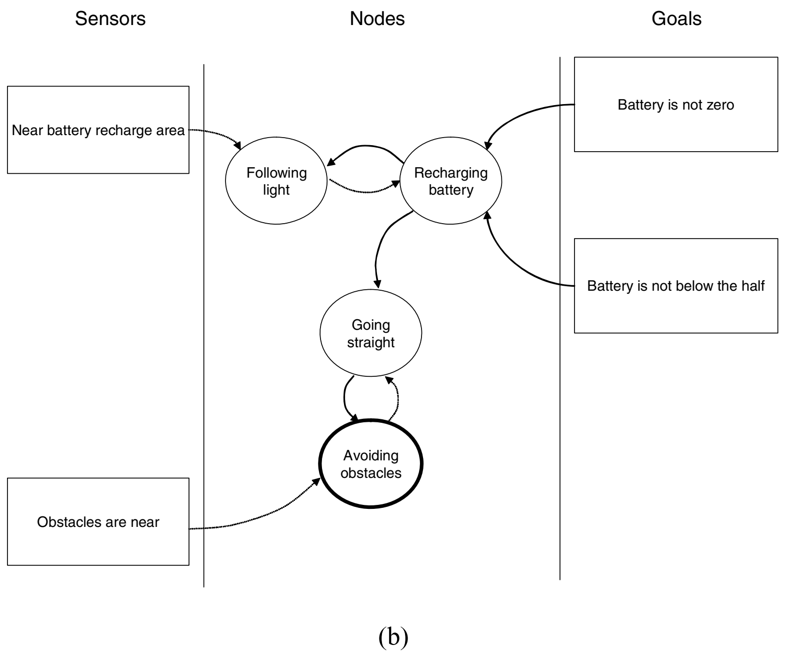
Fig. 16 Trajectories of the robot from different starting points show the robustness of the proposed method. (Triangles indicate the starting points.)

which were obtained by using different starting positions. In the simulation environment, the robot's position is represented as a vector of two real-values which range from 0 to 1000. From any starting points, the behavior patterns are similar.

Figure 17(a) shows that the robot selects the "following light" behavior before approaching the battery recharging area. Figure 17(b) shows that the robot selects the "avoiding obstacles" behavior before reaching the battery recharging area. Our experimental environment is similar to that in Floreano's work [25] except that their enviornment presented no obstacles which blocked the way to the battery recharging area. In the work, only the simple "following light behavior" was evolved. We have introduced some obstacles to make

Fig. 17 Internal status of the behavior network model. (Only activated links and sensors are marked.) (a) When the battery level = 500, the "following light" behavior is selected. (b) When the battery level = 487, the "avoiding obstacles" behavior is selected though the battery level is low because there are obstacles





the problem more diffiult than it was in Floreano's previous work.

To show the usefulness of the behavior selection mechanism, we have attempted to use IF-THEN rules for combining the basic behaviors [26]. The rules designed are as follows:

```
  IF ("Battery Recharging" area)
     Execute "Battery Recharge" behavior
  ELSE
IF (Battery level <0.66×(Maximum level of battery)) AND
         (Minimum value of light sensors ≤ 450)
         IF (Maximum value of distance sensors ≤ 200)
            Execute "Following Light" behavior
          ELSE Execute "Avoiding Obstacles" behavior
     ELSE IF (Maximum value of distance sensors ≤ 250)
       Execute "Going Straight" behavior
          ELSE Execute "Avoiding Obstacles" behavior
```

The rule-based integration shows a weakness when the robot has to adapt to changing environments. Figure 18(a) and (b) show the experimental results in the previous environment and (c) and (d) show the results in the environment that includes some obstacles. This shows that the rule-based integration is not adaptive to situations that change slightly.

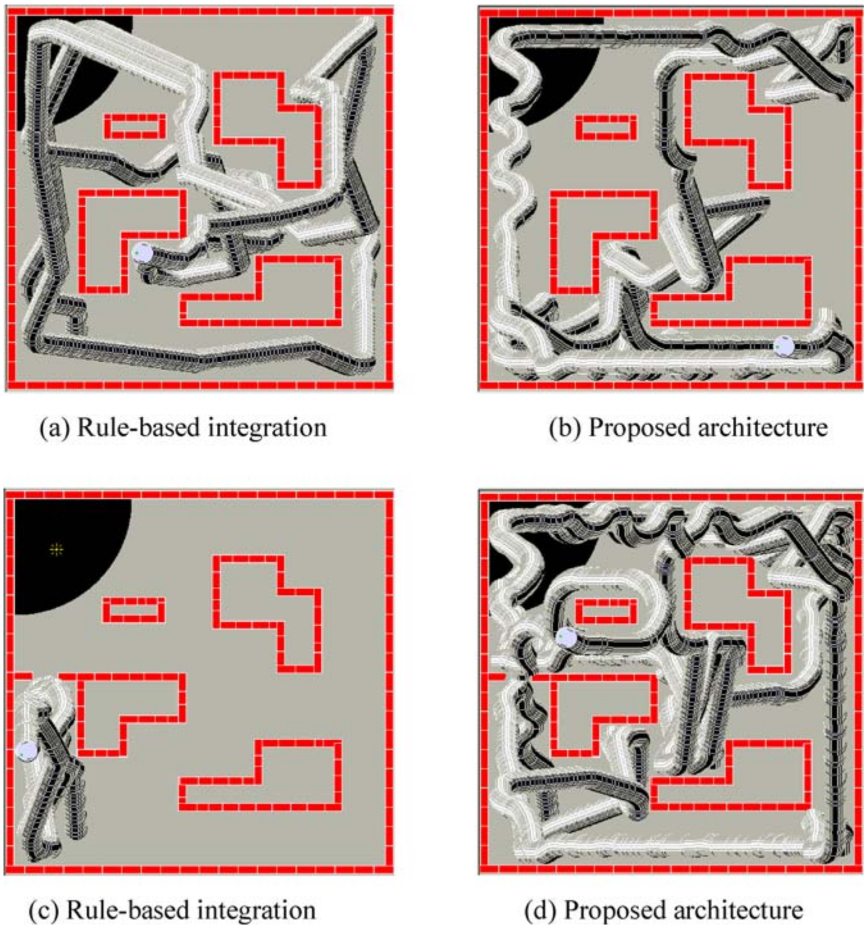


Fig. 18 Trajectory comparison of (a) and (c) rule-based integration and (b) and (d) the proposed architecture in the original environment and the modified one, respectively

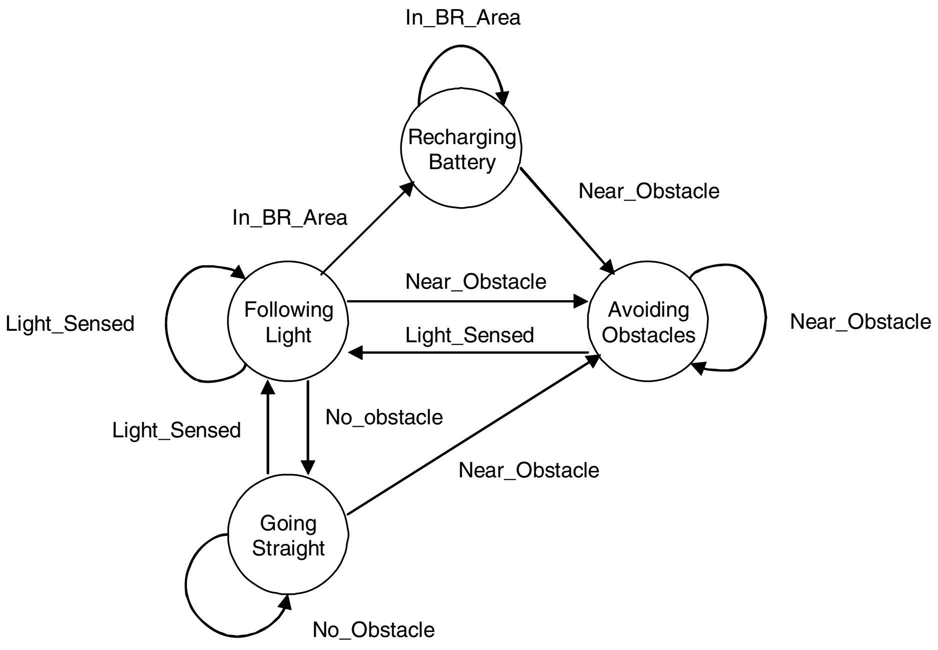


Fig. 19 Finite state machine for action selection

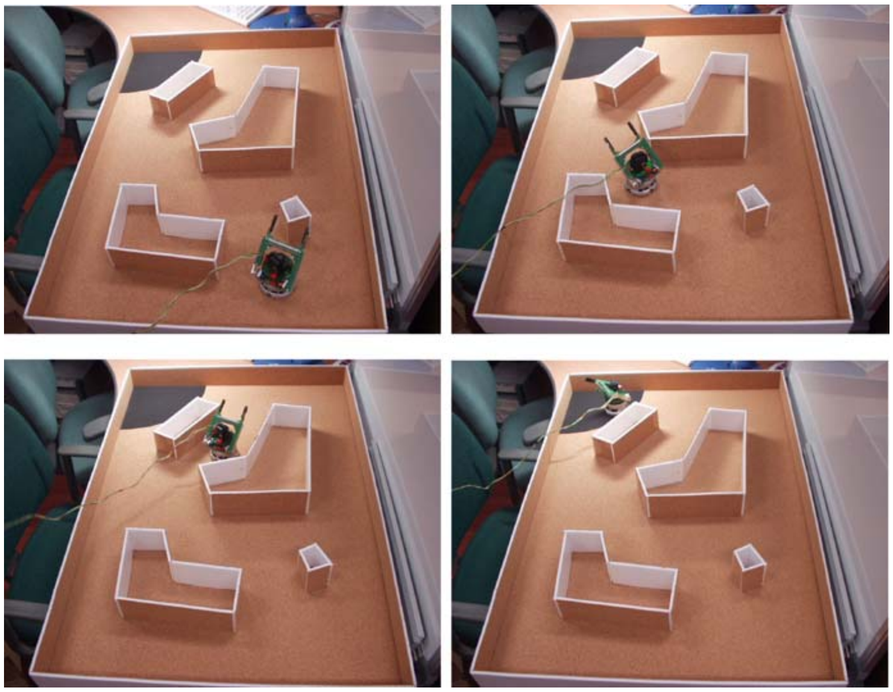


Fig. 20 Experimental results on Khepera II robot

Figure 19 shows an action selection model designed using a finite state machine. According to Pirjanian's survey of the action selection mechanism, FSA (Finite State Automata) is a state-based arbitration method [9]. In this model, it is difficult to coordinate the robot because it selects only the "following light" behavior in the "light-sensed" situation and it selects the "recharge battery" behavior in the battery recharging area until there is an obstacle. Generating high-level behaviors using the interchange of many behaviors is difficult in that model, because it considers only one state transition among candidates and the complex interchange of behaviors is difficult to describe. In simulation, the model fails to achieve the goal.

Figure 20 shows the results of our experiments on the Khepera robot. After navigating the environment with less bumping, the robot went to the battery recharging area automatically. Though the simulation results are slightly different, the goal of the robot was achieved. Our previous work [24] focused on evolving basic behaviors (avoiding obstacles) using neural networks based on CA but it didn't consider the integration of behaviors and incremental evolution. Also, the experimental environment used in [24] was simpler than that used in this paper.

5 Conclusions

Even though we have not given many practical applications in this paper, the construction of neural networks based on cellular automata can be easily implemented into VLSI hardware to evolve quickly. In this paper, we have proposed an architecture composed of cellular automata, a neural network, incremental evolution, and a behavior selection mechanism. We applied this architecture to the problem of mobile robot control. In simulations with different environments, the robot was able to recharge its battery autonomously even though there were obstacles. A comparison with the rule-based integration of other behaviors shows the advantages of the proposed model.

Future works are as follows. It is necessary to apply the proposed architecture to a real robot. The automatic construction of the behavior network is crucial to achieve scalability of the proposed architecture. The number of the behaviors for the experiments must also be increased.

Acknowledgments This research was supported by the Ministry of Information and Communication, Korea under the Information Technology Research Center support program supervised by the Institute of Information Technology Assessment, IITA-2005-(C1090-0501-0019).

References

1. Crutchfield JP, Mitchell M (1995) The evolution of emergent computation. In: Proceedings of the National Academy of Sciences, vol 91, pp 10742–10746
2. Morales FJ, Crutchfield JP, Mitchell M (2001) Evolving two-dimensional cellular automata to perform density classification: a report on work in progress. Parall Comput 27(5):571–585

3. Gers F, de Garis H, Korkin M (1997) CoDi-1Bit: a simplified cellular automata based neural model. In: Proc Conf on Artificial Evolution, Nimes, France, pp 315–334
4. de Garis H, Korkin M (2002) The CAM-Brain machine (CBM): an FPGA-based hardware tool that 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):35–68
5. Nicolescu MN, Mataric MJ (2001) Learning and interacting in human-robot domains. IEEE Trans on Systems, Man and Cybernetics-Part A 31(5):419–430
6. Mataric MJ (2001) Learning in behavior-based multi-robot systems: Policies, models and other agents. Cognitive Systems Research 2(1):81–93
7. Mali AD (2002) On the behavior-based architectures of autonomous agency. IEEE Trans Syst, Man Cybernetics-Part C 32(3):231–242
8. Lyons DM (1993) Representing and analyzing action plans as networks of concurrent processes. IEEE Trans Robot Autom 9(3):241–256
9. Pirjanian P (1999) Behavior coordination mechanisms: State-of-the-art. Tech-report IRIS-99-375 Institute for Robotics and Intelligent Systems, School of Engineering, University of Southern California
10. Shi L, Zhen Y, Zengqi-Sun (2000) A new agent architecture for RoboCup tournament: cognitive architecture. In: Proc of the 3rd World Congress on Intelligent Control and Automation vol 1, pp 199–202
11. Guessoum Z (1997) A hybrid agent model: a reactive and cognitive behavior. In: The International Symposium on Autonomous Decentralized Systems, pp 25–32
12. Lyons DM, Hendriks AJ (1995) Planning as incremental adaptation of a reactive system. Robot Autonom Syst 14(4):255–288
13. Murphy RR, Hughes K, Marzilli A, Noll E (1999) Integration explicit path planning with reactive control of mobile robots using Trulla. Robot Autonom Syst 27(4):225–245
14. Aguirre E, Gonzalez A (2000) Fuzzy behaviors for mobile robot navigation: design, coordination and fusion. Int J Approx Reas 25(3):255–289
15. Nolfi S, Floreano D (2002) Synthesis of autonomous robots through evolution. Trends in Cogn Sci 6(1):31–37
16. Lee W-P (1999) Evolving complex robot behaviors. Inform Sci 121:1–25
17. Iwakoshi Y, Furuhashi T, Uchikawa Y (1998) A fuzzy classifier system for evolutionary learning of robot behaviors. Appl Math Comput 91:73–81
18. Floreano D, Mondada F (1998) Evolving neurocontrollers for autonomous mobile robots. Neur Netw 11(7–8):1461–1478
19. Gutowitz H (1991) Cellular automata: theory and experiment. Physica D 45(1–3)
20. Goldberg DE (1989) Genetic algorithms in search, optimization, and machine learning Addison-Wesley Publishing Company
21. Mathias KE, Whitley LD (1994) Initial performance comparisons for the delta coding algorithm. In: Proc. of IEEE Conf on Evolutionary Computation vol 1, pp 433–438
22. Gomez F, Mikkulainen R (1997) Incremental evolution of complex general behavior. Adapt Behav 5:317–342
23. Team (1999) Khepera Simulator Version 5.02 User Manual
24. Cho S-B, Song G-B (2000) Evolving CAM-Brain to control a mobile robot. Appl Mathe Comput 111:147–162
25. Floreano D, Mondada F (1996) Evolution of homing navigation in a real mobile robot. IEEE Trans Syst, Man Cybern-Part B 26(3):396–407
26. Song G-B, Cho S-B (1999) Rule-based integration of multiple neural networks evolved based on cellular automata. In: Proc of the IEEE Int Conf on Fuzzy Syst, pp 791–796

Kyung-Joong Kim (Student Member, IEEE) received the B.S. and M.S. degree 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.