Available online at www.sciencedirect.com
SCIENCE DIRECT®
Applied Soft Computing 5 (2005) 245–257
Applied Soft Computing
www.elsevier.com/locate/asoc

Evolutionary behavior learning for action-based environment modeling by a mobile robot

S. Yamada*

National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda, Tokyo 101-8430, Japan

Received 5 November 2003; received in revised form 28 June 2004; accepted 20 July 2004

Abstract

This paper describes an evolutionary way to acquire behaviors of a mobile robot for recognizing environments. We have proposed Action-based Environment Modeling (AEM) approach for a simple mobile robot to recognize environments. In AEM, a behavior-based mobile robot acts in each environment and action sequences are obtained. The action sequences are transformed into vectors characterizing the environments, and the robot identifies the environments with similarity between the vectors. The suitable behaviors like wall-following for AEM have been designed by a human. However the design is very difficult for him/her because the search space is huge and intuitive understanding is hard. Thus we apply evolutionary robotics approach to design of such behaviors using genetic algorithm and make simulations in which a robot recognizes the environments with different structures. As results, we find out suitable behaviors are learned even for environments in which human hardly designs them, and the learned behaviors are more efficient than hand-coded ones.

Keywords: Evolutionary robotics; Action-based environment modeling; A mobile robot; Genetic algorithm

1. Introduction

Previous research on an autonomous agent which recognizes environments have been done primarily in robotics. The most studies have tried to build a precise geometric map using a robot with high-sensitive and global sensors like vision [1]. Since their main aim is to navigate a robot with accuracy, a precise map is necessary. However, to a simple and important task like environment recognition, such a strict map may be unnecessary. Actually many natural agents like animals seem to recognize the environments only with low-sensitive and local sensors [2]. In terms of engineering, it is important to build a mobile robot which can recognize environments only with the inexpensive sensors.

Thus we have tried to build a mobile robot which recognizes environments only with low-sensitive and local sensors. Since such a robot does not know its position by dead reckoning in the environment, it cannot build a global map of the environment. Hence we proposed approach that a mobile robot can recognize the environment with *action sequences*.

* Tel.: +81 3 4212 2562; fax: +81 3 4212 2562.
E-mail address: seiji@nii.ac.jp.

1568-4946/$ – see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.asoc.2004.07.004

We call this approach Action-based Environment Modeling (AEM) [3,4]. In AEM, a mobile robot is behavior-based and acts using *given* suitable behaviors like wall-following in environments. Then the action sequences executed in each environment are obtained, and transformed into environment vectors. A robot identifies the by comparing environment vectors.

Through the research on AEM, we have recognized a significant problem: *where the suitable behaviors to AEM come from*? An easy solution is that a human designs them. This approach has been done thus far, and has succeeded in simple domains. However the behavior design becomes quite difficult for a human designer as the variety of environments increases. Because a search space becomes huge and the intuitive understanding on suitable behaviors becomes hard. Hence an automatic design method is necessary [5], and we apply *evolutionary robotics* approach [6–8] in which the behaviors of a robot are designed using evolutionary computation.

In this paper, we apply evolutionary robotics to acquire the suitable behaviors to AEM. We propose the evolutionary design method of such behaviors using Genetic Algorithm (GA) and make experiments for evaluation [9]. For future implementation on a real mobile robot, we use a Khepera simulator in the experiments. From the experimental results, we found out that our evolutionary robotics approach is promising to automatically acquire suitable behaviors for AEM, and the acquired behaviors are more efficient than hand-coded ones.

2. Related work

In the similar approach to AEM, several studies have been done in robotics [10] and artificial life [11]. Nehmzow and Smithers studied on recognizing corners in simple enclosures with a self-organizing network [11]. They used direction-duration pairs, which indicate the length of walls and shapes of past corners, as an input vector to a self-organizing network. After learning, the network becomes able to identify corners. However the recognized structure is very local. Mataric represented an environment with automaton in which nodes correspond to landmarks [10]. Though the representation is more robust than geometric one, a mobile robot must segment raw data into landmarks and identify them. Nakamura et al. utilized a sequence of sonar data in order to reduce the uncertainties in discriminating the local structure [12]. Though the sequence consists of sensor data (not actions), their approach is similar to AEM. Kato et al. used modified random-walking for extracting environment structure in statistical way. In most researches, *wall-following* has been used as suitable behaviors in [4,10,11]. The behaviors were described by human designers, and fixed independently of tasks. Hence they have a significant problem that the design of the behaviors is very difficult.

There are several studies for applying Genetic Programming (GP) [13] to learn behaviors of a mobile robot [14–17]. Unfortunately, in all the studies, very few and simple behaviors like obstacle avoidance were learned. In contrast with them, our aim is to learn the suitable behaviors to AEM, and the behaviors is complicated one consisting of several kinds of primitive behaviors.

Our research is also relevant to automatic generation of pattern recognition procedure because the environment is considered describing a pattern. Evolutionary design of neural networks for pattern recognition was studied [18]. However they focus on a traditional pattern recognition task, and never deal with the cost for recognition, which is significant in this research.

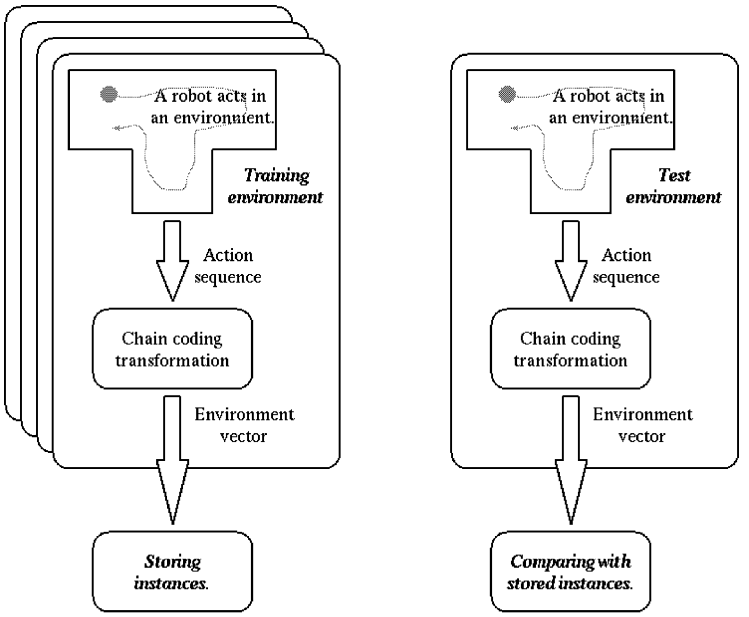
Reinforcement learning is also widely applied to design behavior of an autonomous agent like a robot [19]. In reinforcement learning, behaviors are described a policy consisting of states in an environment and actions which can be executed in each state, and rewards are assigned to some actions in some states. The purpose of reinforcement learning is to search for the optimal policy with which an agent can obtain the maximum expected rewards by executing actions. The all studies of reinforcement learning in robotics also try to the optimal policy with which a robot can trace the shortest path in an environment. Since our objective of behavior learning in this study is to search suitable behaviors to AEM, not the shortest path from a start point to a goal point in an environment, the objective of reinforcement learning is significantly different from ours. Furthermore, it is intuitively difficult to design a reward function of reinforcement learning which is equivalent to a fitness

function for our evolutionary robotics in this task. Because our fitness function evaluates the recognition ability of AEM using behaviors and does not directly evaluate actions in certain states like a reward function of reinforcement learning.

3. Task: action-based environment modeling

In AEM [4], a mobile robot is designed in a behavior-based approach [20]. The *behavior* means mapping from *states* to *actions*. As mentioned later, a state is described with conditions on sensed values, and an action is described with motor commands. A human designer describes states, actions and behaviors so that action sequences can represent environment structure. An AEM procedure consists of two stages: a *training phase* and a *test phase* (Fig. 1). In the training phase, *training environments* having a *class* are given to a robot. The class means a category in which the environment should be included, and plural environments may be included in the same class. The mobile robot acts in the training environments using given behaviors, and obtains sequences of executed actions (called *action sequence*) for each of them. The action sequences (lists of symbols) are transformed into real-valued vectors (called *environment vector*s) using a chain coding-like method. The environment vectors are stored as *instances*, and a training phase finishes.

In the test phase, a robot is placed in a *test environment* which is typically one of training environments. The robot tries to identify the test environment with one of training environments, and we call this task *environment recognition*. Note that though a test environment may be one of training environments, the action sequence of the test environment is significantly different from every sequence of training environments because of noise in sensing and action. Thus generalization is necessary for environment recognition. The identification is done using 1-Nearest Neighbor method [21], i.e. the robot selects the most similar instance to the test environment and considers the class of the instance is that of the test environment. The similarity is evaluated with Euclidean distance between environment vectors.



(a) Traning phase (b) Test phase

Fig. 1. Overview of AEM.

We fully implemented an unsupervised version of AEM in which Self-Organizing Maps (SOM) [22] were applied to classify the training environment without a teacher. Through experiments with a real mobile robot, we found out AEM was promising for environment recognition and adaptive to noisy environments [4].

However, in AEM, there is a significant problem: where the suitable behaviors come from. Since the suitable behaviors depend on environment structure which a robot should recognize, they have been described by human designers thus far. However the task is potentially very difficult for them. Because the search space for a suitable behavior is very huge: the computational complexity is $O(a^s)$ where a and s are the number of actions and states. Thus, we propose the evolutionary robotics approach to automatically acquire the suitable behaviors.

4. States, actions and environment vectors

Using real mobile robots as individuals in GA is currently impractical because it is impossible to operate several tens of real robots for more than 100 generations. Thus we use a simulator for acquiring behaviors, and intend to implement the learned behaviors on a real mobile robot.

Note that in this research, *on-line evolutionary learning* on a single real robot like [16] is hardly applicable. In on-line evolutionary learning, the behaviors of each individual are executed in order for a few or a single time step like time sharing system and the short sequences of executed actions are independently evaluated by a fitness function. However, as mentioned later, for evaluation of AEM, a large number of actions need to be continuously executed by an individual because long sequences of actions should be evaluated for AEM. Thus we cannot apply on-line evolutionary learning to acquire suitable behavior to AEM, and learning by a real mobile robot is very hard.

4.1. A simple mobile robot: Khepera

We use a miniature mobile robot Khepera™ (Fig. 2) as a target real robot. It has Motorola 68331 Micro processor, 256 kbyte RAM. As shown in Fig. 3, it also has two DC motors as actuators and eight Infra-Red proximity sensors which measure both distance to obstacles and light strength. Since the sensor data is imprecise and local, Khepera cannot know its position in a global map. In the later experiments, the simulator build for Khepera will be used.

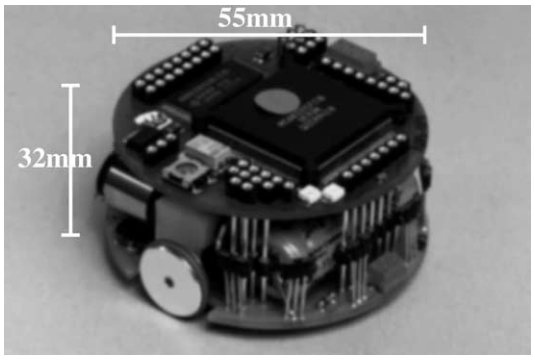


Fig. 2. Khepera.

4.2. States and actions

We describe a state with a range of a sensed value. For reducing the search space of behaviors, we restrict the number of states and actions. A sensor on Khepera returns 10 bit (0–1023) value for distance and light strength, and the value is very noisy and crisp. Thus we transform the distance value into binary values 0 or 1. The value ‘‘0’’ means an obstacle exists within 3 cm from a robot. The value ‘‘1’’ means that no object exists there. Furthermore only three sensors (0, 2 and 5 in Fig. 3) are used for reducing states.

Next, states for light strength are described. The only four sensors (0, 2, 5 and 7 in Fig. 3) are used. We describe a state using the sensor with the strongest

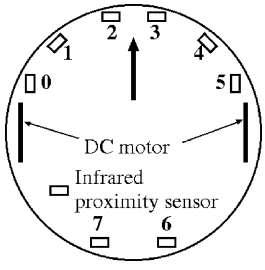


Fig. 3. Sensors on Khepera.

light value and binary values which mean a light is "near" or "far". A state in which all of the sensors have almost same values is also considered. As a result, the number of states for light is nine. The total number of states is 72 (= $2^3 \times 9$)

We describe the following four actions. Through experiments for our research [3,4], we found the actions were sufficient for a mobile robot to do simple behaviors like wall-following.

- *A1*: Go 5 mm straight on.
- *A2*: Turn 30° left.
- *A3*: Turn 30° right.
- *A4*: Turn 180° left.

A mobile robot acts in an environment by executing the actions above, and consequently an action-sequence is obtained. Note that we utilize an action sequence, not a behavior sequence. Because we consider an action describes the environment structure, and several behaviors may contain the same action.

4.3. Environment vectors

The generated action-sequence is transformed into an environment vector. Let an action-sequence and its environment vector be $[a_0, a_1, a_2, \ldots, a_n]$ ($a_i \in \{A1, A2, A3, A4\}$, $a_0 = 0$) and $V = (v_1, v_2, \ldots, v_m)$ ($m \geq n$), respectively. The vector values of V are determined by the following rules. They change the vector value when the direction of movement changes in the similar way to *chain coding* [23]. An example of an environment vector is shown in Fig. 4.

- If $a_i = A1$ then $v_i = v_{i-1}$.
- If $a_i = A2$ then $v_i = v_{i-1} + 1$.
- If $a_i = A3$ then $v_i = v_{i-1} - 1$.
- If $a_i = A4$ then $v_i = -v_{i-1}$.

As mentioned in Section 3, in training phase, training environments are given to a robot for learning. The robots acts in the given environments, and stores the environment vectors transformed from the action sequences. Next, in test phase, test environments are given to the robot. It identifies the test environment with one of training environments by 1-Nearest Neighbor method using Euclidean distance as similarity.

5. Applying GA to acquire behaviors

A behavior is mapping from each state to one of actions. Since we have 72 states and 4 actions, the number of possible behaviors is $4^{72} = 2.2 \times 10^{43}$. We apply GA [24] to search the suitable behaviors to AEM in such a huge search space.

5.1. GA procedure and coding

We use a simple GA procedure and parameters obtained experimentally in the followings:

Step 1: *Initializing population*: An initial population $I_1, \ldots, I_N$ are randomly generated.

Step 2: *Computing fitness*: Compute the fitness $f_1, \ldots, f_N$ for each individual $I_1, \ldots, I_N$.

Step 3: If a terminal condition is satisfied, this procedure finishes. A concrete terminal condition will be defined in Section 6.

Step 4: *Selection*: Using $f_1, \ldots, f_N$, select a child population C from the population.

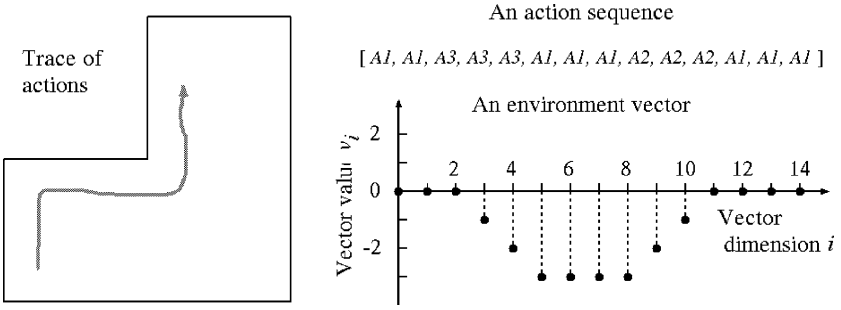


Fig. 4. Transformation into environment vectors.

Step 5: *Crossover*: Select pairs randomly from C on probability P_{cross}. Generate two children by applying a crossover operator to each pair, and exchange the children with the pairs in C.

Step 6: *Mutation*: Mutate the individuals in C based on mutation rate P_{mut}.

Step 7: Go to *Step 2*.

- Population size: 50.
- *Crossover operator*: Uniform crossover.
- *Selection method*: Elite strategy and tournament selection (the tournament size = 2).
- Crossover rate P_{cross}: 0.8.
- Mutation rate P_{mut}: 0.05.

Since we deal with deterministic action selection, not probabilistic, the behavior is mapping from a single state to a single action. Thus we use the coding in which one of actions $\{A1, \ldots, A4\}$ is assigned to each state, and the genotype is shown in Fig. 5.

5.2. Defining a fitness function

Since fitness is a very important for GA, we have to carefully define the fitness function for AEM. We consider three conditions for suitable behaviors to AEM: *termination of actions*, *accuracy of recognition* and *efficiency of recognition*. The fitness functions for each condition are defined and then they are integrated.

5.2.1. Termination of actions

A mobile robot needs to stop its actions by itself. Otherwise it may act forever in an environment and no action sequence is obtained. Thus the termination of actions is the most important condition. We use *homing* for terminating actions, and a robot stops when it returns to the neighborhood of a start point. Homing makes the length of an obtained action sequence depend on the size of an environment. A method to terminate actions in a fixed length of an action sequence does not have such advantage. The termination is evaluated with the following function g.

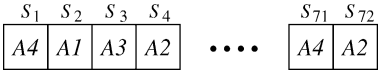


Fig. 5. A coded behavior.

$$g = \frac{(\text{no. of E-trials}) + (\text{no. of H-trials})}{2 \times (\text{total no. of trials})}$$

where E-trials and H-trials means trials in which a robot escaped from the neighborhood of the start point and trials in which it succeeded in homing. Hence this function g evaluates escape as well as homing. Its range is [0,1], and it returns 1 when a robot succeeded in homing in all the environments.

5.2.2. Accuracy of recognition

Another important criterion is accuracy of identifying test environments. The accuracy is evaluated by the following function h. Its range is [0,1], and $h = 0$ when $g \neq 1$.

$$h = \frac{\text{no. of successful test env.}}{\text{total no. of test env.}}$$

5.2.3. Efficiency of recognition

In AEM, a robot needs to *act* by operating physical actuators for recognizing an environment, and the actions significantly cost. Hence the actions should be as small as possible for efficiency. We introduce the following evaluation function k.

$$k = 1 - \frac{\sum_{i=1}^{n} S_i}{nS_{\max}}$$

where S_i is the size of an action sequence obtained in an environment i, $S_{\max}$ is the given limited size of an action sequence, and $k = 0$ when $h \neq 1$. The function have range [0,1] and has more value as more efficient. The obstacle avoidance is implicitly evaluated by the function k because the collision increases the length of an action sequence.

We finally integrate three fitness function into $f = g + h + k$ having range [0,3], and f is used in this research. Since the function h (or k) takes 0 when the value of g (or h) is less than 1, the function f is phased: the termination of actions is satisfied when $1 \leq f$, the recognition is completely correct when $2 \leq f$, and the recognition efficiency is improved when $2 \leq f < 3$.

Fig. 6 shows the whole procedure overview of evolutionary design developed in this research. It integrates a simple GA procedure with a Khepera simulation for evaluating individuals' fitness. The detail procedure has already been shown in Section

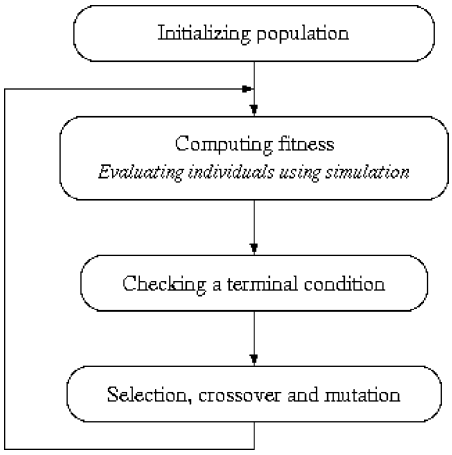


Fig. 6. Whole procedure.

5.1. Note that the fitness is computed by simulation of executing behaviors.

6. Experiments by simulation

We implemented our system for designing behaviors with GA by using a Khepera simulator [25], and made experiments. The parameters used in all experiments were described in the followings: the whole environment is a square (100 cm × 100 cm) the neighborhood of a start point for homing was a circle with 100 mm radius, and the limited size of an action sequence was 2000 actions.

In the simulator, the motor has ±10% random noise in velocity, ±5% one in rotation of robot's body. Furthermore an Infra-Red proximity sensor has ±10% random noise in distance, and ±5% one in light strength. These noises make the simulator close to a real environment.

If a robot cannot return home within the limited size of an action sequence, the trial of the individual results in zero fitness. The terminal condition of GA is that the maximum fitness of the population becomes over 2 or the generation number gets 100. When fitness is over 2, both termination and accuracy are satisfied.

In all experiments, we gave each of training environments to a robot once. The robot acted in the environments, and the environment vectors transformed from the action sequences were stored as instances. Next each of the *training* environments was given to the robot as a *test* environment, and the robot identified each of the test environments with one of the training environments. The start points and directions was fixed in bottom center and left.

Note that though the test environments are same to training environments, the action sequences are different because of the random noise in a simulator. Hence a system has to learn robust behavior against such noise. In all the experiments, we had 10 different trials in initial population, and investigated the averages and standard deviations of the fitness for the generation number in which GA stopped.

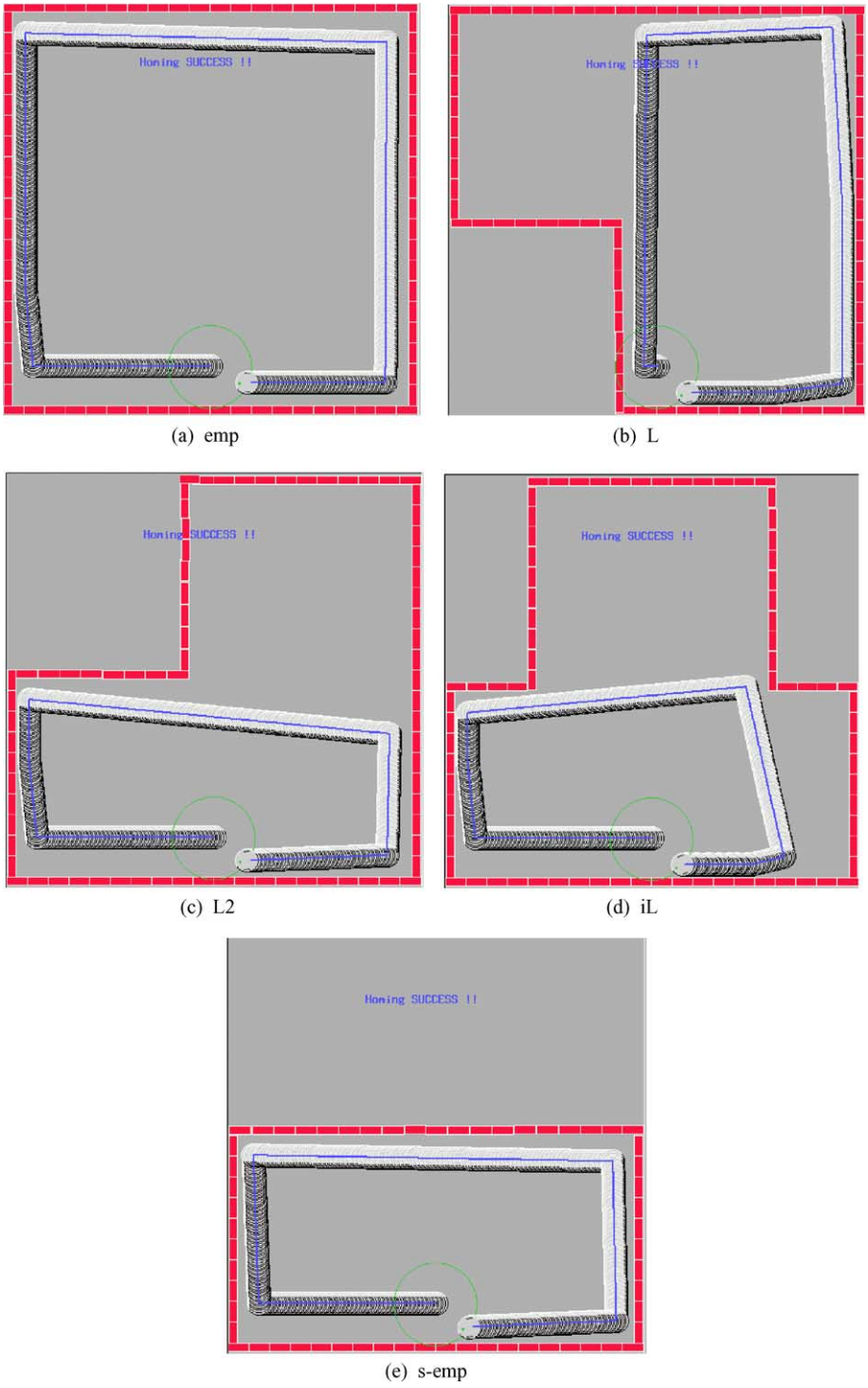
For implementing a system through all the experiments, we employed a PC with PentiumPro (200 MHz), RAM 128 Mbyte and programmed the system using C programming language on Linux and FWM window manager. By using this computational resource, each learning needed 3 min–5 h for convergence.

6.1. Exp-1: different contours in shape

First we made experiments Exp-1 using environments with different contours in shape. The experimental results are shown in Table 1. Four parts ((a–d) in Table 1) of five environments: {emp, L, L2, iL, s-emp} were given to a robot. Additionally 10 and 12 different shape environments were used ((e) and (f) in Table 1. The "GN" is the generation number in which GA stopped, and "MaxF" means the maximum fitness value at GN. The numbers in GN and MaxF stand for averages, and the bracketed numbers are standard deviations. This format is common in all the experimental results. Fig. 7 indicates the trajectories of the best individuals at GN in (d). Fig. 8 shows the maximum values and average values of fitness at every generation for (f) of Exp-1. The average of fitness

Table 1
Experimental results in Exp-1

Env.	Train. env.	GN	MaxF
(a)	{emp, L}	1.0 (0)	2.80 (0.106)
(b)	(a) + L2	2.6 (1.84)	2.43 (0.131)
(c)	(b) + iL	2.8 (1.69)	2.44 (0.025)
(d)	(c) + s-emp	2.8 (1.75)	2.44 (0.093)
(e)	10 env.	2.1 (0.738)	2.51 (0.024)
(f)	12 env.	5.2 (3.08)	2.48 (0.07)



(a) emp (b) L

(c) L2 (d) iL

(e) s-emp

Fig. 7. Trajectories in Exp-1.

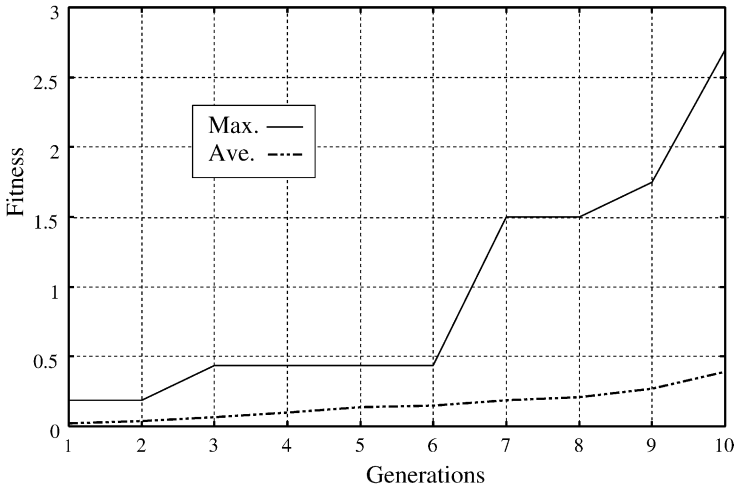


Fig. 8. Fitness for 12 environments.

increased monotonously as well as the maximum of fitness. In such simple environments, the suitable behaviors for AEM were obtained within few generations. The standard deviation was large in GN and small in MaxF, and this tendency was observed through all the experiments. Seeing from Fig. 7, different action sequences were obtained depending on the structure of the environments.

6.2. Exp-2: different lights and shape

Next, by adding different lights to environments in number and position, we made five environments: {emp, 1-la, 2-la, 3-la, 4-la}. Exp-2A is made by using the subsets of the environments. Light was so strong that a robot can detect the light direction in any place. The experimental results are shown in Table 2.

Fig. 9 indicates the trajectories of the best individual at GN in (j). In the figures, a black circle stands for a light.

Table 2
Experimental results in Exp-2A

Env.	Train. env.	GN	MaxF
(g)	emp, 1-la	1.6 (0.966)	2.68 (0.196)
(h)	(g) + 2-la	4.8 (2.78)	2.59 (0.131)
(i)	(h) + 3-la	9.3 (5.19)	2.59 (0.111)
(j)	(i) + 4-la	10.0 (5.94)	2.62 (0.075)

Though the GN increased over ones in Exp-1, the suitable behaviors were obtained. Note that we cannot intuitively understand the behaviors in Fig. 9. This means that it is very hard for human to design such behaviors by hand-coding and this automatic design method is quite effective.

We also set six environments: {emp, 1-la, L, L1-la, iT, iT1-la} by adding lights to three environments with different contours, and made experiments Exp-2B using them. Each environment is included different class and all of them should be distinguished. As a result, GN was 13.2 (7.67) and MaxF was 2.61 (0.091). Fig. 10 shows the trajectories of the best individual at GN. Over all the environments, actions are very different mutually, and the design seems to be difficult.

6.3. Exp-3: a single class including plural training environments

In Exp-1 and Exp-2, each environment is included in different classes. However, in this experiment, plural environments are included in a single class. For recognizing such environments, better generalization is necessary. We assigned three classes to six environments used in Exp-2B: {emp, 1-la}, {L, L1-la}, {iT, iT1-la}, and made experiments Exp-3. As a result, GN was 8.8 (4.24) and the maximum fitness was 2.61 (0.091). Thus our approach is valid for induction from several instances of a class.

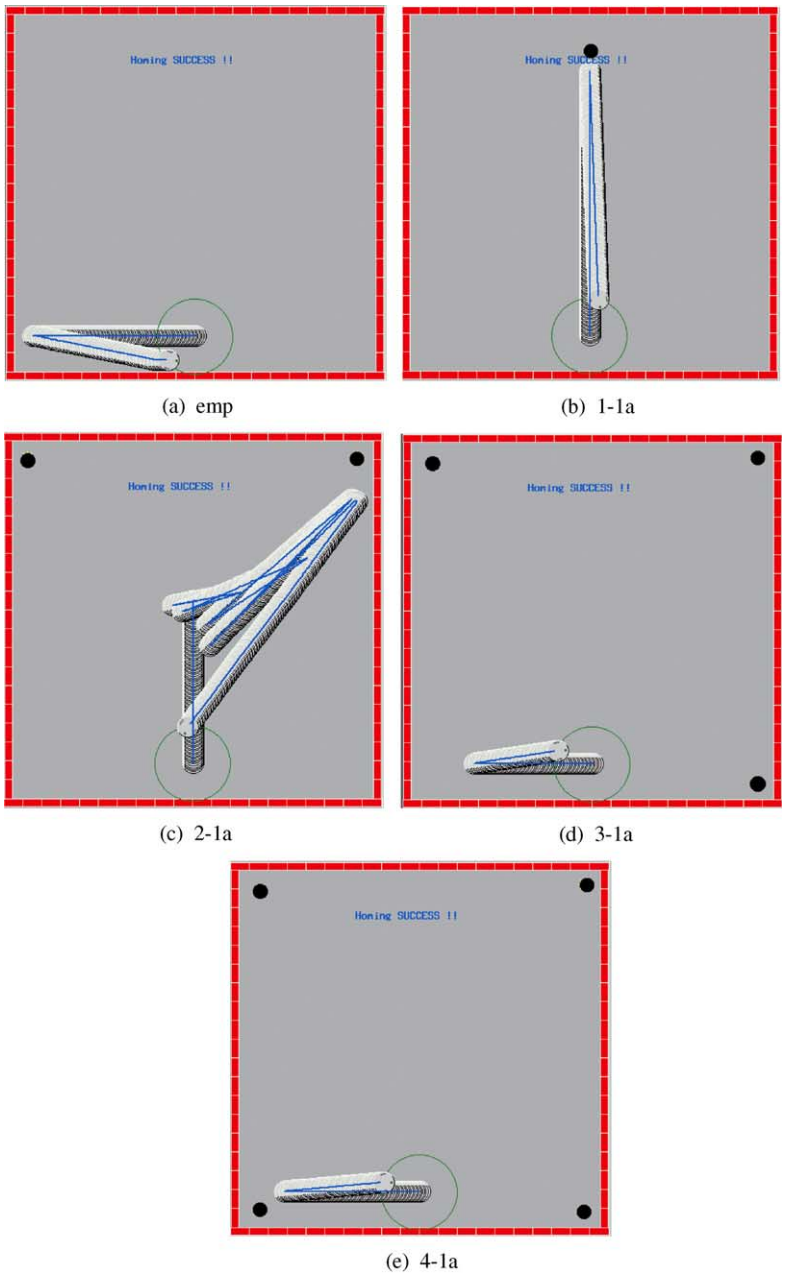


Fig. 9. Trajectories in Exp-2A.

7. Discussion

7.1. Comparison with hand-coded behaviors

It is very difficult that a human designs the behaviors obtained in most of the above experiments. Note that the obtained behaviors are different from the behaviors, like wall-following, random walking, which a human has designed for AEM thus far. Since the search space is huge and we have few heuristics for efficiency, the design task becomes very difficult.

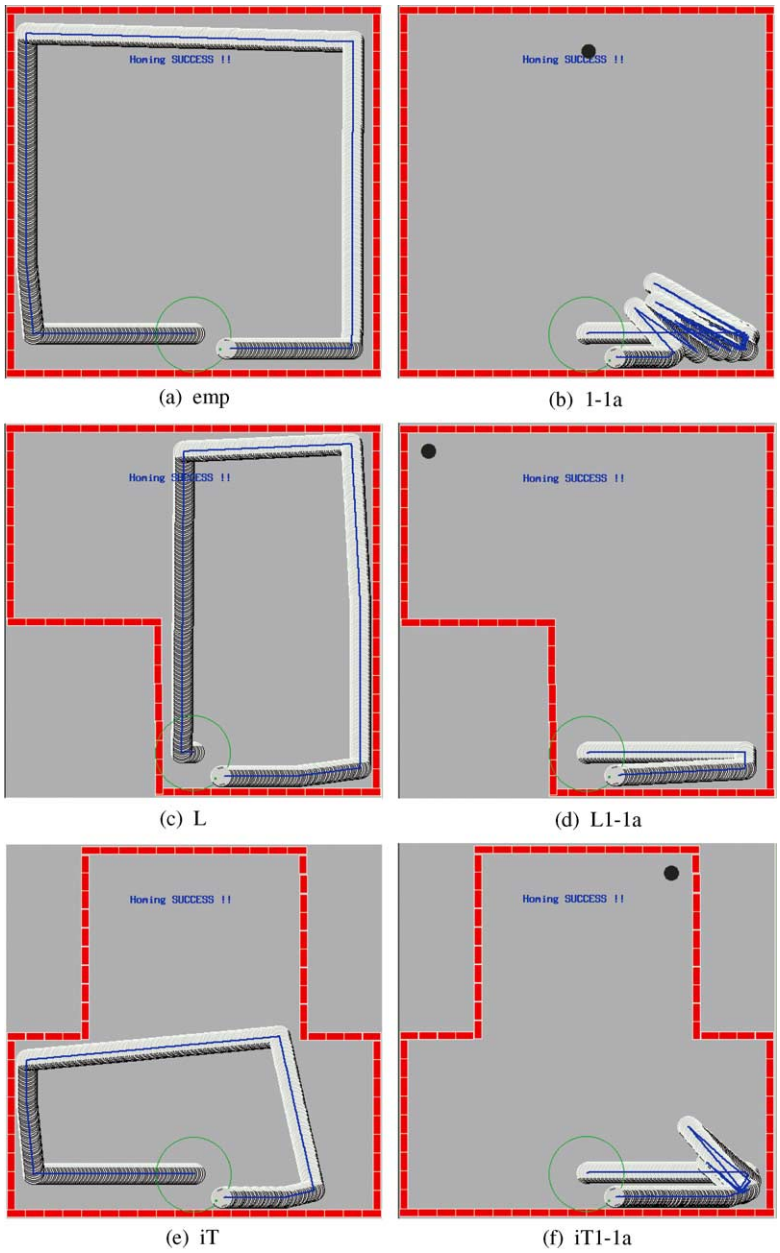


Fig. 10. Trajectories in Exp-2B.

We compare the learned behaviors with *wall-following* which is typical hand-coded behaviors for AEM as mentioned in Section 1. Fig. 11 shows average length of action sequences for test environments in Exp-1 (Table 1). As seeing from the figure, learned behaviors are more efficient than wall-following in all environment sets. Thus our approach designed more efficient behaviors than typical hand-coded behaviors.

Furthermore we again made Exp-1 by introducing 1% random noise to initial positions and rotation of a mobile robot. Though the noise is very small, it made

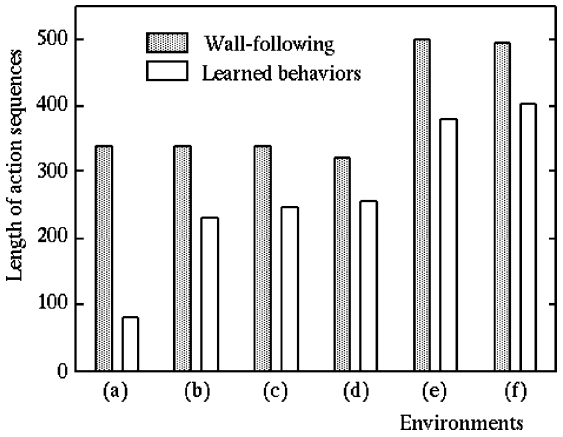


Fig. 11. Comparison with wall-following.

the problem very difficult and the behaviors were acquired in only (a), (b), (c) except (d), (e), (f). This difficulty has been pointed out [14] and is still an open problem in our approach.

7.2. Difficulty of behavior acquisition

As seeing from Env. (a) in Table 1, the suitable behaviors are already included in the initial population. This implies that though the search space is huge, the large number of suitable behaviors exist for Env. (a). To verify this expectation, we investigated the executed behaviors of individuals with maximum fitness in 10 different trials. As results, the executed behaviors are categorized into five classes shown in Table 3. The bracketed number indicate the number of the categorized behaviors in the class, and the actions were described earlier. The intuitive explanations on the states are as follows:

- State S0: No obstacle and light around a robot.
- State S9: Obstacle in the left and no light.
- State S18: Obstacle in the front and no light.
- State S36: Obstacle in the right and no light.

As seeing from Table. 3, this 10 individuals experienced only the subsets of {S0, S9, S18, S36}. Thus the actual combination of states is $4^4 = 256$. Also each of BS-1 and BS-2 has four suitable behaviors,[1] and each of BS-3, BS-4, BS-5 has a single suitable behavior. As results, the total number of suitable behaviors is 11. Since the population size is 50, the probability that any suitable behavior exists in the initial population is $1 - ((256 - 11)/256) = 0.89$. This supports the fact that suitable behaviors exist in the initial population. Furthermore we investigated the average number of suitable behavior classes was 3.4–18.4. Hence we found out a robot actually experienced a small part of 72 states. We are developing heuristics to speedup evolution using the bias of executed behavior [26].

[1] Because BS-1 does not care the action for S9 and the number of actions is four. BS-2 is similarly.

Table 3
Executed behaviors

	Behaviors
BS-1 (5)	S0 → A1
	S18 → A2
	S36 → A2
BS-2 (1)	S0 → A1
	S9 → A3
	S18 → A3
BS-3 (2)	0 → A1
	S9 → A3
	S18 → A2
	S36 → A4
BS-4 (1)	S0 → A1
	S9 → A1
	S18 → A2
	S36 → A4
BS-5 (1)	S0 → A1
	S9 → A4
	S18 → A3
	S36 → A2

8. Conclusion

We proposed evolutionary acquisition of suitable behaviors to Action-based Environment Modeling. GA was applied to search the behaviors, and the simulated mobile robots were used as individuals. States and actions were described for coding chromosomes, and we carefully defined a fitness function. We made various experiments using different environments in shape, lights and both of them, and found out our approach is promising to learn suitable behaviors for AEM. Furthermore some of learned behaviors

were more efficient than hand-coded ones. However there are open problems like the followings.

- *Implementing learned behaviors on a real robot*: Implementation on a real robot is our final target. The gap between simulation and an real environment may make it difficult.
- *Analysis and more complex domain*: We must analyze the experimental results for clarify how to acquire the suitable behaviors. Furthermore we will make experiments in more complex domains, and clarify problems there.
- *Robustness against initial conditions*: When a real robot is used, it is impractical to fix a start point and direction. Thus we must attempt experiments in which the initial situation is noisy. The learning may be difficult because a mobile robot acts sensitively to the initial situation [14].
- *Incremental learning*: For scaling up, we need a method to incrementally learn different kinds of behaviors. It was reported that mutation is biased in a certain environment [27].

References

[1] J.L. Crowly, Navigation of an intelligent mobile robot, IEEE Trans. Robotics Automation 1 (1) (1985) 31–41.

[2] D.S. Olton, Spatial memory, Scientific Am. 236 (6) (1977) 82–99.

[3] S. Yamada, M. Murota, Unsupervised learning to recognize environments from behavior sequences in a mobile robot, in: Proceedings of the 1998 IEEE International Conference on Robotics and Automation, 1998, pp. 1871–1876.

[4] S. Yamada, Recognizing environments from action sequences using self-organizing maps, Appl. Soft Comput. 4 (1) (2004) 35–47.

[5] I. Harvey, P. Husbands, D. Cliff, Issues in evolutionary robotics, in: Proceedings of the Second International Conference on Simulation of Adaptive Behavior, 1992, pp. 364–373.

[6] D. Floreano, F. Mondada, Evolution of homing navigation in a real mobile robot, IEEE Trans. Syst., Man, Cybernetics—Part B: Cybernetics 26 (3) (1996) 396–407.

[7] J. Meyer, P. Husbands, I. Harvey, Evolutionary robotics: a survey of applications and problems, in: Proceedings of the 1st European Workshop Evolutionary Robotics, 1998, pp. 1–21.

[8] H.H. Lund, Adaptive robotics in entertainment, Appl. Soft Comput. 1 (1) (2001) 3–20.

[9] S. Yamada, Evolutionary design of behaviors for action-based environment modeling by a mobile robot, in: Proceedings of Genetic and Evolutionary Computation Conference, 2000, pp. 957–964.

[10] M.J. Mataric, Integration of representation into goal-driven behavior-based robot, IEEE Trans. Robotics Automation 8 (3) (1992) 14–23.

[11] U. Nehmzow, T. Smithers, Map-building using self-organizing networks in really useful robots, in: Proceedings of the First International Conference on Simulation of Adaptive Behavior, 1991, pp. 152–159.

[12] T. Nakamura, S. Takamura, M. Asada, Behavior-based map representation for a sonar-based mobile robot by statistical methods, in: Proceedings of the 1996 IEEE/RSJ International Conference on Intelligent Robots and Systems, 1996, pp. 276–283.

[13] J.R. Koza, Genetic Programming, MIT Press, 1992.

[14] C.W. Reynolds, Evolution of obstacle avoidance behavior: using noise to promote robust solutions, in: J.K.E. Kinnear (Ed.), Advances in Genetic Programming, vol. 1, MIT Press, 1994, pp. 221–241 (Chapter 10).

[15] J.R. Koza, Evolution of subsumption using genetic programming, in: Proceedings of the First European Conference on Artificial Life, 1991, pp. 110–119.

[16] P. Nordin, W. Banzhaf, A genetic programming system learning obstacle avoiding behavior and controlling a miniature robot in real time, Technical report, Department of Computer Science, University of Dortmund, 1995.

[17] T. Ito, H. Iba, M. Kimura, Robustness of robot programs generated by genetic programming, in: Genetic Programming 1996, Proceedings of the First Annual Conference, 1996, pp. 321–326.

[18] S.B. Cho, K. Shimohara, Emergence of structure and function in evolutionary modular neural networks, in: Proceedings of the Fourth European Conference on Artificial Life, 1997, pp. 197–204.

[19] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, MIT Press, 1998.

[20] R.A. Brooks, A robust layered control system for a mobile robot, IEEE Trans. Robotics Automation 2 (1) (1986) 14–23.

[21] B.V. Dasarathy, Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques, IEEE Computer Society Press, 1991.

[22] T. Kohonen, The self-organizing map, in: Proceedings of the IEEE, 1990, pp. 1464–1480.

[23] E.M. Arkin, L.P. Chew, D.P. Huttenlocher, K. Kedem, J.S.B. Mitchell, An efficiently computable metric for comparing polygonal shapes, IEEE Trans. Pattern Analysis Machine Intell. 13 (3) (1991) 209–216.

[24] J. Holland, Adaptation in Natural and Artificial Systems, University of Michigan Press, 1975.

[25] O. Michel, Khepera Simulator v. 2 User Manual, University of Nice-Sophia, Antipolis, 1996.

[26] D. Katagami, S. Yamada, Speedup of evolutionary behavior learning with crossover depending on the usage frequency of a node, in: The 1999 IEEE Systems Man and Cybernetics Conference, 1999, pp. V601–V606.

[27] J. Cairns, J. Overbaugh, S. Miler, The origin of mutants, Nature 335 (8) (1988) 142–145.