Expert Systems with Applications 39 (2012) 12349–12357

Contents lists available at SciVerse ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

Adaptability analysis of genetic network programming with reinforcement learning in dynamically changing environments

Shingo Mabu, Andre Tjahjadi, Kotaro Hirasawa *

Graduate School of Information, Production and Systems, Waseda University, Hibikino 2-7, Wakamatsu-ku, Kitakyushu, Fukuoka 808-0135, Japan

ARTICLE INFO

Keywords:
Evolutionary computation
Genetic network programming
Reinforcement learning
Adaptability
Khepera robot

ABSTRACT

Genetic network programming (GNP) has been proposed as one of the evolutionary algorithms and extended with reinforcement learning (GNP-RL). The combination of evolution and learning can efficiently evolve programs and the fitness improvement has been confirmed in the simulations of tileworld problems, elevator group supervisory control systems, stock trading models and wall following behavior of Khepera robot. However, its adaptability in testing environments, where the situations dynamically change, has not been analyzed in detail yet. In this paper, the adaptation mechanism in the testing environment is introduced and it is confirmed that GNP-RL can adapt to the environmental changes using a robot simulator WEBOTS, especially when unexperienced sensor troubles suddenly occur. The simulation results show that GNP-RL works well in the testing even if wrong sensor information is given because GNP-RL has a function to automatically change programs using alternative actions. In addition, the analysis on the effects of the parameters of GNP-RL is carried out in both training and testing simulations.

1. Introduction

Evolutionary computation such as Genetic Programming (Koza, 1992, 1994) has been successfully applied to automatic program generation for efficient agent behaviors, e.g., robot programming (Kamio & Iba, 2005). Evolutionary computation evolves many individuals usually in simulated environments because it is difficult for all the individuals to make many trials in real environments. After the evolution, the evolved programs are actually used in real environments. However, in real environments, many unexperienced things would occur, e.g., noise and even the hardware troubles such as imprecise sensor information. In order to cope with such cases, the robustness or adaptability of the program is necessary. Since the adaptability provides flexibility to adjust agent actions to unexperienced environments, it enhances the generalization ability of the agent.

Genetic network programming (GNP) is one of the evolutionary algorithms and has been applied to the problems in dynamic environments such as stock trading models (Mabu, Chen, Hirasawa, & Hu, 2007), elevator group supervisory control systems (Hirasawa, Eguchi, Zhou, Yu, & Markon, 2008) and tileworlds (Mabu, Hirasawa, & Hu, 2007). In addition, GNP with reinforcement learning (GNP-RL) (Mabu et al., 2007) has been also proposed as an extended algorithm of GNP, which can efficiently generate programs by combining evolution and learning. Evolutionary computation generally has an advantage in diversified search ability, while reinforcement learning has an advantage in intensified search ability and online learning.

This paper describes an adaptability mechanism of GNP-RL (Mabu, Tjahjadi, Sendari, & Hirasawa, 2010) and analyzes its ability in dynamically changing environments. Although we have already proposed GNP-RL for controlling mobile robot (Mabu, Hatakeyama, Thu, Hirasawa, & Hu, 2006), its adaptability to environmental changes in testing phases has not been confirmed yet. When evolutionary computation adjusts to the environmental changes, it generally takes a long time to change its program structures because they must evolve again. On the other hand, the proposed adaptability algorithm can adapt to the changes quickly. When the proposed algorithm creates programs in the training, alternative actions for troubles are also evolved. Then, if inexperienced things such as sensor troubles occur, the alternative actions are automatically selected by the learning algorithm considering the changes of Q values being updated in the testing environment. During this recovery phase, the use of reinforcement learning and ε-greedy policy have a significant contribution because they cooperatively search for better action sequences online. As a result, recovery can be achieved easily by choosing the alternative actions without having to go through another round of evolution which takes relatively long time.

General reinforcement learning algorithms (Sutton & Barto, 1998) can also do online learning when environments changes, however, GNP-RL can prepare a limited number of alternative

* Corresponding author. Tel./fax: +81 93 692 5261.
E-mail address: hirasawa@waseda.jp (K. Hirasawa).

0957-4174/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.eswa.2012.04.038

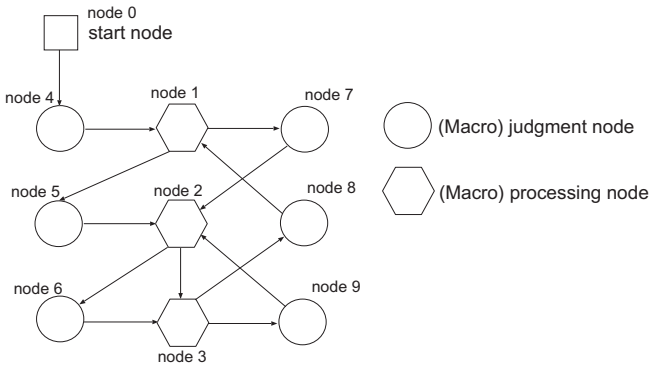

Fig. 1. Basic structure of GNP with reinforcement learning.

actions for troubles, therefore, all the evolution and learning processes do not have to be executed again, which contributes to recovering from troubles quickly.

In the simulations, a robot simulator WEBOTS (Cyberbotics, xxxx) is used for the performance evaluation, where GNP-RL evolves programs of the wall-following behavior of Khepera robot. Then, the adaptability of GNP-RL is verified when the sensor troubles (wrong sensor values are given) suddenly occur.

This paper is organized as follows. Section 2 describes the detailed algorithm of GNP-RL and recovery process from troubles. Section 3 shows a simulation environment and the results. Section 4 is devoted to conclusions.

2. Genetic network programming with reinforcement learning

2.1. Phenotype expression using graph structure

The phenotype expression of GNP-RL is a graph structure which consists of three kinds of nodes; start node, judgment node and processing node. Fig. 1 shows a basic structure of GNP where there are a number of judgment nodes and processing nodes which are connected by directed links to each other. An individual is represented in the form of the inter-connected nodes and the directed links indicate the possible transitions from one node to another. The start node has one outbound connection without any inbound connection, which determines the first node to be executed. The judgment nodes have if-then type branch decision functions, so one of the outbound connections is selected according to the judgment result. Processing nodes determine the actions to be taken by an agent. While judgment nodes have several outbound connections, processing nodes have only one outbound connection. The graph structure of GNP has some inherent characteristics such as compact structure and applicability to partially observable Markov decision processes (Mabu et al., 2007).

GNP has time delays on judgments and processing. In this paper, the time delay of each judgment node is set at one time unit and that of each processing node is set at five time units. In addition, one step of an agent behavior is defined in such a way that one step ends when the used time units exceed nine. For example, when executing seven judgments and one processing, the total used time units become 12, thus one step ends.

2.2. Gene expression of the graph structure

The graph structure of GNP-RL is represented by the combination of the gene structures. A gene structure of node $i(0 \leqslant i < n$ [1]) is shown in Fig. 2 and the correspondence between the judgment/processing nodes and genes is shown in Fig. 3.

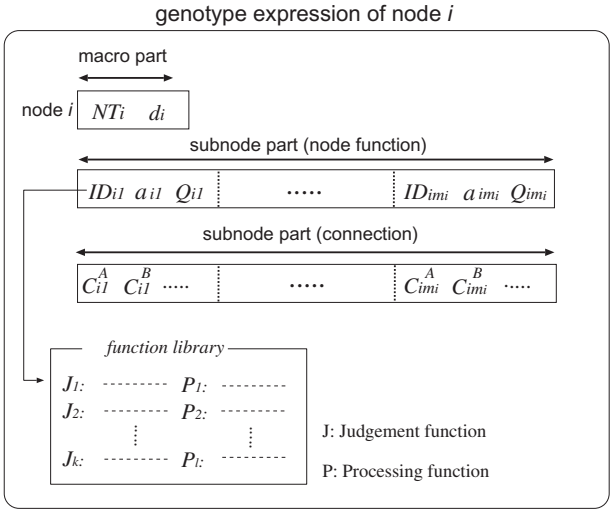

Fig. 2. Structures of nodes.

In GNP-RL, each node (called macro node) contains several subnodes as shown in Fig. 3; therefore the gene structure is divided into macro node part and subnode part. The macro part consists of NT_i and d_i. NT_i represents a node type. $NT_i = 0$ means start node, $NT_i = 1$ means judgment node and $NT_i = 2$ means processing node. d_i is a time delay spent on executing node i.

The subnode part consists of ID_{ip}, a_{ip}, Q_{ip} and C_{ip} ($1 \leqslant p \leqslant m_i$, m_i is the number of subnodes in macro node i). ID_{ip} shows a code number of judgment/processing which is represented by a unique number shown in the function library. This paper uses ID_{ip} to indicate which sensors should be judged by judgment nodes or which wheels (right/left) should be controlled by processing nodes. a_{ip} is used as a threshold in a judgment node or as wheel speed in a processing node, which is explained in Section 2.3 in detail. Q_{ip} means Q value (Sutton & Barto, 1998) which is assigned to each state and action pair (state = node i and action = subnode p). In GNP-RL, state means the current node and action means the selection of the subnodes. $C^A_{ip}, C^B_{ip}, \ldots$ show the next node numbers connected from subnode p in node i.

2.3. How to carry out the node transition

The node transition starts from a start node and the next node is determined by the connection from the start node. When the current node i is a judgment node, first, one Q value is selected from $Q_{i1}, \ldots, Q_{im_i}$ based on ε-greedy policy. That is, the maximum Q value among $Q_{i1}, \ldots, Q_{im_i}$ is selected with the probability of $1 - \varepsilon$, or a random one is selected with the probability of ε. Then, the corresponding ID_{ip} and a_{ip} are selected (Suppose $ID_{ip} = ID_{i1}$ and $a_{ip} = a_{i1}$). ID_{i1} shows a sensor number of Khepera robot whose sensor value is judged. a_{i1} divides sensor input space into two spaces, i.e., when the sensor input is a_{i1} or more, the judgment result is A and the next node number is C^A_{i1}, and when the input is less than a_{ip}, the judgment result is B and the next node number is C^B_{i1}.

When the current node is a processing node, ID_{ip} and a_{ip} are selected in the same way as a judgment node (Suppose $ID_{ip} = ID_{i1}$ and $a_{ip} = a_{i1}$). In this paper, $ID_{i1} = 0$ means the processing node which determines the speed of the right wheel of Khepera robot and $ID_{i1} = 1$ means that of the left wheel. a_{i1} shows the speed of the wheel. The next node number is C^A_{i1}.

In order to appropriately judge an environment and decide its corresponding processing, it is important for GNP-RL to select

[1] Each node has a unique number from 0 to $n - 1$, respectively, when the number of nodes is n.

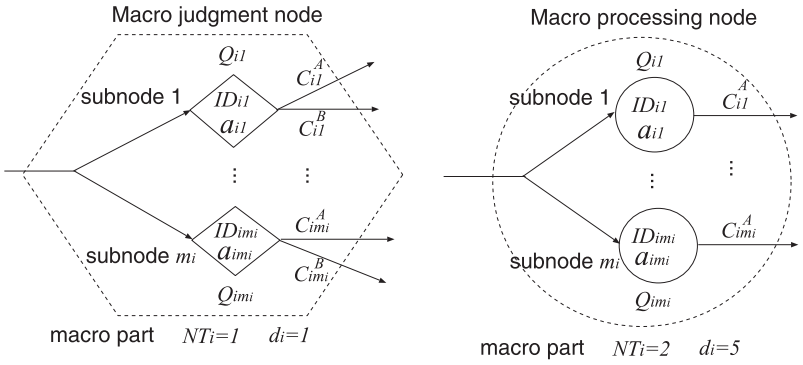

Fig. 3. Macro nodes and subnodes.

appropriate ID_{ip} and a_{ip} according to Q_{ip} in the node transition, which is realized by the combination of evolution and Sarsa described in Sections 2.4.1 and 2.4.2.

2.4. Learning and evolution in the training period

The learning and evolution starts from an initialization of a population. The initialization is carried out by randomly determining the connections, assigning node functions, giving randomly selected integer values to parameter a_{ip} and initializing Q_{ip} at zero.

Fig. 4 shows the flowchart of the training and testing periods. In the training, after the evaluation of all the individuals, the connection between nodes, node functions and parameters are changed by evolution including crossover and mutation.

2.4.1. Q value update by Sarsa during task execution

The node transition is carried out based on Q values and ε-greedy policy and the Q values are updated by Sarsa (Sutton & Barto, 1998) as follows.

(1) Suppose the current node is node i at time t. Then, GNP-RL refers to all Q_{ip} and select one of them based on ε-greedy policy. Suppose that GNP-RL selects Q_{ip_1} $(1 \leqslant p_1 \leqslant m_i)$ and the corresponding node function ID_{ip_1} and parameter a_{ip_1}.
(2) GNP-RL executes the function ID_{ip_1} using a_{ip_1} and gets reward r_t. If the current node is a judgment node, the next node number is determined by the judgment result. For example, it becomes $C^B_{ip_1}$ if the judgment result is B. In the case of a processing node, it is always $C^A_{ip_1}$.
(3) The current node is transferred to the next node j at time $t+1$ and one Q_{jp} is selected by the same way as step 1. Suppose Q_{jp_2} is selected.
(4) Then, the following update is executed.

$$Q_{ip_1} \leftarrow Q_{ip_1} + \alpha(r_t + \gamma Q_{jp_2} - Q_{ip_1})$$
$$\alpha : \text{learning rate} (0 < \alpha \leqslant 1)$$
$$\gamma : \text{discount rate} (0 \leqslant \gamma \leqslant 1)$$

(5) $i \leftarrow j$, $p_1 \leftarrow p_2$ and $t \leftarrow t+1$, then return to step 2.

2.4.2. Genetic operation after task execution

Crossover is executed between two parents and two offspring are generated.

Crossover

(1) Select two parents using tournament selection.
(2) Each node i in the parents is selected as a crossover node with the probability of P_c.
(3) Two parents exchange the genes of the corresponding crossover nodes (i.e., with the same node number).
(4) Generated new individuals become the new ones in the next generation.

Mutation is executed in one individual and a new individual is generated.

Mutation

(1) Select one individual using tournament selection
(2) Mutation operator
  (a) connection: Each node branch is re-connected to another node with the probability of P_m.
  (b) function: Each node function (sensor number in the case of judgment; right/left wheel in the case of processing) is changed to another one with the probability of P_m.
  (c) parameter: Each parameter a_{ip} is changed to another value with the probability of P_m. ($a_{ip} \in [0,1023]$ (judgment node), $a_{ip} \in [-10,10]$ (processing node))
(3) Generated new individual becomes the new one in the next generation.

2.5. Adaptation mechanism of GNP-RL in the testing period

In the testing period, the best individual obtained in the last generation in the training period is used to confirm the adaptation ability, which is the most important point of this paper. The situations are supposed, where some of the sensors are broken in the testing period as environmental changes or troubles. The input values from malfunctioning sensors are incorrect (zero is always given), which means that a robot cannot perceive the approximated distance from obstacles and cannot recognize the new situations. Moreover, the robot does not know whether the sensors are broken or not and even which sensors are broken. As a result, the performance of the robot degrades. In this case, the robot needs to catch the abnormal situations, recover from them and continue working by making use of alternative sensors. In order to realize the adaptation to the troubled situations, Sarsa learning is also carried out in the testing period. The basic recovery process is summarized as follows, which corresponds to Fig. 5.

(1) GNP-RL decides an action according to the node transition.
(2) If the sensor value is incorrect, the judgment result is meaningless.
(3) The actions based on the incorrect information will cause bad situations and less rewards are obtained.
(4) Q value of the subnode which judges the incorrect sensor value will be decreased by Sarsa.
(5) Another subnode is selected to recover from the trouble.

By preparing alternative subnodes in advance, GNP-RL quickly changes subnodes/functions for judging sensors and determining

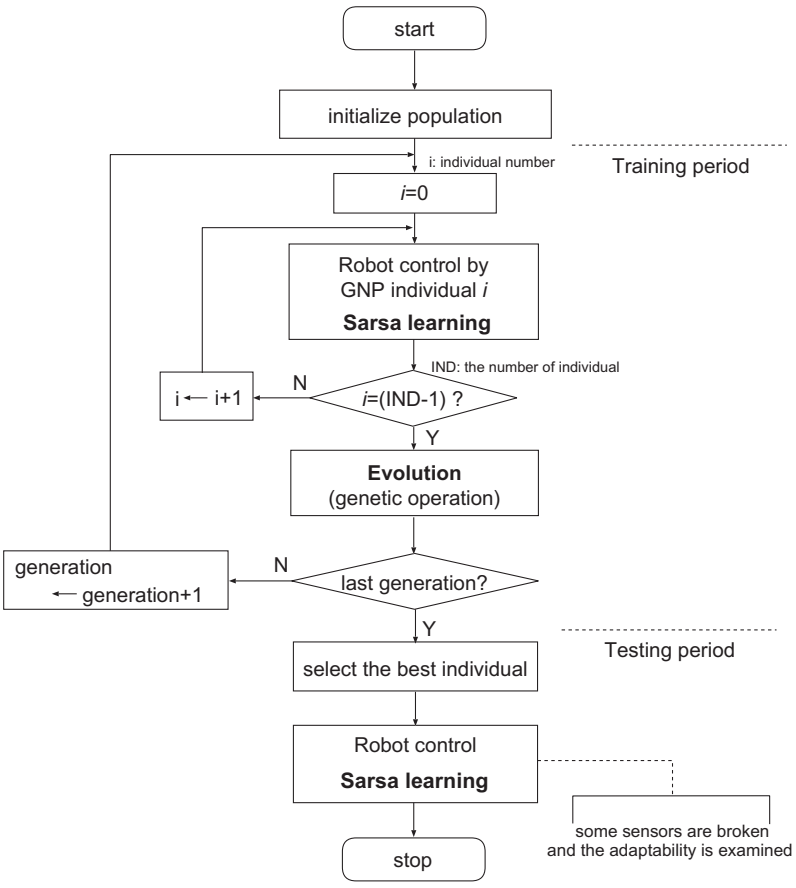

Fig. 4. Flowchart of GNP-RL in the training and testing periods.

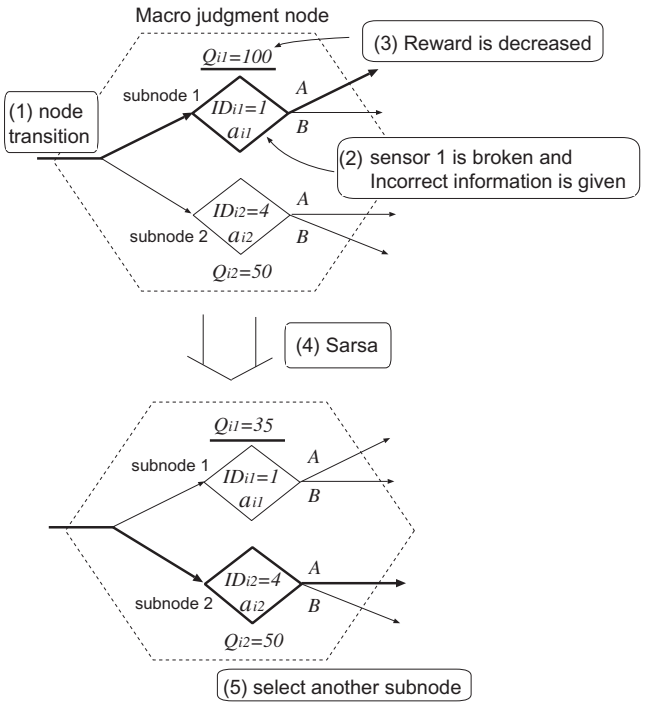

Fig. 5. An example of a recovery process.

the wheel speed. The first feature of this mechanism is that it works without directly detecting the faulty sensors which may require an additional mechanism, but it indirectly detects faulty sensors through the changes of Q values. Therefore, this approach keeps the simplicity of the system. The second feature is the ability to do the adaptation without having to go through another round of evolution which may take a long time. General reinforcement learning can also do this kind of adaptation because it is a online learning method. However, the Q table of GNP-RL is represented by the graph structure where the numbers of nodes and subnodes are fixed at small numbers, which makes quite compact Q table. For example, if the total number of nodes is 15 and the number of subnodes contained in each node is two, the number of Q values is only 30 (=2 × 15).

3. Simulations

The adaptability of GNP-RL is confirmed using a software called WEBOTS (Cyberbotics) which simulates Khepera robot and its operating environment.

3.1. Khepera robot

Khepera robot has eight distance sensors and two wheels. It detects the presence of objects around it using the sensors and moves around the environment using the two wheels. As shown in Fig. 6, the robot has four front sensors, two side sensors, two rear sensors, and two wheels on its right and left sides, respectively. The range of the values returned by the sensors is between zero and 1023. The value of zero means that there is no object detected by the sensor while 1023 means that an object or obstacle is very close to the sensor (almost touches the sensor). Intermediate values have the meaning of the approximated distance between

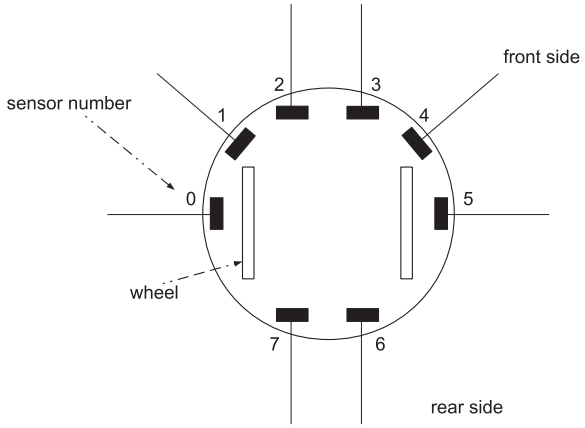

Fig. 6. Khepera robot.

Table 1
Node functions in the library.

Symbol	*ID*	Content
$J_0,\ldots,J_7$	0,...,7	Judge whether the input value from the sensor 0,...,7 is higher than the selected parameter a_{ip}, respectively
P_0	0	Determine the speed of the right wheel
P_1	1	Determine the speed of the left wheel

the sensor and the object. In this paper, the value of zero is always returned from a faulty sensor. The speed range of the wheels is between −10 and +10. Negative and positive speed values have the meaning of backward and forward rotation, respectively.

3.2. Condition of GNP-RL

Table 1 shows the judgment and processing functions. There are eight judgment functions and two processing functions since Khepera robot has eight sensors and two wheels. Each judgment function $\{J_0,\ldots,J_7\}$ returns *A* or *B* as the judgment result. For example, J_2 judges the value of sensor 2 (x_2), and if $x_2 \geqslant a_{ip}$, the judgment result becomes *A*, otherwise *B*. Each processing node sets the speed of the left or right wheel at a_{ip}. The evolution and learning parameters are shown in Table 2. This combination of values is selected because of its good performance in several simulations. The population in each generation is composed of 359 new individuals generated by mutation, 240 new individuals generated by crossover, and one elite individual. The number of judgment nodes is eight and the number of processing nodes is six. Including the start node, the total number of nodes is 15 in every individual. Initially, the *Q* values of the subnodes are set to zero, the connections between nodes are determined randomly, and the values of a_{ip} are also randomly determined within the valid range of a_{ip} (see mutation procedure in Section 2.4.2).

As shown in Fig. 3, a node has several subnodes and the number of them, i.e., m_i, is determined by a designer in advance. In the simulations, several numbers of subnodes are tested in order to find the optimal setting. In detail, GNP-RL with 2, 3, 4, and 5 subnodes are compared in addition to the comparison with standard GNP (SGNP). We can also regard GNP-RL with one subnode as standard GNP.

3.3. Reward and fitness in the wall-following behavior

As descried in Section 2.1, one step ends when 10 or more time units are used. One judgment node takes one time unit and one processing node takes five time units. Therefore, in each step before executing an actual movement of the robot, SGNP and GNP-RL carry out the node transitions within the limited time units and determine the speed of the wheels. Then, the robot actually moves in the environment and obtains a reward. If no processing nodes are executed in a step, the robot takes the same actions, i.e., the same wheel speed as the previous step. In each generation, every individual in the population has to perform the wall-following behavior during one trial/episode. A trial ends when the time step reaches the predefined time step limit (=1000). Every step, the reward obtained by the robot is calculated by Eq. (1), and after a trial ends, the fitness is calculated by Eq. (2). These functions are defined referring to (Nordin, Banzhaf, & Brameier (1998); Floreano & Mondada, 1994). The reward function is designed in order to learn the wall-following behavior, that is, the robot has to move along the wall as fast as and as straight forward as possible. The fitness is an average of all the rewards obtained in a trial. Fig. 7 shows the environment used in the simulation.

Table 2
Simulation parameters.

The number of individuals	600 (mutation: 359, crossover: 240, elite: 1)
The number of nodes	15 (6 processing nodes, 8 judgment nodes and 1 start node)
Parameters of evolution	$P_c = 0.1$, $P_m = 0.01$
Parameters of learning	$\alpha = 0.9$, $\gamma = 0.3$, $\varepsilon = 0.1$

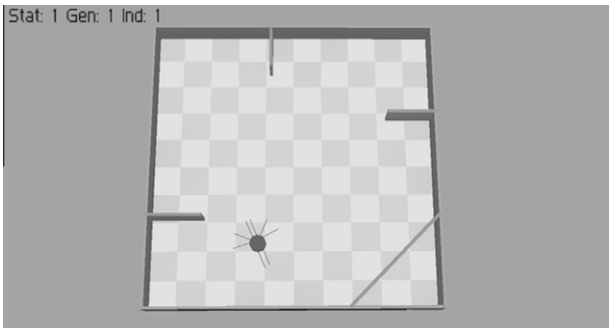

Fig. 7. Simulation environment.

$$\text{Reward}(t) = \frac{v_R(t) + v_L(t)}{20} \times \left(1 - \sqrt{\frac{|v_R(t) - v_L(t)|}{20}}\right) \times C \tag{1}$$

$$\text{Fitness} = \sum_{t=1}^{T} \text{Reward}(t)/T \tag{2}$$

where $v_R(t)$, $v_L(t)$: the speed of right and left wheels at step t, respectively, $C = \begin{cases} 1: \text{all the sensor value are less than 1000, and at least one of them is more than 100} \\ 0: \text{otherwise} \end{cases}$. T : total steps (=1000)

3.4. Training results

The curves in Fig. 8 show the average fitness of Standard GNP (SGNP) and GNP-RLs over 10 independent simulations in the training period. Hereafter, all the simulation results are based on the averages over 10 trials. The numbers attached to the curves indicate the fitness in the last generation. Based on those five figures, GNP-RL with 2 subnodes shows the best training result. Compared to the performance of SGNP, GNP-RLs with 2 and 3 subnodes have higher fitness due to the optimization of node transition by Sarsa. However, if the number of subnodes is more than three, the fitness of GNP-RL becomes lower than that of SGNP because a larger

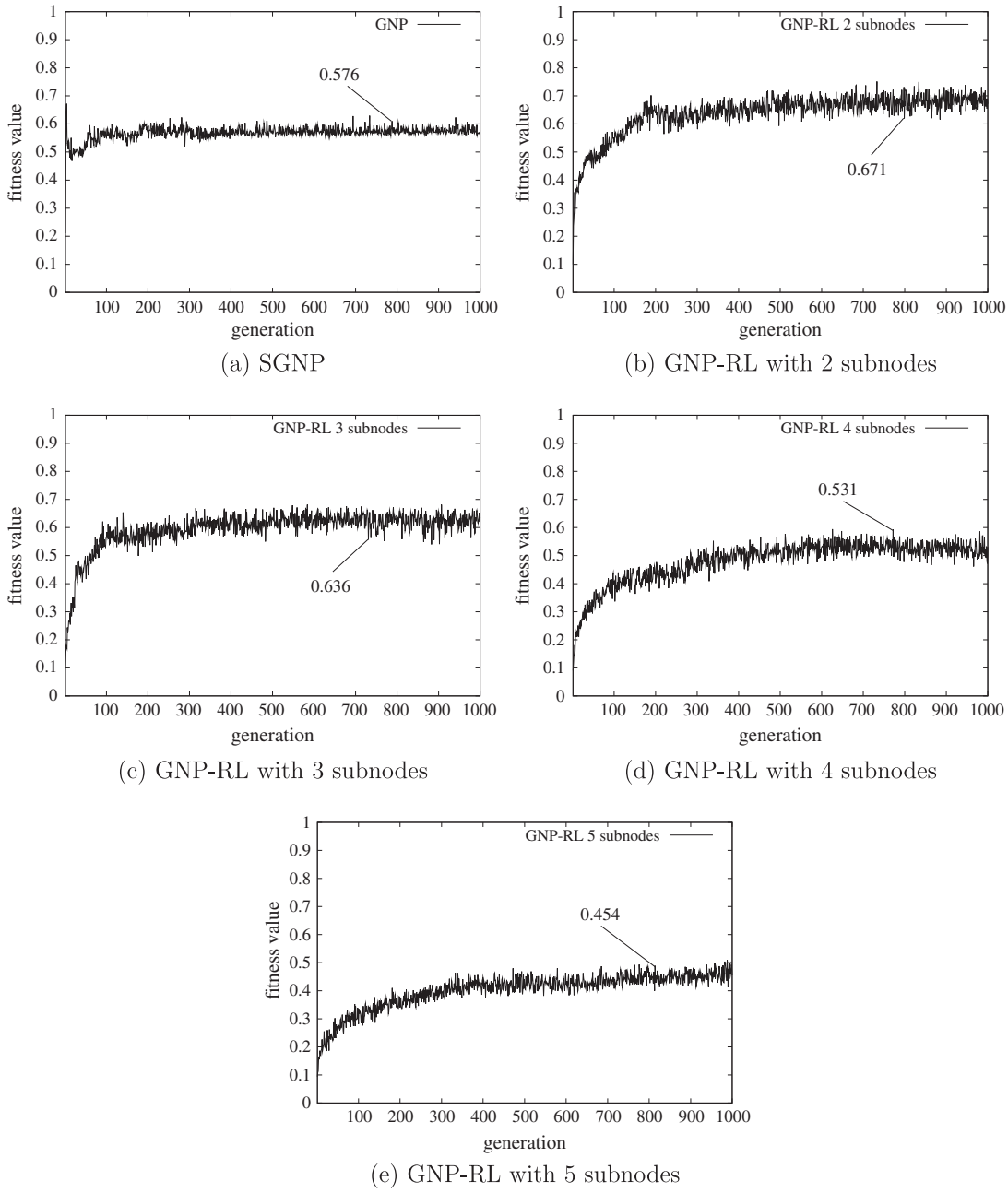

Fig. 8. Fitness curves in the training.

number of subnodes means a larger search space which requires more exploration. It is the reason why GNP-RLs with 4 and 5 subnodes obtain lower fitness and their curves grow more slowly. The number of subnodes also affects the fluctuation of the fitness curves. The fitness curve of SGNP is relatively persistent compared to that of GNP-RLs which tends to fluctuate more. This is caused by the number of options which ε-greedy policy can choose. In other words, the randomness causes the curves of GNP-RLs to fluctuate.

3.5. Testing results

For the purpose of testing the adaptability of SGNP and GNP-RL, two testing cases (simulation I and II) are implemented. In both cases, five sensors are set to faulty and the order of the sensors being set to faulty is sensor 2, 5, 6, 4, and 7. In simulation I, the sensors are gradually set to faulty one by one, while in Simulation II, several sensors are suddenly set to faulty at the same time.

3.5.1. Simulation I

In this simulation, one more sensor is set to be faulty every 1000 steps. It means that the situation becomes severe as the time step goes on because although all the sensors can work in the first 1000 steps, one sensor is set to faulty after 1000 steps and two sensors are after 2000 steps, and so on. This situation simulates a situation where the functions of machines deteriorate as time goes on like actual systems.

Table 3 shows the average rewards when no sensors are faulty, and one, two, three, four and five sensors are faulty. From Table 3, we can see that SGNP has steep performance decline after one or more sensors are faulty although it obtains a relatively high average reward until 1000 steps when all the sensors function properly.

Table 3
Average rewards in Simulation I.

Situation	GNP	GNP-RL$_{2sub}$	GNP-RL$_{3sub}$	GNP-RL$_{4sub}$	GNP-RL$_{5sub}$
No faulty sensor	0.224	0.198	0.188	0.147	0.111
1 faulty sensor	0.096	0.143	0.162	0.136	0.111
2 faulty sensors	0.060	0.111	0.147	0.128	0.109
3 faulty sensors	0.055	0.086	0.135	0.129	0.098
4 faulty sensors	0.033	0.063	0.084	0.116	0.071
5 faulty sensors	0.033	0.058	0.076	0.112	0.067

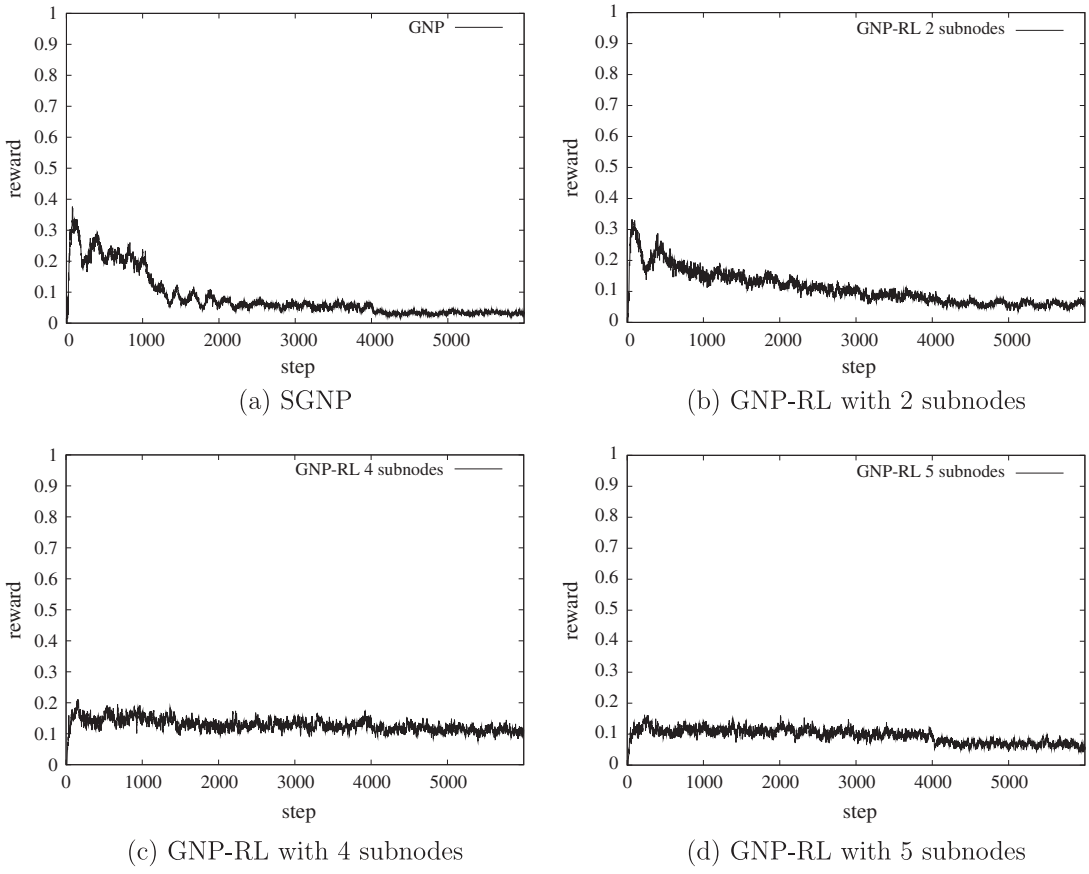

Fig. 9. Reward curves in Simulation I.

All GNP-RLs obtain lower average rewards than SGNP when no sensors are faulty because GNP-RLs are still doing reinforcement learning with ε-greedy policy which explores for better solutions using random actions. After having one or more faulty sensors, the average rewards of all GNP-RLs are higher than that of GNP.

The reward curves of SGNP, GNP-RLs with 2, 4 and 5 subnodes are plotted in Fig. 9, where the vertical axis measures the reward whereas the horizontal axis shows the steps. As described before, all the sensors work normally from step 0 to step 1000 and one sensor is set to faulty from step 1001 to step 2000. According to the curve in Fig. 9 (a), SGNP does not show any sign of recovery after having one malfunctioning sensor. In Fig. 9 (b), the reward curve of GNP-RL with 2 subnodes shows a gradual performance degradation as the number of faulty sensors increases. The curve in Fig. 9 (c) indicates lower rewards than GNP-RL with 2 subnodes in the first 1000 steps, however, the performance degradation of GNP-RL with 4 subnodes is not drastic. Even five sensors are faulty, the average reward is still above 0.1. In the case of GNP-RL with 5 subnodes (Fig. 9 (d)), the rewards become low due to the size of the search space that requires longer training time although the size itself basically gives the opportunity to prepare more alternative functions/actions. In other words, GNP-RL with 5 subnodes searches for better alternative functions in the testing for recovery, but good alternative functions cannot be found in a short time. From the above results, it is clarified that the adaptation mechanism of GNP-RL is effective in the troubled situations and GNP-RL with 4 subnodes has the best adaptability in the given complexity level of the environment.

3.5.2. Simulation II

In simulation II, one or more sensors are set to faulty at 500th step. The sudden changes make it more difficult for GNP-RL to find suitable alternative judgment or processing functions, so longer adaptation time than simulation I is required. The total number of steps is set at 1000, so GNP-RL has to adapt to the changes within 500 steps.

The simulation results are represented in Table 4. As simulation I, SGNP obtains the highest average reward in the first 500 steps. After one or more sensors are set to faulty, GNP-RLs with 2 and 3 subnodes always get higher average rewards than SGNP. GNP-RLs with 4 and 5 subnodes also obtain higher rewards than SGNP in most of the cases where more than one sensors are faulty.

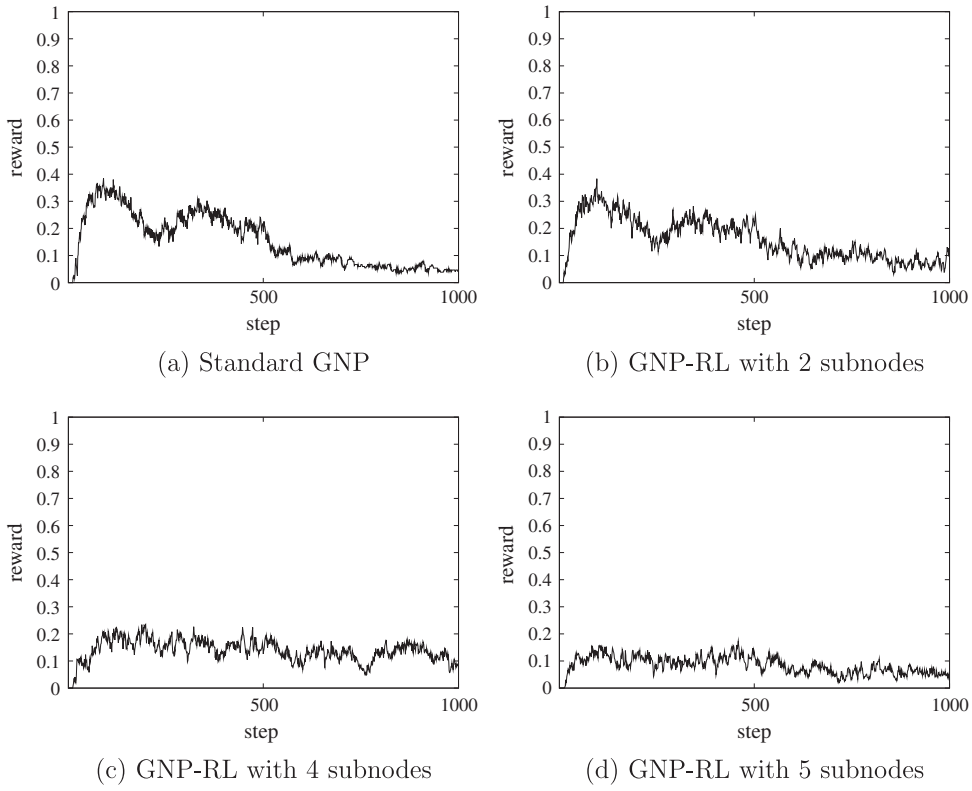

Fig. 10. Reward curves in Simulation II.

Table 4
Average rewards in Simulation II.

Situation	GNP	GNP-RL$_{2sub}$	GNP-RL$_{3sub}$	GNP-RL$_{4sub}$	GNP-RL$_{5sub}$
No faulty sensor	0.212	0.210	0.162	0.137	0.113
1 faulty sensor	0.131	0.173	0.165	0.121	0.103
2 faulty sensors	0.097	0.160	0.147	0.125	0.113
3 faulty sensors	0.099	0.187	0.150	0.142	0.114
4 faulty sensors	0.067	0.135	0.090	0.126	0.070
5 faulty sensors	0.064	0.091	0.098	0.123	0.060

The low average rewards of GNP-RLs with 2 and 3 subnodes when five sensors are faulty show that they do not have enough alternative functions to adapt to the changed situations, although they obtain relatively high average rewards until four sensors become faulty. The reason why GNP-RLs with 2 and 3 subnodes obtain higher rewards when the number of faulty sensors is less than four is as follows. When the environmental changes are not so large, it is easier for GNP-RL with smaller number of subnodes to find suitable alternative functions within the limit time (500 steps). However, when larger changes occur, the small number of subnodes (alternative functions) is not enough to recover from the troubles.

Fig. 10 shows the reward curves of SGNP, GNP-RLs with 2, 4 and 5 subnodes when five sensors are set to faulty after 500 steps. The curve of SGNP in Fig. 10 (a) drops steeply after 500 steps, and that of GNP-RL with 2 subnodes in Fig. 10 (b) also drops, however, it is higher than that of SGNP. The curves of GNP-RLs with 4 and 5 subnodes in Fig. 10 (c) and (d), respectively, have less noticeable drops. The adaptability of GNP-RL with 4 subnodes is considered the best in this test. It has enough alternative functions which are effective for the adaptation when 5 sensors are malfunctioning. The larger search space of GNP-RL with 5 subnodes makes it difficult to find appropriate alternative functions due to the insufficient recovering time, although larger search space provides the ability to have more alternative functions. From the results, GNP-RL with 4 subnodes can find alternative functions in a relatively short period of time and continue performing the desired task although with less rewards than the case where no sensors are faulty. In other words, GNP-RL with 4 subnodes has a good balance between the search space and learning efficiency.

4. Conclusions

This paper describes the adaptation mechanism of GNP-RL and analyzes the recovery abilities. The simulation results show that GNP-RL can adapt to the troubles quickly when the appropriate number of subnodes are prepared. The structure of GNP-RL has an advantage of being able to store several subnodes which work as alternative judgment and processing functions. GNP-RL prepares the alternative functions during the evolution in the training period and uses them for the adaptation when the environment changes in the testing period. This structure gives GNP-RL flexibility to adjust itself to the changing environment. The utilization of the alternative functions is achieved by reinforcement learning (Sarsa) and ε-greedy policy. In the simulations, faulty sensors emulate the environmental changes. When one or more sensors are set to be faulty, the alternative functions are selected to judge other sensors working normally and the speed of the wheels is also set

again. In other words, different action rules are created when the situation changes. It is also found that the number of subnodes has an important effect on the fitness and adaptability, so the number of subnodes has to be appropriately determined. Larger number of subnodes provides GNP-RL the opportunity to have more alternative functions which may increase the adaptation ability, but the adaptation time required to find good alternative functions also increases as the search space becomes larger.

Therefore, the setting of the appropriate number of subnodes is a remaining problem. In the future, we will develop an automatic adjustment mechanism of the number of subnodes in order to determine the appropriate number of alternative functions which keeps the balance between the adaptation ability and search efficiency. In detail, troubled (abnormal) situations could be detected by the changes of Q values or rewards, and the number of subnodes is adjusted by judging whether or not the current program is quickly recovering from the troubles.

References

Cyberbotics. <www.cyberbotics.com/>.

Floreano, D. & Mondada, F. (1994). Automatic creation of an autonomous agent: Genetic evolution of a neural network driven robot. In: *Proceedings of the third international conference on simulation of adaptive behavior*, (pp. 421–430).

Hirasawa, K., Eguchi, T., Zhou, J., Yu, L., & Markon, S. (2008). A double-deck elevator group supervisory control system using genetic network programming. *IEEE Transactions on Systems Man and Cybernetics, 38*(4), 535–550.

Kamio, S., & Iba, H. (2005). Adaptation technique for integrating genetic programming and reinforcement learning for real robots. *IEEE Transaction on Energy Conversion, 9*(3), 318–333.

Koza, J. R. (1992). *Genetic programming, on the programming of computers by means of natural selection*. Cambridge, Mass: MIT Press.

Koza, J. R. (1994). *Genetic programming II, automatic discovery of reusable programs*. Cambridge, Mass: MIT Press.

Mabu, S., Chen, Y., Hirasawa, K., & Hu, J. (2007). Stock trading rules using genetic network programming with actor-critic. In: *Proceedings of the 2007 IEEE Congress on Evolutionary Computation (CEC2007)*, pp. (508–515).

Mabu, S., Tjahjadi, A., Sendari, S., & Hirasawa, K. (2010). Evaluation on the robustness of genetic network programming with reinforcement learning. In: *Proceedings of the 2010 IEEE International Conference on Systems, Man, and Cybernetics*, (pp. 1659–1664)

Mabu, S., Hatakeyama, H., Thu, M. T., Hirasawa, K., & Hu, J. (2006). Genetic network programming with reinforcement learning and its application to making mobile robot behavior. *IEEJ Transaction on EIS, 126*(8), 1009–1015.

Mabu, S., Hirasawa, K., & Hu, J. (2007). A graph-based evolutionary algorithm: Genetic network programming (GNP) and its extension using reinforcement learning. *Evolutionary Computation, 15*(3), 369–398.

Nordin, P., Banzhaf, W., & Brameier, M. (1998). Evolution of a world model for a miniature robot using genetic programming. *Robotics and Autonomous Systems, 25*, 105–116.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement Learning – An Introduction*. Cambridge, Massachusetts, London, England: MIT Press.