ICROS-SICE International Joint Conference 2009
August 18-21, 2009, Fukuoka International Congress Center, Japan

Robust Genetic Network Programming Using SARSA Learning for Autonomous Robots

Sung Gil Park, Shingo Mabu and Kotaro Hirasawa

Graduate School of Information, Production and Systems, Waseda University
2-7 Hibikino, Wakamatsu-ku, Kitakushu, Fukuoka 808-0135, Japan
(E-mail: sgrhermes@ruri.waseda.jp, mabu@aoni.waseda.jp, hirasawa@waseda.jp)

Abstract: Adaptive and robust control has attracted increasing attention in the field of artificial intelligence. Adaptive controller makes use of some adaptation mechanisms which are designed to learn explicitly the unknown parameters of the system and the uncertain situation under control. An evolutionary algorithm called ”Genetic Network Programming, GNP” has been already proposed to control intelligence systems. In conventional GNP, when they have a certain problems such as the hardware or software of the system, the nodes and connections of GNP may not work well. In this paper, GNP with SARSA Learning is applied to construct the robust GNP for autonomous robots.

Keywords: Genetic Network Programming, Evolutionary Computation, Reinforce Learning, KHEPERA robot

1. INTRODUCTION

Interest in autonomous robotics is increased in the industries such as space exploration robots in the high-tech industry or home cleaning robots in the housing industry. The autonomous robotics and robust systems are necessary studies for intelligence robotics. Adaptive controller makes use of some adaptation mechanisms which are designed to learn explicitly the unknown parameters of the system and uncertain situations. An evolutionary algorithm called ”Genetic Network Programming, GNP” has been proposed to control intelligent systems. GNP represents its solutions as graph structures, which can improve the expression ability and performances [1-2]. GNP showed the efficient results in autonomous robotics. GNP can determine an action by not only the current, but also the past information. The graph structure of GNP has an implicit memory function and the ability to re-use nodes, where compact structures are obtained. On the other hand, GNP combined with Reinforcement Leaning [3] (GNP-RL) [4-5] has been also studied. GNP-RL can search for better solutions during every judgment and processing in task execution, besides the evolutional operations executed after the task. And it also takes the advantage of the sophisticated diversified search ability of evolution and the intensified search ability of learning. However, it is hard to reuse the function in a certain circumstance like hardware and software problems in testing field.

In this paper, in order to find more robust structures of GNP for dealing with such problems, Robust GNP with SARSA Learning is proposed. The Robust GNP aims at designing controllers that lead the system to a desired performance despite the presence of uncertainties.

Robust GNP with SARSA Learning is applied to the KHEPERA robot and compared to the conventional method of GNP.

2. ROBUST GNP WITH SARSA LEARNING

2.1 Basic Structure of GNP-SARSA

Robust GNP-SARSA has a number of judgment nodes and processing nodes. Fig. 1 is the@graph structure of Robust GNP-SARSA.

Judgment nodes have conditional branch decision functions. Each judgment node returns a judgment result and determines the next node to be executed. Processing nodes determine the agent’s actions / processings in order to achieve the task. Processing nodes have no conditional branch.

Each node has a number of sub nodes which have its own function. In each node, one of the sub nodes is selected by SARSA-Learning based on Q values.

In Robust GNP-SARSA, when some problems occur in the nodes, the function (sub node) to be executed in the node is changed to another function to overcome the problems by SARSA-Learning and to improve the robustness of GNP.

2.2 Gene Structure of GNP-SARSA

Human has a chromosome and it has a lot of information called gene. Similarly, the structure of GNP-SARSA is determined by the combination of the genes. Fig. 2 is a basic gene structure of GNP-SARSA. The gene has the node number which is *i* (0≤*i*≤*n*-1, *n* is the total number of nodes). The first part of the gene is the node gene. K_i represents the node type. There are three types of nodes, start node, judgment node and processing node. K_i=1 means judgment node and K_i=2 means processing node. ID_i is a code number of the judgment or processing, and it is represented as a unique number shown in the LABRARY. GNP-SARSA has time delays. The time delay of GNP-SARSA spent on judgment or processing is represented by d_{im}. And d^A_{im}, d^B_{im} are the ones spent on the transition from node *i* to the next nodes. The superscripts *A*, *B*,$\cdots$ represent the judgment results, for example, d^A_{i1} shows the time delay when the judgment result is ”*A*”. *m* shows the sub node number. *A* judgment

PR0002/09/0000-0523 ¥400 © 2009 SICE

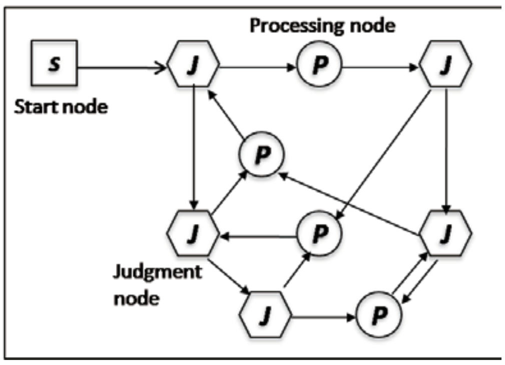

Fig. 1 Basic graph of Robust GNP-SARSA

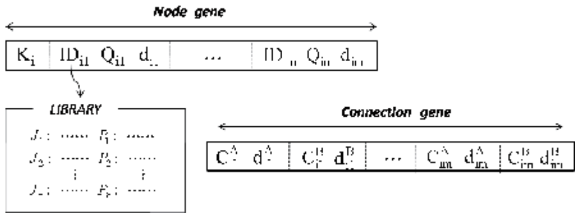

Fig. 2 Basic graph of Robust GNP-SARSA

node has some sub nodes which have their own judgment functions, and one of them is selected by SARSA. In this paper, for simplicity, d_{im}^A, d_{im}^B are set at zero time unit, d_{im} of each judgment node and processing node are set at one time unit. The one step is defined in such a way that one step ends when an agent uses 10 time units or more time units. Q_{im} shows the *Q* value of sub node *im*. C_{im}^A, C_{im}^B show the node number of the next node from sub node *im*.

2.3 Judgment Node

As a human can analyze the information to take an action, a judgment node has a function to judge environments to take an action. Fig. 3 shows a judgment node of Robust GNP-SARSA. Here, the proposed method is applied to the KHEPERA robot which has 8 sensors. The information comes from the sensors to the judgment node. Each judgment node has several judgment functions (J_1, J_2, $\cdots$) in the node, and one of them are selected by ε-greedy policy[3]. That is, with the probability of 1-ε, the sub node with the maximum *Q* value is selected, and with the *Q* value is selected, and with the probability of ε, the sub node is randomly selected. Suppose the sub node with Q_{i1} is selected and its function is J_1(judge sensor 1), the following procedure is executed. If the value x_1 of sensor 1 is equal or larger than threshold *a*, the next node becomes node C_{i1}^A. Otherwise it becomes C_{i1}^B.

2.4 Processing Node

Agents take actions when the current node is a processing node. Fig. 4 shows the processing node of Robust GNP-SARSA. In the processing node, some robot actions are prepared as sub nodes. And one of the actions is selected based on the *Q* values. The KHEPERA robot has two wheels which are the left one and the right one. The actions represent left and right wheel speed. The speed

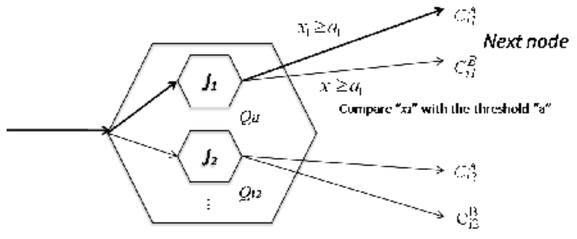

Fig. 3 Judgment Node *i* of Robust GNP-SARSA

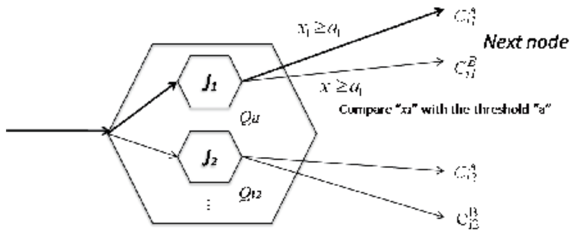

Fig. 4 Processing Node *i* of Robust GNP-SARSA

of the KHEPERA robot can take the range from -10 to +10. The plus value means going forward and the minus means going backward. Each processing node has several processing functions (P_1, P_2, $\cdots$), and one of them are selected by ε-greedy policy. For example, if the sub node P_{i1} is selected, the next node is C_{i1}^A.

2.5 Evolution Phase

Fig. 5 shows the whole flowchart of GNP-SARSA. An, initial population of randomly generated candidate solutions comprises the first generation. The fitness function is applied to the candidate solutions. Parents for the genetic operations are chosen by tournament selection, and the offspring is generated by crossover and mutation.

2.5.1 Crossover

1. Select two parents using tournament selection.

2. Each node is selected as a crossover node with the probability of P_c.

3. Two parents exchange the genes of the corresponding crossover nodes.

4. Generated new individuals become the new ones in the next generation.

2.5.2 Mutation

1. Select one individual using tournament selection.

2. Mutation operator.

2-1. connection: Each branch becomes connected to other node randomly with the probability P_m.

2-2. sensor number: Each function of sub nodes in judgment nodes is changed to other one randomly with the probability P_m.

2-3. speed value: Each function of sub nodes in processing nodes is changed to other one randomly with the probability P_m.

3. Generated new individual becomes the new one in the next generation.

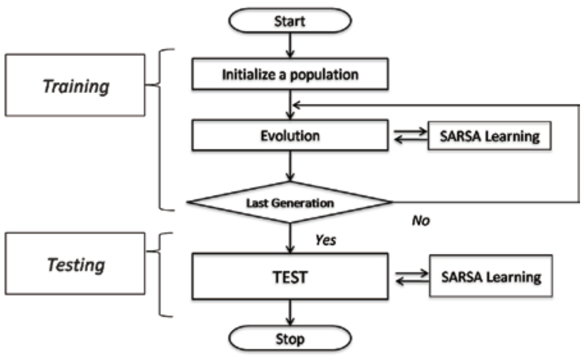

Fig. 5 Flow Chart

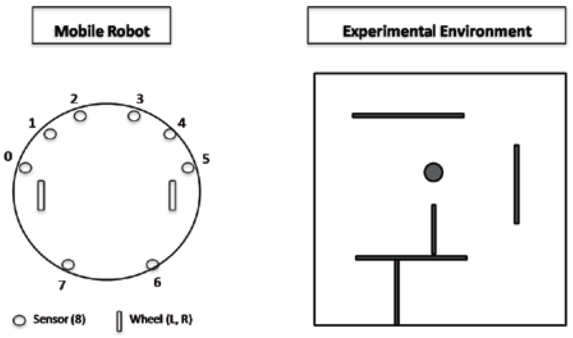

Fig. 6 Experiment Environment

2.6 Learning Phase

The SARSA Learning updates Q value based on the following equation.

$$\begin{aligned} Q(s_t, a_t) \quad \leftarrow \quad & Q(s_t, a_t) + \\ & \alpha[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \quad (1) \end{aligned}$$

where,

s_t	: current state(node) at time step *t*
a_t	: function of node selected at state s_t
r_t	: reward at time step *t*
a	: learning rate($0<a\leq 1$)
γ	: discount rate($0<\gamma\leq 1$)

2.6.1 The procedure of SARSA Learning

1. At time t, GNP refers to $Q_{i1}, Q_{i2}, \cdots$ of the node *i* and select one of them based on the Q values.
2. GNP executes the function of the sub node with the selected Q value.
3. At time *t+1*, the current node is changed to the next node *j* and GNP selects one Q value in the same way as step 1.
4. Q value is updated as equation (1).
5. $t \leftarrow t+1$, $i \leftarrow j$.
6. Steps 2 5 are repeated until the final step.

2.6.2 The function of SARSA Learning

SARSA Learning is concerned with how an agent ought to take actions in the environment directly from the experiences without a model environment. The method updates Q values based on other learned estimates, without waiting a final outcome. Even if an accident occurs, the learning procedure is the same (step1 5) to overcome the accident. For example, when an accident occurs at sub node *i1* in Fig. 3, that is, sensor 1 is broken, GNP still executes the function of the sub node with the selected Q value and Q value is updated based on equation (1). So, the Q value of the selected sub node is getting decreased because of the accident, that is, the robot could not work because of the accident. When the Q value becomes smaller than the Q values of other sub nodes, the sub node to be selected is changed to the appropriate one based on the Q values. Finally, the robot can work again adaptively.

Table 1 Simulation Condition

Generation	1000
Individual	300
Best Individual	1
Judgment nodes	8
Processing nodes	2
Crossover Rate	0.1
Mutation Rate	0.01
Learning Rate	0.7
Discount Rate	0.3

Table 2 Nodes Functions

Symbol	ID	Content
$J_1,\cdots,J_8$	1,$\cdots$,8	Judge the value of the sensor 1,$\cdots$,8
P_1	1	Determine the speed of the right wheel
P_2	2	Determine the speed of the left wheel

3. SIMULATIONS

In order to evaluate the ability of the proposed method, it is applied to the controller of the KHEPERA robot.

3.1 Settings of the KHEPERA

Fig. 6 shows the experiment environment of the simulations. The simulated KHEPERA robot includes 8 infrared sensors allowing it to detect the proximity of objects in front of it, behind it and to the right and left side of it by reflection. Each sensor returns a value ranging between zero and 1023. 1023 means that an obstacle (wall) is very close to the sensor, while zero means that no obstacle is perceived. Intermediate values may give an approximate idea of the distance between the sensors and obstacle. Two motors turn the right and left wheels of the robot, respectively, and they can take the speed value ranging between -10 and +10.

3.2 Settings of GNP-SARSA

Table.1 shows the simulation conditions. There are 300 individuals and the generation is 1000 to get the best individual for the robot. 8 judgment nodes, 2 processing nodes and 1 start node are prepared. The parameters of evolution such as crossover rate and mutation rate are set

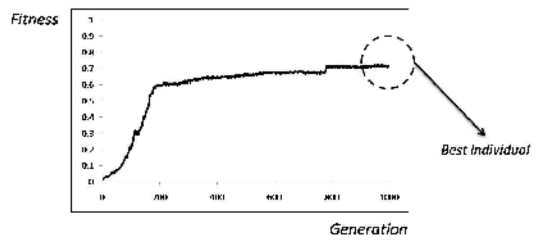

Fig. 7 Fitness curve in the training

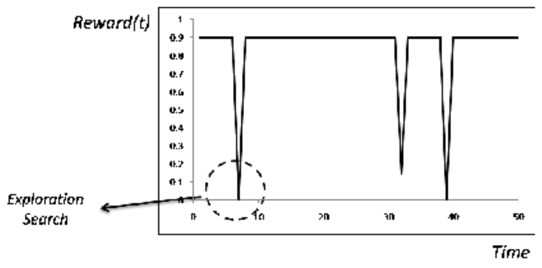

Fig. 8 Change of rewards obtained by Robust GNP-Sarsa

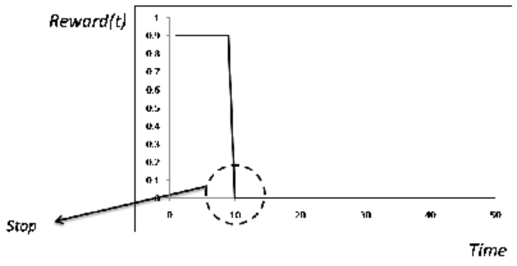

Fig. 9 Change of rewards obtained by GNP without Sarsa

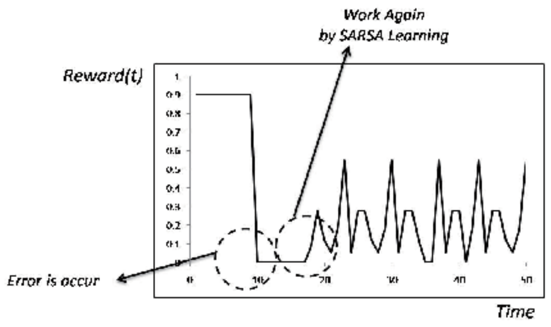

Fig. 10 Change of rewards obtained by Robust GNP-Sarsa when the accident occurs

at 0.1, 0.01, respectively. At the end of each generation, 170 new individuals are generated by crossover, 120 new individuals are generated by mutation, and 10 elite individuals are selected to the next population.

Table.2 shows the nodes functions. There are eight kinds of judgment nodes which judge the eight sensors, respectively. There are two kinds of processing nodes which control the speed of the right wheel and the left wheel, respectively.

3.3 Reward and Fitness Functions

A reward function defines the goal for the robust GNP-SARSA. The sole objective of the agent is to maximize the total reward it receives in the running period. In this paper, the KHEPERA robot is to do the as Wall-following behavior. In this simulation, one step includes 10 time units, each judgment node spends one time unit, and processing node spends five time units. In each step, GNP judges the values of the sensors and determines the speed of the wheels based on Q values. The total step is 1000 and then fitness is calculated. Robot learns the wall-following behavior to maximize the fitness. The reward function is the following.

$$\text{Reward}(t) = \frac{V_R(t)+V_L(t)}{20}\times (1-\sqrt{\frac{|V_R(t)+V_L(t)|}{20}})\times C \qquad (2)$$

where,

$V_R(t)$	: right wheel speed at time step *t*
$V_L(t)$	: left wheel speed at time step *t*
$C=1$	: all the sensor values are less than 1000, and at least one of them is more than 100
$C=0$	: otherwise

A fitness function is a particular type of objective function that quantifies the optimality of a solution in GNP-SARSA.

$$\text{Fitness}(k) = \sum_{t=1}^{K} \text{Reward}(t) \qquad (3)$$

where,

K : total time step

3.4 Training and Testing Results

Fig. 7 shows the fitness curve in the training simulation where the individuals to be used in the testing simulation one evolved. The fitness value is very low at the beginning of the generations, but the individuals are evolved by the genetic operations during evolution. So, the fitness improves as the generation goes on.

Fig. 8 shows the testing result using the best individual which is obtained by evolution. Using this individual, the robot works well in the environment. During the testing sometimes the rewards were small because SARSA-Leaning has an exploration function to explore unknown environments to find better actions.

Fig. 9 shows the testing result without SARSA when the accident occurs. The accident is that one of the sensors is broken by the accident, so the sensor returns incorrect information to the judgment node. If the front sensor is broken, the robot tries to go forward even if there is an obstacle. At the time t=10 in Fig. 9, robot can not move because of the accident.

Fig. 10 shows the testing result when the accident occurs, but SARSA-Learning overcomes the accident. When the accident occurs, the robot can not move as usual for a while after the accident. Since the reward of the sub node related to the accident is getting smaller, the node transition is changed to the appropriate sub node to overcome the accident by SARSA-Learning. Finally, the robot moves again step by step.

4. CONCLUSIONS

In this paper, Robust Genetic Network Programming with SARSA-Learning (Roust GNP-SARSA) is proposed. SARSA Learning is concerned with how an agent ought to take actions in environments directly from experiences without a model, so, GNP can change the node transition in order to control the robot well in accidents. The proposed method can get a good result, when the system is damaged and the correct sensor information can not be obtained. Henceforth, the generalization ability of Robust GNP-SARSA is studied in more detail and it is executed in more complicated environments.

REFERENCES

[1] S. Mabu, K. Hirasawa and J. Hu, ”A Graph-based Evolution Algorithm: Genetic Network Programming (GNP) and Its Extension Using Reinforcement Learning”, Evolutionary Computation, MIT Press, Vol. 15, No. 3, pp. 369-398, 2007.

[2] H. Katagiri, K. Hirasawa and J. Hu, ”Genetic network programming-application to intelligent agents”, in Proc. of the IEEE International Conference on System, Man and Cybernetics, pp. 3829-3834, 2000.

[3] R. S. Sutton and A. G. Barto, ”Reinforcement Learing”, MIT Press Cambridge, Massachusetts, London, England, 1998.

[4] S. Mabu, K. Hirasawa and J. Hu, ”Genetic Network Programming with Learning and Evolution for Adapting to Dynamical Environments,” In Proc. of the Congress on Evolutionary Computation, pp.67-76, 2003.

[5] S. Mabu, K. Hirasawa and J.Hu, ”Genetic Network Programming with Reinforcement Learnging and its Performance Evaluation,” in Proc. of the Genetic and Evolutionary Computation Conference, part 2, pp. 710-711, 2004.