TELKOMNIKA, Vol.17, No.3, June 2019, pp.1447~1454
ISSN: 1693-6930, accredited First Grade by Kemenristekdikti, Decree No: 21/E/KPT/2018
DOI: 10.12928/TELKOMNIKA.v17i3.12232

Exploration of genetic network programming with two-stage reinforcement learning for mobile robot

Siti Sendari*[1], Arif Nur Afandi[2], Ilham Ari Elbaith Zaeni[3], Yogi Dwi Mahandi[4], Kotaro Hirasawa[5], Hsien-I Lin[6]
[1,2,3,4]Department of Electrical Engineering, Universitas Negeri Malang,
Jalan Semarang No. 5 Malang, Jawa Timur 65145, Indonesia
[5]Graduate School of Information, Production and Systems, Waseda University,
Hibikino 2-7, Wakamatsu-ku, Kitakyushu, Fukuoka 808-0135, Japan
[6]Graduate Institute of Automation, National Taipei Univeristy of Technology,
Rm 801A, Integrated Technology Complex, I, Sec. 3, Chung-Hsiao E Road, Taipei 106, Taiwan
*Corresponding author, e-mail: siti.sendari.ft@um.ac.id

Abstract

This paper observes the exploration of Genetic Network Programming Two-Stage Reinforcement Learning for mobile robot navigation. The proposed method aims to observe its exploration when inexperienced environments used in the implementation. In order to deal with this situation, individuals are trained firstly in the training phase, that is, they learn the environment with ϵ-greedy policy and learning rate α *parameters. Here, two cases are studied, i.e., case A for low exploration and case B for high exploration. In the implementation, the individuals implemented to get experience and learn a new environment on-line. Then, the performance of learning processes are observed due to the environmental changes.*

Keywords*: genetic network programming, inexperienced changes*, *two-stage reinforcement learning*

1. Introduction

A mobile robot using reactive strategies determines its behavior based on the sensory information, where the robot carries out a simple task, such as the wall following, obstacle avoidance or object following behaviors. Futhermore, the main mobile robot navigation problem is to follow the wall are the wall and its enviromenment changes. In order to build navigation systems based on the reactive strategies workable in unknown environments, the robustness to the changes of the environments should be considered.

Reinforcement Learning (RL) [1] is an attractive method to provide adaptation mechanisms in the dynamic environments through trial and error, where the rewards are given by the environments depending on the actions taken by the agent. The objective of the agent is to maximize the rewards, and RL learns a policy to maximize the accumulated rewards. Many researches [2–5] show that RL is well suited to learn control policies for mobile robot navigations.

State of the art in this research can be seen that the integration of RL to Evolutionary Algorithms (EA), such as Genetic Algorithm (GA) [6–8], Genetic Programming (GP) [9–11] and Genetic Network Programming (GNP) [12] were studied in many researches [13–17], where the integration can improve the performance as shown in GNP with RL (GNP-RL) which was implemented to navigate the mobile robot [18]. EA has the evolving ability for capturing the environment using selection, crossover and mutation, while the integration of RL to EA improves the adaptability to the dynamic environments.

The aim of this research is to observe the robustness to the changes of the environments by using Genetic Network Programing, several effective mechanisms were studied, such as (1) adding noises during the training phase [14, 19, 20]; (2) introducing the two-stage reinforcement learning structure [21, 22]; and (3) controlling parameter learning [23–25]. The first method improves the exploration ability of the agent in the training phase, then the agent becomes more robust when facing inexperience situations in the implementation with noises. Here, the proposed method is to get the effectiveness of the learning mechanisms of RL. The learning mechanism is applied to the second method, where a large search space is separated in two stages, so that the actions can be determined more appropriately. The third method introduces a mechanism to control the duality of exploitation and exploration, which have the ability of re-learning quickly and flexibly when sudden changes occur in the environments [26].

Received October 11, 2018; Revised January 9, 2019; Accepted January 29, 2019

The proposed navigation system of the mobile robot in this paper is based on GNP, where GNP has advantages [12] such as (1) re-usability of the nodes which make the structures more compact and (2) applicability to Partially Observable Markov Decision Problem (POMDP). Compared to the other methods, such as Evolutionary Neural Network (ENN) and GP, GNP has better performance [12, 18]. Here, GNP with Two-Stage Reinforcement Learning (GNP-TSRL) to face inexperienced changes of the environments was studied. TSRL has two kinds of RLs represented by two Q-tables, that is, a Q-table for sub node selection (SS method) and Q-table for branch connection selection (BS method). The actions selections of SS and BS methods are carried based on ϵ greedy policy. This paper is organized as follows. Section 2 describes a mechanism of GNP-TSRL with ϵ-greedy policy and learning rate α. Section 3 shows the simulation conditions and results. Finally, conclusion and future work are given in section 4.

2. Two Stage Reinforcement Learning (TSRL) with Changing Mechanism

This section describes a mechanism of changing ϵ and α of GNP-TSRL structures.

2.1. Structures of GNP-TSRL

The structures of GNP-TSRL consist of a start node and a fix number of processing nodes and judgment nodes, which are connected to each other as a directed graph as shown in Figure 1. The start node has no function and its only role is to determine the first node to be executed, while the judgment nodes have functions to judge the assigned inputs (sensor values), return the judgment results and determine the next node in the transitions. In the former paper [20], it is found that the integration of fuzzy logic into the judgment nodes (fuzzy judgment nodes) can perform well in the noisy environments to determine the node transitions probabilistically, therefore the fuzzy judgment nodes are still used in the proposed method. On the other hand, the function of the processing nodes is for agent to do the actions, that is, to set the speed of the wheels of a Khepera robot. In order to do the effective learning using GNP-TSRL, the structures of the nodes of the conventional GNP-RL are modified, i.e., while the conventional GNP-RL has sub nodes for the the alternative functions [15], GNP-TSRL has not only sub nodes for the alternative functions, but also several branches for the alternative connections. The structures of the judgment nodes and processing nodes of GNP-TSRL are shown in Figure 2. The gene structure of node i is shown in Figure 3, which is divided into the macro node part, sub node part and branch part.

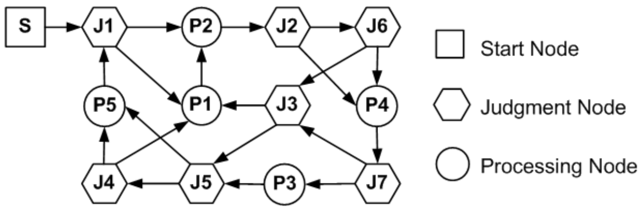


Figure 1. Phenotype structure of a fuzzy GNP-TSRL

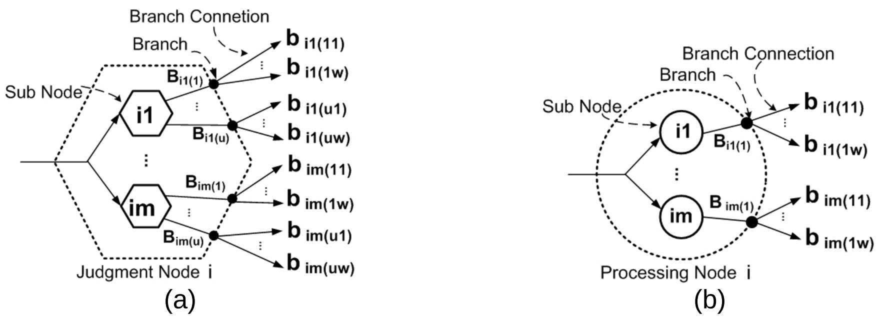


Figure 2. Fuzzy judgment node and processing node of GNP-TSRL
(a) judgment node structure, (b) processing node structure

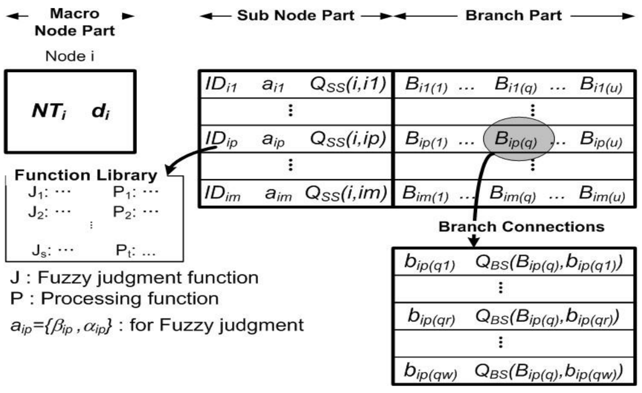

Macro Node Part
Sub Node Part
Branch Part
Node i
NT_i d_i
Function Library
J1: ··· P1: ···
J2: ··· P2: ···
Js: ··· Pt: ...
J : Fuzzy judgment function
P : Processing function
$a_{ip}=\{\beta_{ip},\alpha_{ip}\}$: for Fuzzy judgment
ID_{i1} a_{i1} $Q_{SS}(i,i1)$
ID_{ip} a_{ip} $Q_{SS}(i,ip)$
ID_{im} a_{im} $Q_{SS}(i,im)$
$B_{i1(1)}$ ... $B_{i1(q)}$ ... $B_{i1(u)}$
$B_{ip(1)}$ ... $B_{ip(q)}$ ... $B_{ip(u)}$
$B_{im(1)}$ ... $B_{im(q)}$ ... $B_{im(u)}$
Branch Connections
$b_{ip(q1)}$ $Q_{BS}(B_{ip(q)},b_{ip(q1)})$
$b_{ip(qr)}$ $Q_{BS}(B_{ip(q)},b_{ip(qr)})$
$b_{ip(qw)}$ $Q_{BS}(B_{ip(q)},b_{ip(qw)})$

Figure 3. Genotype of GNP-TSRL

The macro node of node i is defined by NT_i and di. NT_i represents a node type, that is, NT_i=0, 1, 2 encodes the start node, judgment node and processing node, respectively. d_i represents the time delay spent on executing node i, for example in this paper, d_i=0 on the start node, d_i=1 on the judgment node and d_i=5 on the processing node. When the sequence of nodes called node transition uses at least 10 time units, it is defined as one time step of the GNP-based agent behavior. For example, after executing three judgment nodes and one processing node, if another processing node is executed, the total time delay is 13 time units, it means that one time step of GNP is executed.

The node i has m sub nodes as shown in Figure 2 whose functions are described in the sub node part as shown in Figure 3. The node function of sub node $ip \in \{i1,\dots,im\}$ is defined by ID_{ip}, a_{ip} and $Q_{SS}(i,ip)$. ID_{ip} is a code number of the judgment/processing node, which is represented by a unique number shown in the function library. When the node is a judgment node, ID_{ip} represents the sensor number of a Khepera robot, e.g., ID_{ip} = 0 means that sensor number 0, etc. However, when the node is a processing node, ID_{ip} = 0 means the speed of the right wheel of a Khepera robot, while ID_{ip} = 1 means that of the left wheel. a_{ip} is a parameter of the judgment/processing nodes. Because the fuzzy judgment nodes are used in the proposed method, $a_{ip} = \{\beta_{ip},\alpha_{ip}\}$ represents the parameters of fuzzy membership functions. On the other hand, when the node is a processing node, a_{ip} represents the speed of the wheel of a Khepera robot. $Q_{SS}(i,ip)$ means the Q value of SS method, which is assigned to each state-action pair, i.e., the state is node i, and the action is sub node ip selection. Here, the Q_{SS} value is updated using Sarsa learning in the first stage of RL.

The branch part of GNP-TSRL has a unique feature [23]. When the number of the judgment results is u, sub node ip has the branch of $B_{ip(1)},\dots,B_{ip(q)},\dots,B_{ip(u)}$. On the other hand, the processing node has only $B_{ip(1)}$. While the conventional GNP-RL has only one branch connection for each branch, GNP-TSRL has several branch connections for each branch, i.e., w. Branch $B_{ip(q)}$ has branch connections of $b_{ip(q1)},\dots,b_{ip(qr)},\dots,b_{ip(qw)}$. The Q-value of branch connection $b_{ip(qr)} \epsilon \{b_{ip(q1)},\dots,b_{ip(qw)}\}$ is represented by $Q_{BS}(B_{ip(q)},b_{ip(qr)})$, which is updated using Sarsa learning in the second stage of RL.

2.2. Reinforcement Learning (RL)

RL studies the interaction between agents and the environments to adapt to the dynamic environments based on trial and error. The goal of RL is to learn a policy $\pi(s,a)$ by selecting action a at state s to maximize expected cumulative reward R_t. In the POMDP, the agent observes the state using incomplete information on the state, where the actions are more appropriately determined by ϵ-greedy policy to learn the near optimum behavior, where on-line learning by Sarsa algorithm [1] estimates $Q^{\pi}(s,a)$ as follows,

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r(t) + \gamma Q(s',a') - Q(s,a)) \quad (1)$$

where, α is learning rate such that $0 < \alpha \leq 1$. In GNP-RL, the current state is current node i and the action is sub node selection ip [15].

2.3. Algorithm of GNP-TSRL

In order to improve the adaptability of agents in the dynamic environments, the effective learning is done using TSRL [21]. In GNP-TSRL, the following two RLs are combined, that is, (1) RL with Sub Node Selection (SS method) and (2) RL with Branch Connection Selection (BS method) as shown in Figure 4. Thus, GNP-TSRL has two Q-tables, that is, Q_{SS}-table and Q_{BS}-table. SS Method. SS Method is carried out in the first stage of RL using GNP-RL, where the current state is current node i and the action is sub node selection ip.

BS Method. After executing the sub node selected in SS method, one of the several branches from the sub node is determined. Then, the branch connection is determined in the second stage of RL by using BS method. Here, the state is represented by branch $\left(B_{ip(q)}\right) \in \left\{B_{ip(1)}, \dots, B_{ip(u)}\right\}$, while the action is represented by branch connection selection $b_{ip(qr)} \in \{b_{ip(q1)}, \dots, b_{ip(qw)}\}$. The procedure of updating GNP-TSRL using Sarsa learning is explained in Figure 5.

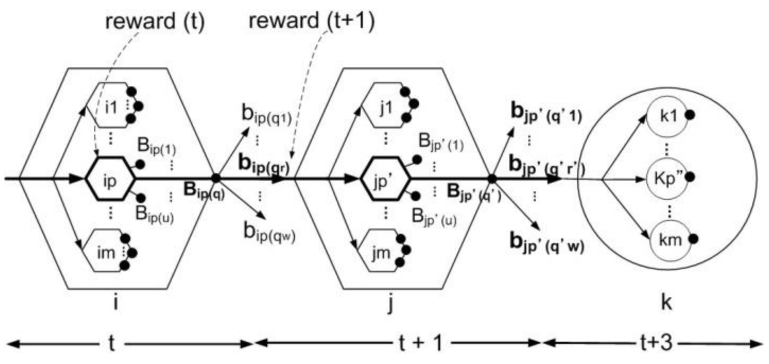


Figure 4. Sarsa learning for TSRL

1. At time step t, it is supposed that the current node is i, and GNP-TSRL refers to all Q_{SS} values for sub node selection, i.e., $\{Q_{SS}(i, i1), \dots, Q_{SS}(i, im)\}$, and selects one of them based on ϵ-greedy policy. It is supposed that GNP selects $Q_{SS}(i, ip) \in \{Q_{SS}(i, i1), \dots, Q_{SS}(i, im)\}$, and the corresponding node function ID_{ip} and parameter a_{ip} are selected.
2. Then, GNP-TSRL executes function ID_{ip} using parameter a_{ip} and determine the branch, which is supposed to be $B_{ip(q)} \in \left\{B_{ip(1)}, \dots, B_{ip(u)}\right\}$.
3. In this step, the second stage of RL, i.e., branch connection selection is started. GNP-TSRL refers to all Q_{BS} of the branch connection selection resulted from step 2, i.e., $\{Q_{BS}\left(B_{ip(q)}, b_{ip(q1)}\right), \dots, Q_{BS}\left(B_{ip(q)}, b_{ip(qw)}\right)\}$ and select one of them based on ϵ-greedy policy. It is supposed that $Q_{BS}\left(B_{ip(q)}, b_{ip(qr)}\right) \in \{Q_{BS}\left(B_{ip(q)}, b_{ip(q1)}\right), \dots, Q_{BS}\left(B_{ip(q)}, b_{ip(qw)}\right)\}$ is selected, then the next node is node j which is connected by $b_{ip(qr)}$.
4. GNP gets reward $r(t)$.
5. At time step $t + 1$, GNP-TSRL repeats step 1 to 3 for node j. Here supposed that $Q_{SS}(j, jp')$ and $Q_{BS}\left(B_{jp'(q')}, b_{jp'(q'r')}\right)$ are selected.
6. Then, the Q values are updated by the following procedures.

$$Q_{SS}(i, ip) \leftarrow Q_{SS}(i, ip) + \alpha\left(r(t) + \gamma Q_{SS}(j', jp') - Q_{SS}(i, ip)\right) \tag{2}$$

$$Q_{BS}\left(B_{ip(q)}, b_{ip(qr)}\right) \leftarrow Q_{BS}(B_{ip(q)}, b_{ip(qr)}) + \alpha\left(r(t+1) + \gamma Q_{BS}\left(B_{jp'(q')}, b_{jp'(q'r')}\right) - Q_{BS}\left(B_{ip(q)}, b_{ip(qr)}\right)\right) \tag{3}$$

   where α is a learning rate $(0 < \alpha \leq 1)$ and γ is discount rate $(0 \leq \gamma \leq 1)$.
7. $t \leftarrow t + 1$, $i \leftarrow j$, $p \leftarrow p'$ and $q \leftarrow q'$. Then, it returns to step 4.

Figure 5. The procedure of updating GNP-TSRL

3. Simulations Settings

The proposed method is used to navigate a Khepera robot in the dynamic environments. This section describes the simulation settings and the results in the training and implementation phases.

3.1. Khepera Robot

The proposed method is simulated to Khepera robot. It has eight infrared distance sensors which are used to perceive objects in front of it, behind of it, to the right and left of it by its reflection. Each sensor returns a value ranging between zero and 1023. Zero means that no object is perceived, while 1023 means that an object is very close to the sensor (almost touching the sensor). Intermediate values may give an approximate idea of the distance between the sensor and object. Two motors turn the right and left wheels of the robot, respectively. The range of v_R and v_L is between -10 to +10, where v_R is the speed of the right wheel and v_L is that of the left wheel. Negative values rotate the wheel backward, while positive values rotate the wheel forward.

3.2. Reward and Fitness in Wall Following Behaviors

GNP-TSRL judges the values of the sensors and determines the speed of the wheels depending on node function ID_{ip} and parameter a_{ip}, while the robot moves in the environment and gets rewards. A trial ends when the individual uses 1000 time steps, then the fitness is calculated. In this simulation, GNP-TSRL learns the wall following behavior, i.e., the robot must move along the wall as fast as and as straight as possible. The reward $r(t)$ at time step t and fitness are calculated by the following equations [15]

$$\mathrm{r(t)} = \frac{\mathrm{v_R(t) + v_L(t)}}{20} \times \left(1 - \sqrt{\frac{|\mathrm{v_R(t) - v_L(t)}|}{20}}\right) \times \mathrm{C}, \tag{7}$$

$$\mathrm{Fitness} = \textstyle\sum_{\mathrm{t=1}}^{1000} \mathrm{r(t)}/1000, \tag{8}$$

where, $v_R(t)$ and $v_L(t)$ are the speed of the right and left wheels at time step t, respectively. The range of $v_R(t)$ and $v_L(t)$ is between -10 to +10. If all the sensors have values less than 1000 and at least one of them is more than 100, then C is equal to 1, otherwise C is equal to 0.

3.3. Simulation Conditions

The node functions of the judgment nodes and processing nodes are shown in Table 1. Each judgment function, $J0, \ldots, J7$, judges the sensor value and determines the next node in the node transitions probabilistically [20]. Each processing node determines the speed of the left or right wheel. The simulation conditions of GNP-TSRL in Table 2, where these values are selected appropriately through the simulations. In this paper, Gaussian noises $(\mu = 0, \sigma = 50)$ are added to the sensor values in the training phase to improve the generalization ability of GNP-TSRL in noisy environments of the implementation phase [20].

In the training phase, 300 individuals are evolved, where at the end of each generation, 300 individuals are generated to form a new population for the next generation; 179 individuals are generated by mutation, 120 individuals are generated by crossover, and one individual is the elite. Each individual uses 61 nodes including 40 fuzzy judgment nodes (5 for each kind), 20 processing nodes (10 for each kind) and one start node. Each of the fuzzy judgment nodes and processing nodes of GNP with TSRL has 2 sub nodes, and each branch of the sub nodes has 2 branch connections which are determined by the evolution. The best individual in the last generation is selected for the implementation. Figure 6 shows the flowchart of the proposed method of GNP-TSRL. The performance of GNP-TSRL is studied in two aspects, that is, in the training phase and implementation phase. The successful trajectories of the robot in the training and implementation environments are shown in Figure 7.

Evolution phase. The evolution of GNP-TSRL starts from the initialization of individuals. Each individual has one start node and a fix number of judgment nodes and processing nodes. The function of node (ID_{ip}) is assigned by a unique number which is shown in the function library. The parameter of node (a_{ip}) is set at a randomly selected integer. When the node is a judgment node, its parameter is $a_{ip} = \{\beta_{ip}, \alpha_{ip}\}$, where α_{ip} is larger than β_{ip}; that is α_{ip} is set at between 0 and 1023, and β_{ip} is set at between 0 and α_{ip}, while when the node is a processing node, its parameter is set at between -10 and 10. The initial connection of the node by branch $b_{ip(qr)}$ is determined randomly. All Q values (Q_{SS} and Q_{BS}) are set at zero initially. The connections between nodes, node functions and parameters of the individuals are changed by crossover and mutation whose rates are P_c and P_m, respectively. The reader can refer to [20] for genetic operators in details.

Table 1. Node Functions Used in the Function Library

Symbol	ID	Content
$j0,\ldots J7$	0, ..., 7	judge the value of the sensor 1,2,..., 8
$P0$	0	determine the speed of the right wheel
$P1$	1	determine the speed of the left wheel

Table 2. Simulation Conditions

The number of individuals	300 (mutation: 179, crossover: 120, elite: 1)
The number of nodes	61 (20 processing nodes, 40 fuzzy judgment nodes, and 1 start node)
The number of sub nodes	2 for each fuzzy judgment and processing node
The number of branch connections	2 for each branch
Parameter of evolution	$Pc = 0.1$, $Pm = 0.01$, Tournament sizes = 7
Parameter of learning (Training phase)	$\gamma = 0.9$, $\in_A = 0.01, \alpha_A = 0.10$ (CaseA) $\in_B = 0.15, \alpha_B = 0.70$ (CaseB)
Parameter of learning (Implementation phase)	$\gamma = 0.9$, $\in_{step} = 0.01, \alpha_{step} = 0.10$,

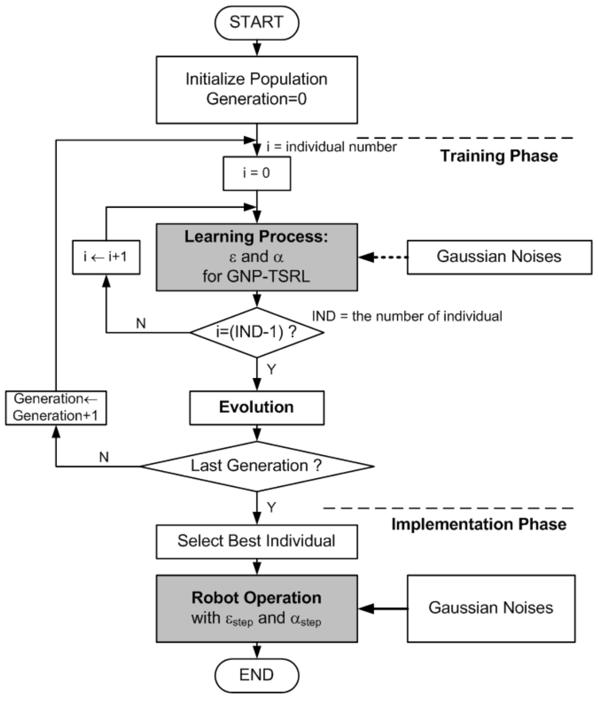
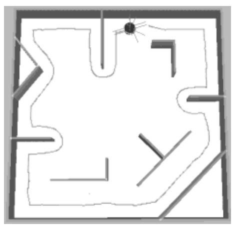
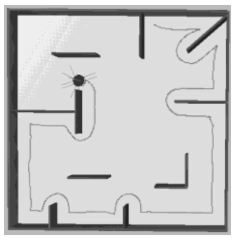


Figure 6. Flowchart of GNP-TSRL



(a)



(b)

Figure 7. Successful trajectories of the robot in the training and implementation environments (a) training environment, (b) implementation environment

Learning phase. The learning processes of GNP-TSRL is observed by using cases for high exploration and low explorations. These cases are used during learning phase and are implemented in the implementation phase. In the training phase, the individuals learn the environments using two cases A and B, where case A for low exploration and case B for high exploration. In case A, parameters use ϵ_A = 0.01 and $\alpha_A = 0.10$, while in case B, parameters use ϵ_B = 0.15 and $\alpha_B = 0.70$. In the implementation phase, the effects of ϵ_{step} and α_{step} are studied using learning parameters with values of $\in_{step} = 0.01$ and $\alpha_{step} = 0.10$ with its life time step, i.e., 3000 time-steps.

3.4. Training Results

Firstly, the adaptability of GNP-TSRL is studied in the training phase, where the parameters of ϵ and α are choose appropriately in order to learn the environments. The average fitness of GNP-TSRL trained $(TSRL)$ is compared when it is use constant ϵ and α. In the training phase, the values of ϵ and α are shown in Table 2, i.e., $TSRL(A)$ use $\epsilon_A = 0.01$ and $\alpha_A = 0.10$ and $TSRL(B)$ use $\epsilon_B = 0.15$ and $\alpha_B = 0.70$. The average fitness is shown in Figure 8, which is calculated over 10 best individuals of 10 independent training simulations.

The average fitness of $TSRL(A)$ converges faster and higher than that of $TSRL(B)$, because the actions with higher Q-values can be selected more frequently in $TSRL(A)$, while $TSRL(B)$ carries out random action selections more frequently than $TSRL(A)$. In the other words, $TSRL(A)$ and $TSRL(B)$ carry out higher and lower exploitation, respectively. In this case, when the random action selections are carried out with high probability, the actions cannot be reinforced well, then the Q-values are small, while when the exploitation of action selections is carried out with high probability, the good act ions are reinforced, but the alternative actions cannot be reinforced well.

3.5. Implementation Results

In the implementation phase, the performace of the proposed method is studied when the individuals are implemented with parameters of ϵ_{step} and α_{step} as shown in Table 3. The simulations are done 3000 times, that is, 10 best individuals from 10 independent runs in the training phase are implemented 300 times using 10 different start positions.

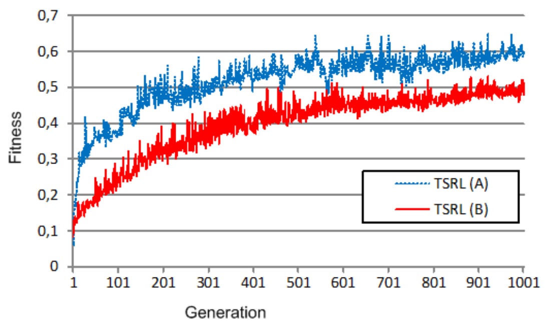


Figure 8. Average fitness in the training phase

Table 3. Average reward of TSRL(A), TSRL(B) in Testing phase

Individual	Average	Stdev	T-test one tail (p-value)
TSRL(A)	0.070	0.030	-
TSRL(B)	0.112	0.031	0.0065

The results shows that, the individuals trained by $TSRL(A)$ and $TSRL(B)$ are implemented using constant ϵ_{step} and α_{step}, i.e., $\epsilon_{step} = 0.01$ and $\alpha_{step} = 0.10$. When an inexperienced environment used in the implementation, the action selections are selected considering situations learned in training phase, while the Q-values of the current transition, are used to leceted actions due to the changes of the environments. Here, $TSRL(A)$ has the lower average reward as shown in Table 3, because it was trained with higher exploitation, and the Q-values of the alternative actions had small values. Thus, due to the changes of the environments, the actions of $TSRL(A)$ cannot be selected appropriately, while as $TSRL(B)$ was trained with higher exploration, although the performance of $TSRL(B)$ in the training phase is worse than $TSRL(A)$, the average reward of $TSRL(B)$ is higher than $TSRL(A)$ in the implementation phase, because the Q-values of the alternative actions had larger values. Thus, due to an inexperienced of environments, the actions of $TSRLco(B)$ can select be selected more appropriately. The proposed method, $TSRL(B)$ has the better result than $TSRL(A)$, which means the two-stage reinforcement learning can reinforce good actions and the alternative actions. Here, $TSRL(B)$ shows more efficient and effective compared to $TSRL(B)$.

4. Conclusion

The two stage reinforcement learning of Genetic Network Programming (GNP-TSRL) has been proposed to improve the performance of conventional GNP-RL. In the training phase, the average fitness of $TSRL(A)$ converges faster and higher than that of $TSRL(B)$, while $TSRL(B)$

has the better result than $TSRL(A)$ in testing phase, which means the two stage reinforcement learning can reinforce good actions and the alternative actions. It shows that the exploration of learning two stage RL (GNP-TSRL) can improve the performance of GNP-TSRL efficiently and effectively by providing alternative connections. In the future work, we will study the performance of the proposed method by studying the adaptability when severe conditions occur.

References

[1] Sutton RS, Barto AG. Reinforcement learning: An introduction. MIT press. 1998.
[2] Kaelbling LP, Littman ML, Moore AW. Reinforcement learning: A survey. *J Artif Intell Res* 1996; 4: 237–285.
[3] Smart WD, Kaelbling LP. *Effective reinforcement learning for mobile robots*. Robotics and Automation 2002 Proceedings, ICRA'02. IEEE International Conference. 2002: 3404–3410.
[4] Dong D, Chen C, Chu J, et al. Robust quantum-inspired reinforcement learning for robot navigation. *IEEEASME Trans Mechatron* 2012; 17(1): 86–97.
[5] Khriji L, Touati F, Benhmed K, et al. Mobile robot navigation based on Q-learning technique. *Int J Adv Robot Syst* 2011; 8(6): 4.
[6] Anual SN, Ibrahim MF, Ibrahim N, Hussain A, Mustafa MM, Huddin AB, Hashim FH. Ga-based optimisation of a lidar feedback autonomous mobile robot navigation system. *Bulletin of Electrical Engineering and Informatics*. 2018; 7(3): 433-441.
[7] Patle BK, Parhi DRK, Jagadeesh A, et al. Matrix-Binary Codes based Genetic Algorithm for path planning of mobile robot. *Comput Electr Eng* 2018; 67: 708–728.
[8] Santiago RMC, De Ocampo AL, Ubando AT, et al. *Path planning for mobile robots using genetic algorithm and probabilistic roadmap*. Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment and Management (HNICEM), 2017 IEEE 9th International Conference. 2017: 1–5.
[9] Koza JR. Genetic programming II, automatic discovery of reusable subprograms. Cambridge, MA: MIT Press. 1992.
[10] Contreras-Cruz MA, Ayala-Ramirez V, Hernandez-Belmonte UH. Mobile robot path planning using artificial bee colony and evolutionary programming. *Appl Soft Comput* 2015; 30: 319–328.
[11] Yang X, Cai M, Li J. *Path planning for unmanned aerial vehicles based on genetic programming*. Control and Decision Conference (CCDC), 2016 Chinese. 2016: 717–722.
[12] Hirasawa K, Okubo M, Katagiri H, et al. *Comparison between genetic network programming (GNP) and genetic programming (GP)*. Evolutionary Computation, 2001. Proceedings of the 2001 Congress. 2001: 1276–1282.
[13] Yao X. *Evolving artificial neural networks*. Proc IEEE. 1999; 87(9): 1423–1447.
[14] Ito T, Iba H, Kimura M. *Robustness of robot programs generated by genetic programming*. Proceedings of the 1st annual conference on genetic programming. 1996: 321–326.
[15] Mabu S, Hirasawa K, Hu J. A graph-based evolutionary algorithm: Genetic network programming (GNP) and its extension using reinforcement learning. *Evol Comput* 2007; 15(3): 369–398.
[16] Yan XY, Wu QW, Liu H. An improved robot path planning algorithm. *TELKOMNIKA Telecommunication Computing Electronics and Control*. 2012; 10(4): 629-36.
[17] Liu F, Liang S, Xian DX. Optimal Path Planning for Mobile Robot Using Tailored Genetic Algorithm. *TELKOMNIKA Indonesian Journal of Electrical Engineering*. 2014; 12(1): 1-9.
[18] Mabu S, Hatakeyama H, Thu MT, et al. Genetic network programming with reinforcement learning and its application to making mobile robot behavior. *IEEJ Trans Electron Inf Syst*. 2006; 126: 1009–1015.
[19] Watabe H, Kawaoka T. *Automatic generation of behaviors for mobile robot by GA with automatically generated action rule-base*. Industrial Electronics Society, 2000. IECON 2000. 26th Annual Confjerence of the IEEE. 2000: 1668–1674.
[20] Sendari S. Fuzzy Genetic Network Programming with Noises for Mobile Robot Navigation. *J Adv Comput Intell Intell Inform*. 2011; 15(7): 767–776.
[21] Zaragoza JH, Morales EF. *A two-stage relational reinforcement learning with continuous actions for real service robots*. Mexican International Conference on Artificial Intelligence. 2009: 337–348.
[22] Sendari S, Mabu S, Hirasawa K. *Two-Stage Reinforcement Learning Based on Genetic Network Programming for Mobile Robot*. SICE Annual Conference (SICE), 2012 Proceedings. 2012: 95-100.
[23] Ishii S, Yoshida W, Yoshimoto J. Control of exploitation–exploration meta-parameter in reinforcement learning. *Neural Netw*. 2002; 15(4-6): 665–687.
[24] Tokic M. *Adaptive ε-greedy exploration in reinforcement learning based on value differences*. Annual Conference on Artificial Intelligence. 2010: 203–210.
[25] Osugi T, Kim D, Scott S. *Balancing exploration and exploitation: A new algorithm for active machine learning*. Data Mining, Fifth IEEE International Conference. 2005: 8.
[26] Murakoshi K, Mizuno J. *A parameter control method inspired from neuromodulators in reinforcement learning*. Computational Intelligence in Robotics and Automation, 2003. Proceedings. 2003 IEEE International Symposium. 2003: 7–12.