Paper

Online Learning of Genetic Network Programming and its Application to Prisoner's Dilemma Game

Shingo Mabu* Student Member
Kotaro Hirasawa** Member
Jinglu Hu* Member
Junichi Murata* Member

A new evolutionary model with the network structure named Genetic Network Programming (GNP) has been proposed recently. GNP, that is, an expansion of GA and GP, represents solutions as a network structure and evolves it by using "offline learning (selection, mutation, crossover)". GNP can memorize the past action sequences in the network flow, so it can deal with Partially Observable Markov Decision Process (POMDP) well. In this paper, in order to improve the ability of GNP, Q learning (an off-policy TD control algorithm) that is one of the famous online methods is introduced for online learning of GNP. Q learning is suitable for GNP because (1) in reinforcement learning, the rewards an agent will get in the future can be estimated, (2) TD control doesn't need much memory and can learn quickly, and (3) off-policy is suitable in order to search for an optimal solution independently of the policy. Finally, in the simulations, online learning of GNP is applied to a player for "Prisoner's dilemma game" and its ability for online adaptation is confirmed.

Keywords: Genetic Algorithm, Genetic Programming, Network Structure, Online learning, Q learning, Prisoner's dilemma game

1. Introduction

Evolutionary processes of organisms are very complicated and sophisticated. They are based on the mechanisms such as learning, selection and evolution, so that they can harmonize well with environments.

"Genetic Algorithm (GA) [1]" and "Genetic Programming (GP) [2][3]" are typical methods that are based on the evolutionary processes of organisms. Because the conventional control theory should obey the rules predefined in advance and it cannot be adapted to the dynamical environments rapidly, the evolutionary computation overcoming the above problems attracts the attention.

GA and GP mainly applied to optimization problems represent solutions as a string and a tree structure, respectively and evolve them. GP was devised later in order to expand the representation ability of GA and to solve more complex problems. But, GP might be difficult to search for a solution because of the bloat of its tree structure, which expands the depth of the tree unnecessarily although it is sometimes useful for expanding a search space to find a solution.

Recently, a new evolutionary computation method named "Genetic Network Programming (GNP)" was proposed [4]~[6]. GNP, that is, an expansion of GA and GP represents solutions as a network structure. Since GNP can memorize the past action sequences in the network flow and can deal with Partially Observable Markov Decision Process (POMDP) well, GNP was applied to complicated agent systems involving uncertainty such as decision-making problems. But, conventional GNP are based on "offline learning", that is, after GNP is carried out to some extent, it is evaluated and evolved according to rewards given by an environment. However, the adaptation to dynamical environments might be difficult because in offline learning if the changes of an environment occurred, offline learning must do many trials to evaluate GNP and evolve it again and again, therefore, it cannot keep up with the changes of environments quickly.

In this paper, Q learning [7] which is one of the famous online learning methods is introduced for the online learning of GNP. GNP changes its structure considering the rewards given one after another, so that it can change its solution (behavior sequences) immediately after the environmental changes that cause a bad result.

In this paper, Prisoner's dilemma game is used for simulations and the performance of online learning is studied. Prisoner's dilemma game needs two players and they compete with each other to get high scores. First, GNP having a game strategy competed with the famous strategies of Prisoner's dilemma game and showed the good performances. Then, two GNPs competed with each other and showed the online adjustment ability on their strategies considering the opponent's strategy in

* Kyushu University
Department of Electrical and Electronic Systems Engineering, Kyushu University, 6-10-1, Hakozaki, Higashi-ku, Fukuoka, 812-8581, Japan
** Waseda University
Graduate School of Information, Production, and Systems, Waseda University, 2-2, Hibikino, Wakamatsu-ku, Kitakyushu, Fukuoka, 808-0135, Japan

order to get higher scores.

This paper is organized as follows. In the next section, the details of Genetic Network Programming are described. Section 3 explains Prisoner's dilemma game and shows the results of the simulations. Section 4 is devoted to conclusions.

2. Genetic Network Programming (GNP)

In this section, Genetic Network programming is reviewed briefly. GNP is an expansion of GP in terms of gene structures. The original motivation of developing GNP is based on the more general representation ability of graphs than that of trees.

2.1 Basic structure of GNP First, GP is explained in order to compare it with GNP. Fig.1 shows a basic structure of GP. GP can be used as a decision making tree when non-terminal nodes are *if-then* type functions and all terminal nodes are some concrete action functions. A tree is executed from the root node to a certain terminal node in each iteration, therefore, it might fall into the deadlocks because the behaviors of agents made by GP are determined only by the environments at the current time. Furthermore, GP tree might cause the severe bloat that makes search for solutions difficult because of the unnecessary expansion of depth.

Next, the characteristics and abilities of GNP are explained using Fig.2 which shows the basic structure of GNP. Now, it is supposed that agent behavior sequences are created by GNP.

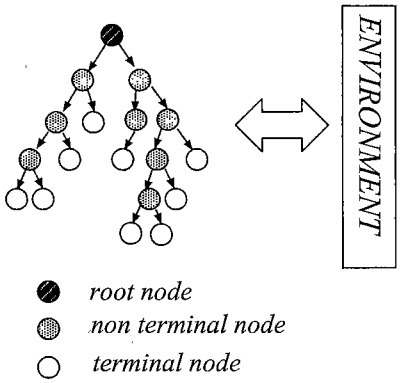


Fig. 1. Basic structure of GP.

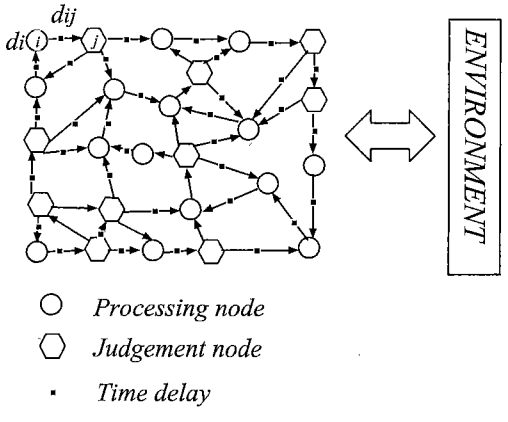


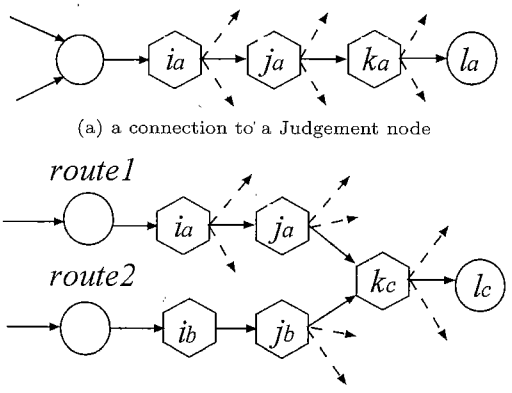
Fig. 2. Basic structure of GNP.

• **Function Sets** GNP has a number of Judgement nodes and Processing nodes. Judgement nodes are *if-then* type decision functions or conditional branch decision functions. They return judgement results for assigned inputs and determine the next node GNP should take. Processing node determines an action/processing an agent should take. Contrary to judgement nodes, processing nodes have no conditional branch. The GNP we used never causes bloat because of the predefined number of nodes, although GNP can evolve phenotypes/genetypes of variable length. Because the network structure of GNP can re-use the nodes unlike GA and GP, if GNP needs to use certain Judgement/Processing nodes many times to achieve a goal, GNP increases the connections to those nodes in the learning process and can re-use them again and again. Therefore, even if the number of nodes is predefined and smaller than GP programs, GNP can perform well by making effective network connections based on re-using nodes. So, we don't have to prepare the excessive number of nodes. As a result, we can easily determine the number of nodes experimentally.

• **Memory function of network** Although GNP is booted up from the start node that is predefined in advance arbitrarily, there are no terminal nodes. After the start node, the current node is determined according to the connections of the nodes and judging results of judgement nodes. In this way, GNP system is carried out according to the network flow without any terminal, that is, the determination of the current node is influenced by the node transitions of the past. So the network structure itself implicitly has a memory function of the past actions of an agent.

• **Connection constraint** As described before, Processing nodes and Judgement nodes are connected by directed links with each other like networks. But, there are some constraints on connection rules. Firstly, plural connections to a Processing node are available. On the other hand, there should be only one connection to a Judgement node. This constraint is introduced in order to judge environment correctly (Fig.3).

In the case of Fig.3(a), the judgement result of node k_a (next node: l_a) is determined by the results of nodes i_a, j_a and k_a (only one judgement route/process). But



(a) a connection to a Judgement node

(b) plural connections to a Judgement node

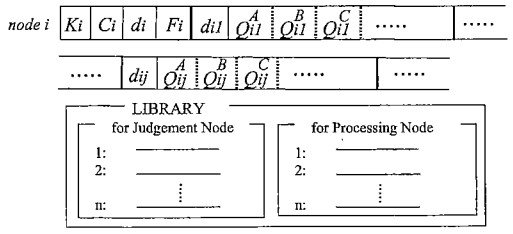
Fig. 3. Connection rules.

in the case of (b), node k_c cannot know which routes (route 1 or route 2) affects the judgement result of k_c, i.e. if we allow plural connections from some nodes to a Judgement node, the judgement on which node to take next is not determined uniquely. So, the correct judgement cannot be done.

• **Time Delays** GNP can have time delays. d_i is the time delay GNP spends on judgement or processing of node i, and d_{ij} is the one GNP spends on transitions from node i to j. In the real world problems, when agents judge environments, prepare for actions and take actions, agents need to spend time. For example, when a man is walking and there is a puddle before him, he will avoid it. At that time, it takes some time to judge the puddle (d_i for node judgement), to put judgement into action (d_{ij} for transition from judgement for node processing) and to avoid the puddle (d_i for node processing). However, d_i and d_{ij} are not used in this paper because the purpose of the simulations using Prisoner's dilemma game is to make and change strategies adapting to the opponent behavior and the time spent for judgement or processing does not need to be considered. Time delay is necessary in the case of practical applications. When various judgements and processes are done in real applications, GNP should change its structure considering the time it spends. Time delay is listed in each node gene which will be described later because it is the unique attribute of the node.

2.2 Genotype expression of GNP node The whole structure of GNP is determined by the combination of the following node genes. A genetic code of node i ($1 \leq i \leq n^{\dagger}$) is represented as Fig.4.

K_i represents the node type, K_i=0 means Processing node, K_i=1 means Judgement node. C_i shows the code number GNP judges or processes and it is represented as a unique number shown in the LIBRARY.



where, K_i : Judgement/Processing classification
0 : Processing node
1 : Judgement node
C_i : code number of Judgement/Processing
d_i : time delay spent for judgement/processing
F_i : Flag : condition of connections to node i
0: free 1:busy
d_{ij} : time delay spent for the transition from node i to j
$Q^A_{ij}, Q^B_{ij}, Q^C_{ij}, \cdots$: transition Q values from node i to j

Fig. 4. Genotype expression of node i.

† Each node has a unique number from 1 to n, respectively, when the number of nodes is n.

d_i is the time delay spent for judgement or processing. F_i is Flag of node i. If node i can accept the connection from the other nodes, Flag is set at "0", otherwise "1". Because Processing nodes can accept unlimited connections, Flags of Processing nodes are always "0". On the other hand, Judgement node accepts only one connection from the other nodes, so when a connection has been built to a Judgement node, Flag of the node becomes "1", otherwise "0". The concrete usage of Flags are described in Section 2.4.4. d_{ij} means time delay spent for the transition from node i to j. $Q^A_{ij}, Q^B_{ij}, Q^C_{ij} \ldots$ means transition Q values from node i to j. Judgement node determines which Q value is used for selecting the transition according to judging results. For example, if the result of judgement is "A", GNP uses Q^A_{ij} to select a transition and connect to a next node, and if the result is "B", it uses Q^B_{ij}. Therefore, the number of $Q^A_{ij}, Q^B_{ij}, Q^C_{ij}, \ldots$ is the same as the number of the results at judgement nodes. On the other hand, Processing node has only Q^A_{ij} because there are no conditional states when processing.

2.3 Coding Example Fig. 5 shows a coding example of GNP. *Node* 1 and 2 are the processing nodes and *Node* 3 and 4 are the judgement nodes. The code of each node (C_i) is shown in the LIBRARY. GNP refers all Q^A_{ij} of node i ($1 \leq i,j \leq 4$), and the node j having the maximum Q^A_{ij}-value is connected from node i. *Node* 3 and 4 have two conditional branches, respectively, so GNP also refers all Q^B_{ij} of *Node* 3 and 4, and connections are made in the same way as the above. Since *Node* 3 has accepted a connection, Flag of *Node* 3 is 1. The time delays are not used in this paper, so all d_i and d_{ij} are set at 0.

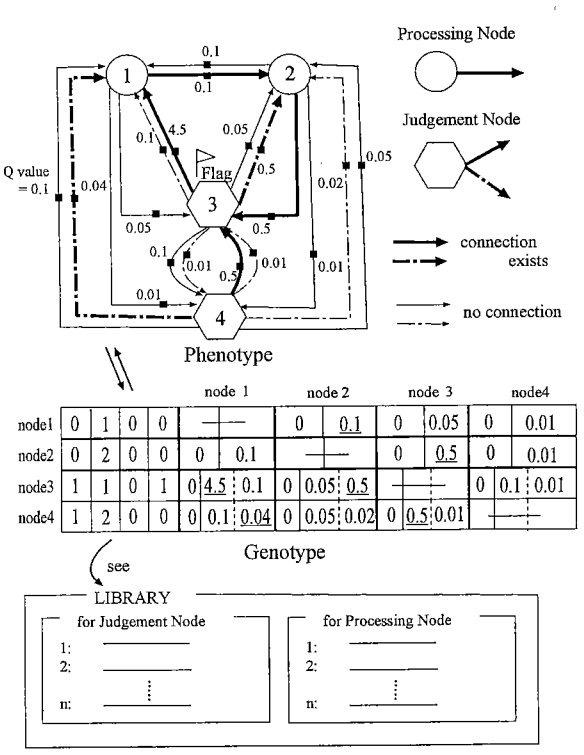


Fig. 5. Coding example of GNP.

2.4 Online learning of GNP In this section, we will explain how to calculate Q_{ij}-values. In the offline learning method that GA, GP and conventional GNP [4]~[6] uses, many individuals calculate their fitness values, and then genetic operations (selection, mutation, crossover) are carried out to generate new populations. This process is called One generation. But, offline learning cannot change its strategy until the current generation ends, while online learning of GNP is based on Q learning and learns its decision making rules every judgement/processing. Therefore, GNP can prevent the useless trials under environmental changes.

2.4.1 Q learning (an off-policy TD control algorithm) Q learning calculates Q values which are functions of state s and action a. Q values mean the sum of the rewards an agent gets in the future, and the update processes of them are implemented as follows. When an agent selects an action a_t in state s_t at time t, the reward r_t is given and the state is changed from s_t to s_{t+1}. As a result, $Q(s_t, a_t)$ is updated as the following equation.

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha\left[r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right] \cdots\cdots (1)$$

If the step size parameter α decreases according to a certain schedule, Q values converge on an optimal values after enough trials. The action selection which has the maximum Q value becomes the optimal policy. $\gamma(0 \leq \gamma \leq 1)$ is a discount rate which shows how long an agent considers the future rewards.

If an agent continues to select the actions having the maximum Q value when the learning is not enough, the action selection rules are not improved even though there are still better ones. One of the method for overcoming the above problem is "ε-greedy". This method makes the agent select the random action by the probability of ε, or select the action having the maximum Q value by the probability of 1-ε. ε-greedy is a general method for keeping a balance between exploitation of experience and exploration. In this paper, ε-greedy is adopted for the action selection.

2.4.2 Relation between GNP and Q learning In Q learning, state s is determined by the information an agent can get, and action a means the actual action it takes. On the other hand, in GNP learning, a state means the state after judging or processing and an action means node transition. Fig. 6 shows an example of node transition. The symbols "•" and the arrows (transitions) "⟶" represent the states and the actions, respectively. The state s and action a are different from the actual judgement and processing executed by judgement node and processing node, respectively.

2.4.3 The reason for applying Q learning (1) Once GNP is booted up from the start node, the current node is transferred one after another without any terminal node. Therefore, the framework of reinforcement learning that uses the sum of the discounted future rewards is suitable for online learning of GNP.

(2) TD control needs only a maximum Q value in the next state, therefore, much memory is not needed and Q value is updated easily.

(3) Because GNP should search for an optimal solution independently of the policy (ε-greedy), off-policy is adopted.

2.4.4 Node transition and Learning The following is the outline of online learning. The transition begins from the start node that can be selected from all the nodes arbitrarily and GNP executes the content of the start node, then GNP selects a transition having a maximum Q value in all transitions and connects the start node and the next one. However, by the probability of ε, GNP selects the transition randomly. After GNP executes the content of the next node and a reward is given, Q value is updated according to eq.(1), so GNP connects the previous node with the one having the largest Q value from the previous one. But, if the transition having a maximum Q value corresponds to the judgement node whose Flag is "1", then GNP selects the transition having the second largest Q value. Similarly, if the transition of the second largest Q value corresponds to the node with Flag "1", GNP selects the third one. This process is repeated until the Flag "0" node is found. After that, the transitions continue following the network flow.

The concrete processes are described using Fig.6.

There are n nodes including judgement nodes and processing nodes, and each node has the number from 1 to n, respectively. At time t, it is supposed that the current node is node $i(1 \leq i \leq n)$. Because node i is the judgement node, it judges the current environment. In

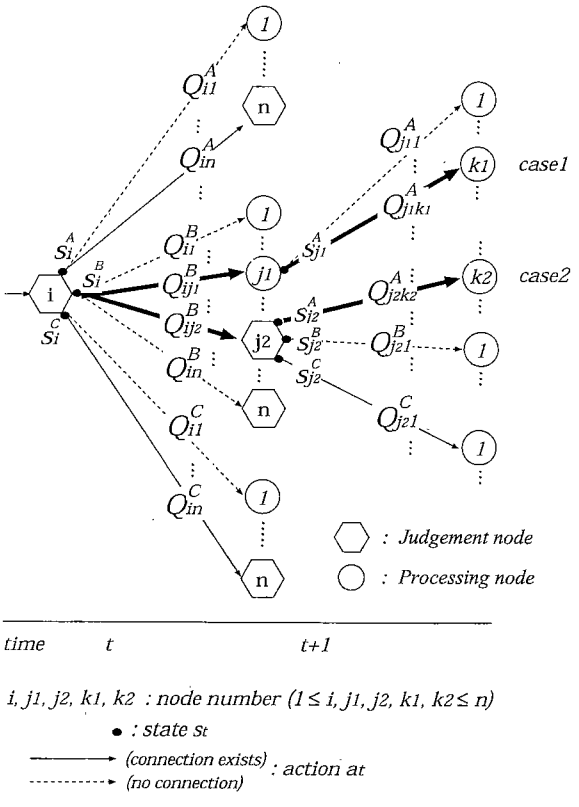


Fig. 6. An example of node transition.

this example, the result of the judgement could be A, B or C and now B is supposed at time t. Then, the state s_t becomes S_i^B, and the transition is selected according to its transition Q_{ij}^B values. But, by the probability of ε, GNP randomly selects the transition which corresponds to a Flag "0" node.

• case *1 : next node is a processing node*

If the transition which has $Q_{ij_1}^B$ is selected, the next node is processing node j_1. GNP processes node j_1 and gets the reward r_t. Then, the time is changed to $t+1$ and state s_{t+1} becomes $s_{j_1}^A$ decisively because the processing nodes have no conditional state unlike judgement nodes. If $Q_{j_1k_1}^A$ is the maximum value among $Q_{j_11}^A, \ldots, Q_{j_1n}^A$, this value is used to update $Q_{ij_1}^B$ as eq.(2). After that, node i is connected to a Flag "0" node as described before. Then GNP selects the transition to k_1 as an action a_{t+1}, or selects the random transition to Flag "0" node by the probability of ε.

$$Q_{ij_1}^B = Q_{ij_1}^B + \alpha\left[r_t + \gamma Q_{j_1k_1}^A - Q_{ij_1}^B\right] \cdots\cdots\cdots (2)$$

• case *2 : next node is a judgement node*

If the transition which has $Q_{ij_2}^B$ is selected, the next node is judgement node j_2 and GNP judges node j_2. However, since the actual action is not done at node j_2, the reward r_t is not given. The time is changed to $t+1$ and the state s_{t+1} could be $S_{j_2}^A$, $S_{j_2}^B$ or $S_{j_2}^C$ according to the judgement at node j_2. It is supposed that the result of the judgement is A in this example, then s_{t+1} becomes $s_{j_2}^A$. If $Q_{j_2k_2}^A$ is the maximum value among $Q_{j_21}^A, \ldots, Q_{j_2n}^A$, it is used for updating $Q_{ij_2}^B$. The update of $Q_{ij_2}^B$ is implemented as eq.(3). Similarly, the node i is connected to a Flag "0" node. Then GNP selects the transition to k_2 as an action a_{t+1} or selects the random transition by the probability of ε.

$$Q_{ij_2}^B = Q_{ij_2}^B + \alpha\left[Q_{j_2k_2}^A - Q_{ij_2}^B\right] \cdots\cdots\cdots\cdots (3)$$

Usually Q-learning assigns Q-values to all possible actions of any states, so when an agent gets the reward, Q-learning updates the Q-value considering the state and the action. There are some differences between GNP and general Q-learning, because GNP has Judgement Nodes and Processing Nodes. Here, one series of Judgement nodes and the following Processing node are called " One Iteration " as is stated in Fig. 7. After GNP understands agents' situation by one series of judgement nodes, the next action is determined by the following Processing node. So, the judgement for understanding the situation is separated from the processing. As a result both Judgement and Processing nodes have their own Q-values. We should evaluate the judgements and processing in One Iteration equally because all Judgement nodes and Processing node in One Iteration should contribute to the reward equally. In other words, if we use the discount rate γ in Eq. (3), the more the judgement nodes are used, the less the previous nodes obtain discounted rewards. As a result, Judgement nodes would not be used in the network flow. This has a bad influence on the system because enough judgements would not be done in GNP. Therefore, the discount rate is not used in Case 2.

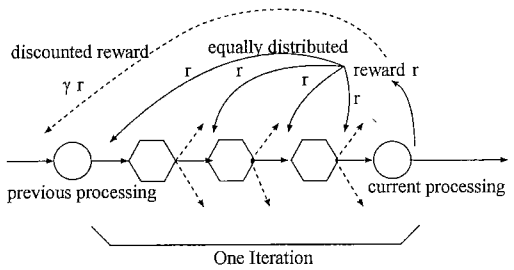


Fig. 7. An example of the flow of One Iteration

2.5 Comparison between GNP and other graph based evolutionary methods There have been developed some graph based evolutional methods such as PADO (Parallel Algorithm Discovery and Orchestration) [8] and EP (Evolutionary Programming) [9]. PADO has both start node and end node in the network and it mainly represents a static program. EP is used for the automatic synthesis of Finite State Machines and in all states, state transitions for all inputs have to be determined, therefore, the structure of EP becomes complicated if the number of states increases. On the other hand, GNP uses only the necessary information for judging the environments, i.e. it can deal with POMDP. Therefore, GNP could be compact even if the system to solve is large enough.

3. Simulations

In this section, GNP is used as a player of "Prisoner's dilemma game". The aim of this simulation is to confirm the effectiveness of the proposed online learning of GNP.

3.1 Prisoner's dilemma game

• story : The police don't have enough evidence to indict two suspects for complicity in a crime. The police approach each suspect separately. "If either of you remains silent, the sentence for your crime becomes a two-year prison in spite of the insufficient evidence. If you confess and your pal remains silent, you are released and the crime of your pal becomes a five-year prison. But, either of you confess, your sentence becomes a four-year prison."

From Table. 1, Profits can be 0, -2, -4 and -5, respectively. In order to make them positive, 5 is added to each profit, then the profit matrix shown in Fig.8 is obtained. Silence is called "Cooperate(C)" because it shortens its prison term. Confession is called "Defect(D)" because it is done in order for only one suspect to be released. If the profits described by the symbols R, S, T and P in the frame follow inequality (4), the dilemma arises. This

Table 1. Relation between Sentence and Confession/Silence

Suspect1	Suspect2	Sentence to suspect1
Confession	Silence	released
Silence	Silence	2-year prison
Confession	Confession	4-year prison
Silence	Confession	5-year prison

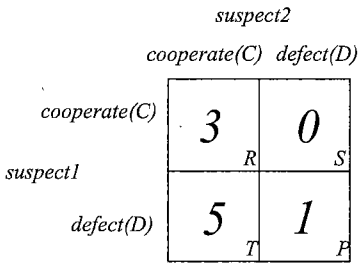


Fig. 8. Profit matrix (profit for suspect1).

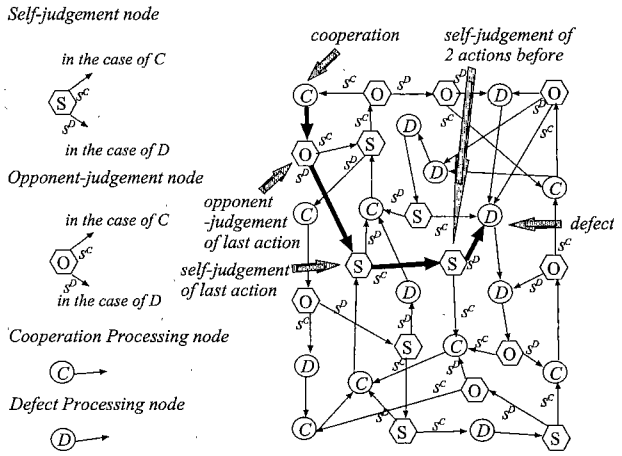


Fig. 9. An example of GNP structure.

matrix is the famous one in many research on Prisoner's dilemma game. Therefore, the results in this paper are calculated using Fig.8. In this simulation, players repeat selecting Cooperation or Defect.

$$\begin{aligned} &T > R > P > S \\ &2R > S + T \end{aligned} \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots (4)$$

If the players have no information about their opponents and they select the action only once, Defect is to be taken. In either case where the opponent takes Cooperation or Defect, Defect gets a higher profit than Cooperation. However, as the competition is repeated, and the characteristic of the opponent comes out, GNP can develop its strategy considering the past strategies of the opponent and itself.

3.2 GNP for Prisoner's dilemma game The player of Prisoner's dilemma game is regarded as an agent using GNP. The nodes of GNP are

- Self-judgement node (judge the action taken by itself)
- Opponent-judgement node (judge the action taken by an opponent)
- Cooperation processing node (C) (take Cooperation)
- Defect processing node (D) (take Defect)

Fig.9 shows an example of GNP structure for Prisoner's dilemma game. The thick arrows show an example of node transitions.

Each judgement node has two kinds of Q value, Q^C and Q^D. Q^C is used when the past action of itself or the opponent is "cooperation" (judgement result is C and the state becomes s^C). Q^D is used when the past action of them is "defect" (judgement result is D and the state becomes s^D). On the other hand, each processing node has one kind of Q value, which differs from a judgement node. After processing, if the next node is a self-judgement node, GNP judges the last action of itself, if an opponent-judgement node, it judges the one of the opponent. If the same kind of judgement node is executed again, GNP carries out the judgement of the action taken two steps before. In case the same kind of judgement nodes continues, GNP judges previous action one after another. After processing, if the next node is a processing one, an agent competes using the corresponding processing against the opponent. The distinguished point of GNP is that we can determine automatically how much the past actions of its own and opponent should be used. That is, GNP can learn how many judgements are necessary to determine the next effective action.

3.3 Simulation results In this simulation, firstly GNP competes against Tit for Tat and Pavlov strategy that are the fixed strategies because they are predefined in advance and not changed. The purpose of these simulations is to show that GNP can learn the characteristics of the opponents and can change its strategies to get high scores. Next, two GNPs compete against each other. This situation can be regarded as a game in a dynamic environment because GNPs can change their strategies each other. Therefore, GNPs should adapt to the changes of the strategy of the opponents.

GNP competes under the following conditions.

the number of nodes(N) : 40
node number : Cooperation (1-5)
: Defect (6-10)
: Self-Judgement (11-25)
: Opponent-Judgement (26-40)
discount rate(γ) : 0.99
step size parameter(α) : 0.8
ε : 0.05
reward(r) : obtained by Fig.8
Q values : zero in the initial state

The competition is carried out for the predifined iterations after Q values are initialized (all Q values are zero at first). This process is called a trial.

3.3.1 Competition between GNP and Tit for Tat

Tit for Tat :

- take Cooperation at the first iteration
- take the last action taken by the opponent

This strategy never loses a game by a wide margin and gets almost the same average scores as the opponent. Therefore, Tit for Tat is an admitted strong strategy. For example, when many Prisoner's dilemma game strategies compete against each other, the strategies might decrease their scores against others, but Tit

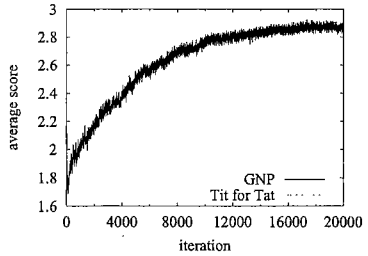
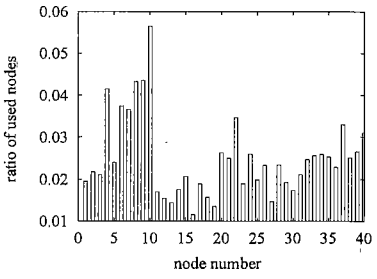
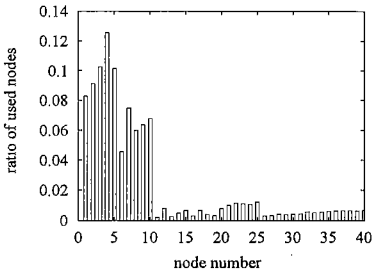


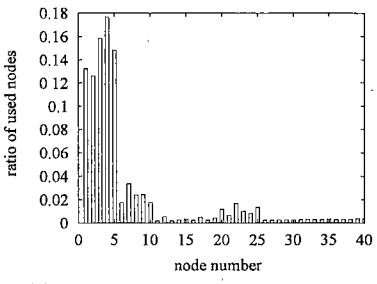
Fig. 10. Average score of competition between GNP and Tit for Tat.



(a) from 1 to 500 iterations



(b) from 501 to 10000 iterations



(c) from 10001 to 20000 iterations

Fig. 11. Ratio of used nodes for specific intervals in competition between GNP and Tit for Tat.

for Tat ends its competition with the same average as the opponent. Therefore, Tit for Tat becomes the strongest strategy as a whole. In this simulation, the competition between GNP and Tit for Tat is done. The moving averages over 10 iterations are calculated and Fig.10 shows the averages of these moving averages over 100 trials. Fig.11 shows how often GNP used each node (ratio of used nodes) for specific intervals and it is also the averages over 100 trials. Because Tit for Tat never loses by a wide margin, the average scores of GNP and Tit for Tat are almost the same and the lines of them are overlapped. As the competition is iterated, GNP gradually gets the high scores because GNP used more Cooperation Processing nodes than Defect ones as shown in Fig.11. If GNP takes Defect, Tit for Tat takes it in the next step and the score is diminished. If GNP continues to take cooperation, Tit for Tat also does so, therefore, GNP learned the cooperative strategy in order to get high scores. GNP also reduces the use of Judgement node because GNP can understand the strategy of Tit for Tat and it doesn't need many Judgements.

Table 2. Pavlov strategy.

Pavlov	Opponent	Next action of Pavlov
Cooperation	Cooperation	Cooperation
Cooperation	Defect	Defect
Defect	Cooperation	Defect
Defect	Defect	Cooperation

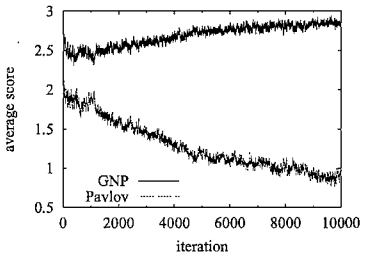
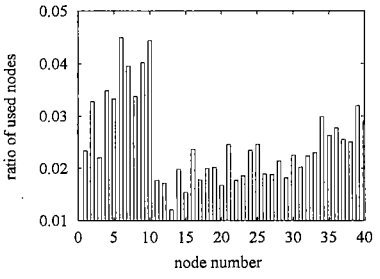
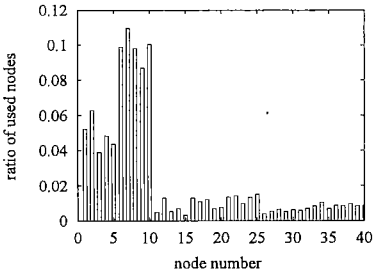


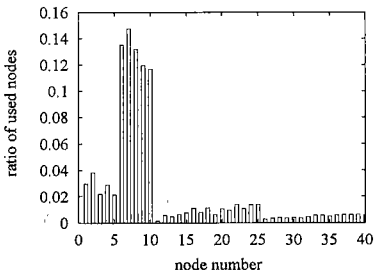
Fig. 12. Average score of competition between GNP and Pavlov strategy.



(a) from 1 to 500 iterations



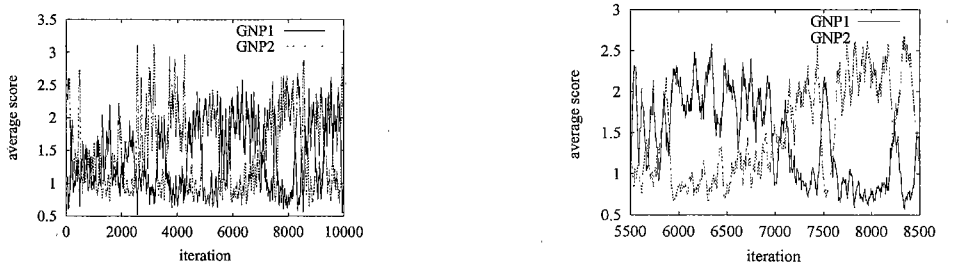
(b) from 501 to 3000 iterations



(c) from 3001 to 10000 iterations

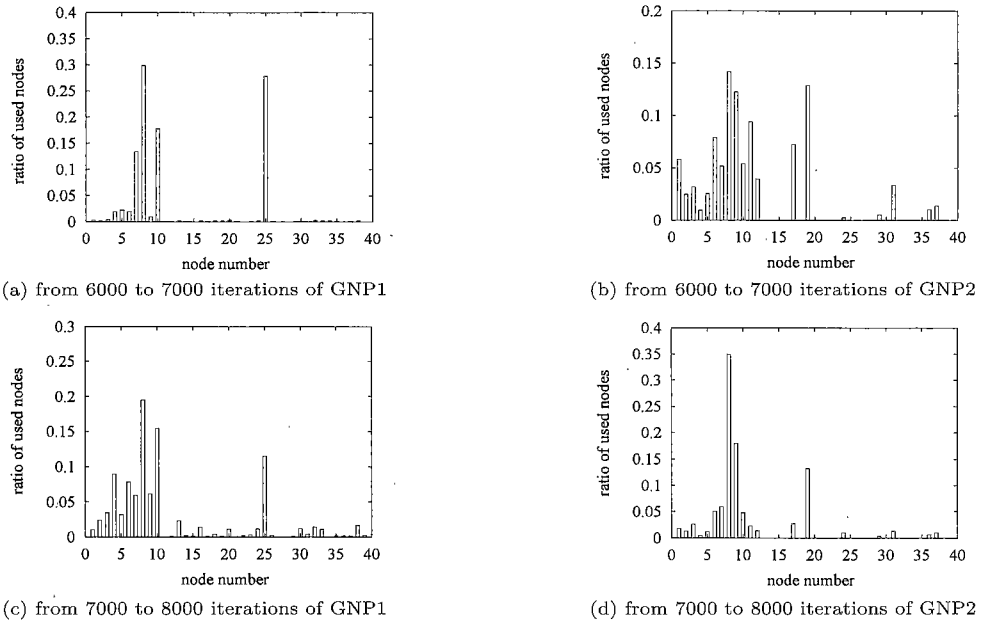
Fig. 13. Ratio of used nodes for specific intervals in competition between GNP and Pavlov strategy.

3.3.2 Competition between GNP and Pavlov strategy Table. 2 shows Pavlov strategy. Pavlov strategy decides the next action according to the com-



(a) from 1 to 10000 iterations (b) from 5500 to 8500 iterations

Fig. 14. Average score of competition between GNPs



(a) from 6000 to 7000 iterations of GNP1 (b) from 6000 to 7000 iterations of GNP2

(c) from 7000 to 8000 iterations of GNP1 (d) from 7000 to 8000 iterations of GNP2

Fig. 15. Ratio of used nodes for specific intervals in competition between GNPs.

bination of the last actions of both strategies.

Fig.12 and 13 show the result calculated by the same procedure as simulation 1. GNP could win by a wide margin because GNP realized that it could defeat Pavlov strategy by taking Defect strategy as shown in Fig.13. If both GNP and Pavlov strategy take Defect, Pavlov strategy takes Cooperation in the next step. Therefore, GNP learnd that it should take Defect next in order to get 5 point.

3.3.3 Competition between GNPs Two GNPs which have the same parameters compete with each other in this simulation. Fig.14 (a) shows a typical trial in order to study the progress of scores in detail. The average scores mean the moving averages over 10 iterations. The characteristic of the result is as follows. While GNP1 increases its scores, GNP2 decreases them and while GNP1 decreases its scores, GNP2 increases them. Fig.14 (b) shows the average scores from 5500 to 8500 iterations and Fig.15 shows how often GNPs used each node from 6000 to 7000 iterations and from 7000 to 8000 iterations, respectively. Unlike the previous simulations, Fig.15 does not show the averages over many trials, but one trial. When GNP1 can win by taking Defect much more than Cooperation [Fig.15 (a)], GNP2 takes Cooperation to some extent [Fig.15 (b)]. Then, GNP2 intends to take much more Defect [Fig.15(d)], so, the scores of GNP1 decrease around 7000 iterations. Therefore, GNP1 intends to take more Cooperation in order to avoid continuing to take Defect each other [Fig.15 (c)]. However, GNP2 becomes to win thanks to the strategy change of GNP1. After that, GNP1 intends to take Defect in turn and thus GNP2 intends to take Cooperation in order to avoid taking Defect each other. This process is repeated. From the viewpoint of online learning which always aims to get higher rewards than ever, this result is natural.

4. Conclusions

In this paper, in order to adapt to dynamic environments quickly, we proposed online learning of GNP and applied to Prisoner's dilemma game to confirm its learning ability. GNP can make its strategies according to Q values learned by Q learning. In the competition with fixed strategy (Tit for tat and Pavlov strategy), GNP increased its scores. In the competition between GNPs where strategies are dynamic, both GNPs can keep up with the changes of their strategies.

Henceforth, in a future, we would integrate online learning and offline learning so that the performance will be improved much more and GNP can model the

learning mechanisms of organisms more realistically.

(Manuscript received March 7, 2002, revised September 24, 2002)

References

(1) John H. Holland: "Adaptation in Natural and Artificial Systems", University of Michigan Press, 1975, MIT Press, (1992)

(2) John R., Koza: "Genetic Programming", on the programming of computers by means of natural selection, Cambridge, Mass., MIT Press, (1992)

(3) John R., Koza: "Genetic Programming II: Automatic Discovery of Reusable Programs, Cambridge", Mass., MIT Press, (1994)

(4) H.Katagiri, K. Hirasawa, and J.Hu: "Genetic Network Programming - Application to Intelligent Agents -", *in Proc. of IEEE International Conference on Systems*, Man and Cybernetics, pp.3829-3834 (2000)

(5) K.Hirasawa, M.Okubo, H.Katagiri, J.Hu, and J.Murata: "Comparison between Genetic Network Programming (GNP) and Genetic Programming (GP)", *in Proc. of Congress on Evolutionary Computation*, pp.1276-1282 (2001)

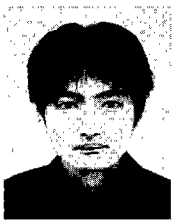
(6) H.Katagiri, K.Hirasawa, J.Hu, and J.Murata: "Network structure Oriented Evolutionary Model - Genetic Network Programming - and Its comparison with Genetic Programming", in 2001 Genetic and Evolutionary Computation Conference Late Breaking Papers, Erik D.Goodman, Ed., San Francisco, California, USA, 9-11 July 2001, pp.219-226 (2001)

(7) Richard S. Sutton and Andrew G. Barto: Reinforcement Learning - An Introduction, MIT Press Cambridge, Massachusetts, London, England, (1998)

(8) A. Teller and M.Veloso: PADO: Learning Tree-structured Algorithm for Orchestration into an Object Recognition System, Carnegie Mellon University Technical Report Library, (1995)

(9) L.J.Fogel, A.J.Owens, and M.J.Walsh: Artificial Intelligence through Simulated Evolution, Wiley, (1966)

Shingo Mabu (Student Member) He received the B.S. degree in Electrical Engineering from Kyushu University, Japan in 2001. Since April, 2001, he has been a master course student in the Graduate School of Information Science and Electrical Engineering, Kyushu University. He is a member of the Institute of Electrical Engineers of Japan and the Society of Instrument and Control Engineers.



Kotaro Hirasawa (Member) He received the B.S. and M.S. degree in Electrical Engineering from Kyushu University, Japan in 1964 and 1966, respectively. From 1966 to 1992, he was with Hitachi Ltd., where he served as a vice president of Hitachi Research Laboratory. From December 1992 to August 2002, he has been a Professor in Graduate School of Information Science and Electrical Engineering of Kyushu University. Now he belongs to Graduate School of Information, Production and Systems of Waseda University. Dr. Hirasawa is a member of the Society of Instrument and Control Engineers, a member of the Institute of Electrical Engineers of Japan, and member of IEEE.

Jinglu Hu (Member) He received the M.S. degree in Electronic Engineering from Zhongshan University, China in 1986, and the Ph.D degree in Computer Science and Engineering from Kyushu Institute of Technology, Japan in 1997. From 1986 to 1993, he worked in Zhongshan University, where he was a Research Associate and then Lecturer. Since 1997, he has been a Research Associate at Graduate School of Information Science and Electrical Engineering, Kyushu University, Japan. His current research interests are system identification, learning network and their applications. Dr. Hu is a member of the Society of Instrument and Control Engineers, a member of the Institute of Electrical Engineers of Japan.

Junichi Murata (Member) He received Doctor of Engineering degree from Kyushu University in 1986. He then became a Research Associate and an Associate Professor at Faculty of Engineering, Kyushu University. Now, he is an Associate Professor at Graduate School of Information Science and Electrical Engineering, Kyushu University. His current research interests are neural networks, self-organizing systems and their applications to control and identification. He is a member of IEEE, SICE, ISCIE and IEEJ