Genetic Network Programming with Reinforcement Learning and its Performance Evaluation

Shingo Mabu[1], Kotaro Hirasawa[1], and Jinglu Hu[1]

Graduate School of Information, Production, and Systems, Waseda University,
Hibikino 2-7, Wakamatsu-ku, kitakyushu, Fukuoka, 808-0135 Japan,
mabu@asagi.waseda.jp, {hirasawa, jinglu}@waseda.jp,

Abstract. A new graph-based evolutionary algorithm named "Genetic Network Programming, GNP" has been proposed. GNP represents its solutions as graph structures, which can improve the expression ability and performance. Since GA, GP and GNP already proposed are based on evolution and they cannot change their solutions until one generation ends, we propose GNP with Reinforcement Learning (GNP with RL) in this paper in order to search solutions quickly. Evolutionary algorithm of GNP makes very compact graph structure which contributes to reducing the size of the Q-table and saving memory. Reinforcement Learning of GNP improves search speed for solutions because it can use the information obtained during task execution.

1 Introduction

A new graph-based evolutionary algorithm named "Genetic Network Programming (GNP)" was proposed[1]. GNP represents its solutions as graph structures which have some distinguished abilities inherently. For example, GNP can memorize the past action sequences in the network flow and make quite compact structures. PADO[2] is also one of the graph-based evolutionary algorithm. The difference between PADO and GNP is that PADO aims to evolve programs in static environments, while GNP aims to evolve them in dynamic environments. However, conventional GNP are based on evolution, i.e., after the programs of GNP are carried out to some extent, they are evaluated and evolved based on their fitness values, so many trials must be done again and again. To overcome this problem and search solutions quickly, Q learning[3] which is one of the famous learning method was introduced for the learning of GNP[4, 5]. In this paper, a new algorithm of GNP with Reinforcement Learning (GNP with RL) is proposed. This method is the extension of the previous GNP, so it becomes more general framework of GNP with RL in terms that 1) we defined the new state and action pairs used by RL which are different from the previous method, 2) the proposed method can change node functions in addition to node connections and 3) reduce the size of the Q-table used by RL, so the calculation time and the physical memory can be saved.

[6, 7] are also the methods that combine RL and evolution. In [6], the special nodes for learning are introduced to GP[8, 9] and the contents of the nodes, namely the actions of agents, are determined by Q-learning, while in [7], Q-table is produced by GP in order to do Q-learning efficiently. For example, if the GP program is (TAB($*xy$)($+z5$)), it represents the 2 dimensional Q-table having 2 axes, $x * y$ and $z+5$. On the other hand, the proposed method in this paper determines node functions and connections of the GNP network efficiently by combining RL and evolution.

This paper is organized as follows. In the next section, the algorithm of the proposed method is described. Section 3 shows the results of the simulations. Section 4 is devoted to conclusions.

2 Genetic Network Programming (GNP)

In this section, Genetic Network programming is explained in detail. GNP is an extension of GP in terms of gene structures. The original motivation to develop GNP is based on the more general representation ability of graphs than that of trees, which is described in detail in this section.

2.1 Basic structure of GNP

First, we explain other evolutionary algorithms in order to compare them with GNP. GP can be used as a decision making tree when function 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. However, GP tree might cause the severe bloat that makes the search for solutions difficult due to the unnecessary expansion of the depth of the tree.

PADO is a graph-based evolutionary algorithm, but it has both the start node and terminal node unlike GNP. The node used in PADO has two functional parts: an action part and a branch-decision part. This branch-decision part selects the branch leaving the current node depending on the stack memory. If the current node reaches the end node before the threshold time, it runs again from the start node without initializing an indexed memory. PADO weights on the indexed memory because the node transition of PADO is based on the memory. PADO has been applied to image and sound classification problems, and splendid results have been shown. So PADO is distinguished for static problems, but GNP is developed to deal with dynamic problems.

Next, we explain the characteristics of GNP.

Components and Structure

Fig. 1 shows the basic structure of GNP. 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. Processing node determines an action/processing an agent should do. Contrary to judgement nodes, processing nodes have no conditional branches. The GNP we used never causes bloat

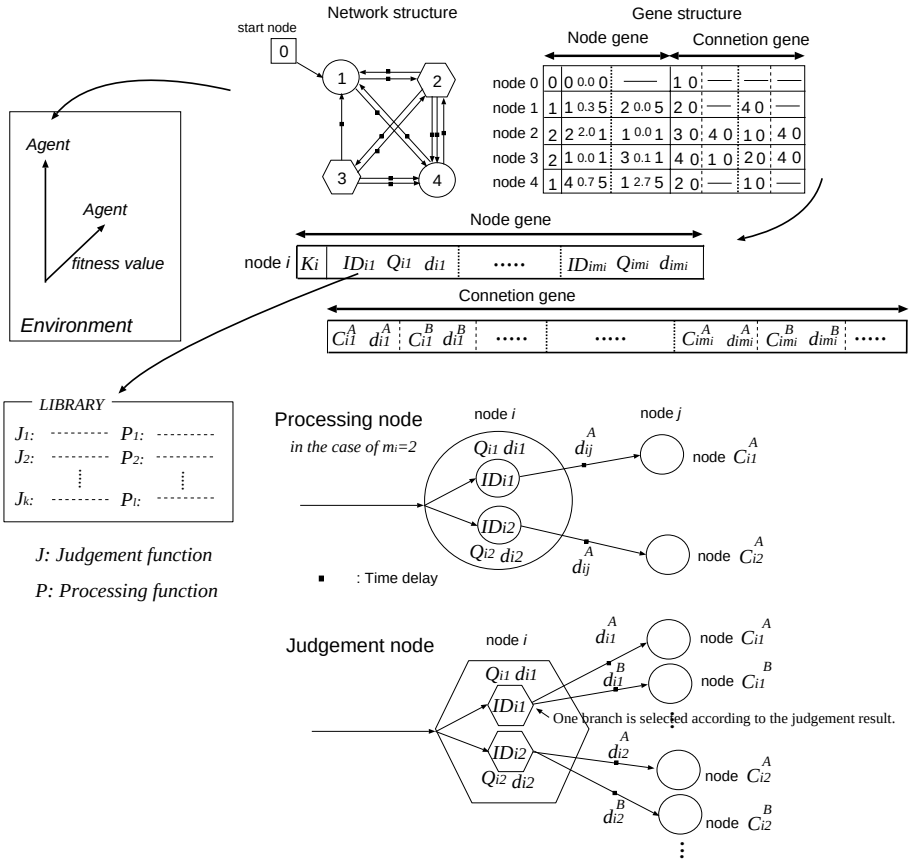

Fig. 1. Basic structure of GNP

because of the predefined number of nodes although GNP can evolve the programs having variable sizes. Because the graph structure of GNP inherently has the ability to re-use nodes unlike GA and GP[1], GNP can use certain Judgement/Processing nodes repeatedly to achieve tasks. Therefore, even if the number of nodes is predefined and smaller than GP programs, GNP can perform well by making effective programs based on re-using nodes. So we do not have to prepare the excessive number of nodes, as a result, we can easily determine the number of nodes experimentally, i.e., determine it manually according to the problems to be solved.

Memory function of graph

[1] GP can also re-use nodes by using ADFs.

Although GNP is booted up from the start node, there are no terminal nodes. After the start node, the current node is determined according to the connections of the nodes and judgement results, so GNP is carried out according to the network flow without any terminals, in other words, the determination of the current node is influenced by the node transitions of the past. Therefore, the graph structure itself implicitly has a memory function of the past actions of an agent. The node transition ends when the end condition is satisfied, for example, the time step reaches the maximum one or the GNP program completes the given tasks.

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 judgement), to put judgement into action (d_{ij} for transition from judgement to processing) and to avoid the puddle (d_j for processing). Since time delays are listed in each node gene and they are the unique attributes of each node, GNP can make the flexible system considering time delays. In this paper, for simplicity, d_{ij} is set at 0, d_i of judgement nodes is set at 1 time unit, and d_i of processing nodes is set at 5 time units. In addition, the one step of an agent's behavior is defined in such a way that one step ends when an agent uses 5 or more time units. So an agent can do less than 5 judgements and 1 processing, or 5 judgements in one step.

2.2 Gene structure of GNP

The whole structure of GNP is determined by the combination of the following node genes. A genetic codes of node i ($0 \leq i \leq n-1$[2]) is shown in Fig.1.

K_i represents the node type, $K_i = 0$ means Start node, $K_i = 1$ means Processing node and $K_i = 2$ means Judgement node. ID_{ip} ($1 \leq p \leq m_i$ [3]) shows the code number of judgements and processings, and they are represented as a unique number shown in the LIBRARY. In Fig. 1, m_i of all nodes are set at 2, i.e., GNP can select the node function ID_{i1} or ID_{i2}. Q_{ij} means Q-value which is assigned to each state and action pair. In this method, "State" means a current node, and "Action" means a selection of node function (ID_{ip}). d_{ip} is the time delay spent for judgement or processing. $C^A_{ip}, C^B_{ip}, \ldots$ show the node number of the next node j. $d^A_{ip}, d^B_{ip}, \ldots$ mean time delay spent for the transition from node i to node j.

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

[3] m_i ($1 \leq m_i \leq M$ M: Maximum number of functions in a node) shows the number of node functions GNP can select at the current node i.

Judgement node determines the upper suffix of the connection gene which indicates the judgement result. We can set the number of branches from the judgement nodes at any value depending on the problem. For example, we suppose branch A and B exist, then if the result of the judgement is "B", GNP refers to C_{ip}^{B} and d_{ip}^{B}, but processing nodes refer to only C_{ip}^{A} and d_{ip}^{A} because they have no conditional branches, i.e., no judgement functions.

2.3 Node transition rule of GNP

GNP starts its node transition from a start node, so the first current node is a start node. Then a current node is moved based on the connection of a start node.

If the current node i is a judgement node, first one Q-value is selected from $Q_{i1}, \ldots, Q_{im_i}$ based on ε-greedy policy. That is, a 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} is selected. After selecting a function, GNP executes the selected judgement function and determine the next node according to the judgement result. For example, if the selected function is ID_{i2} and the judgement result is "B", the next node becomes node C_{i2}^{B}.

If the current node is a processing node, a processing function is selected in the same way as judgement node. After GNP executes the selected processing, the next node becomes node C_{i2}^{A} if the selected function is ID_{i2}.

2.4 Evolution phase

Fig. 2 shows the whole flowchart of GNP. In this sub-section, the genetic operators in the evolution phase are introduced.

GNP realizes evolution using crossover and mutation. In each generation, the elite individuals are preserved and the rest of the individuals are replaced with the new ones generated by crossover and mutation.

Crossover Crossover is executed between two parents and generates two offspring [Fig. 3]. Crossover operator exchanges all the genes of the selected nodes.

1. Select two parents using tournament selection.
2. Each node i ($0 \leq i \leq n-1$) is selected as a crossover node with the probability of P_c.
3. Two parents exchange the genes of the corresponding selected nodes having the same node number.
4. Generated new individuals become the new ones of the next generation.

Fig. 3 is a simple crossover example of the graph structure with 3 processing nodes.

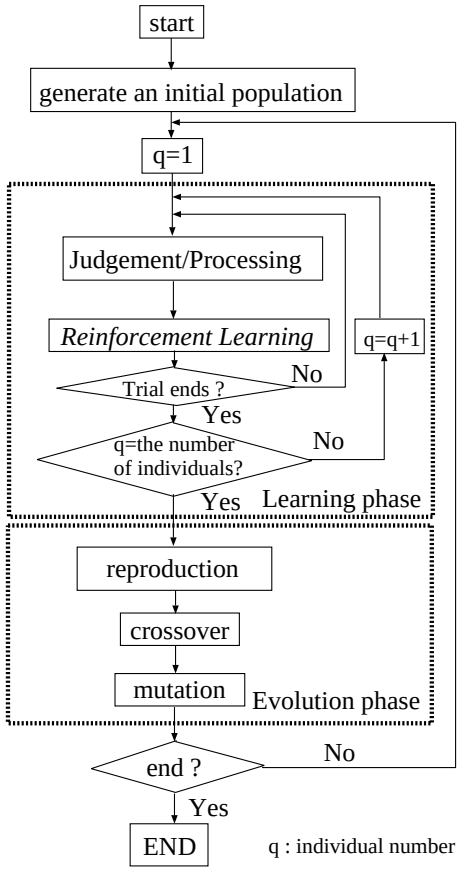

Fig. 2. Flowchart of GNP

Mutation Mutation is executed in one individual and a new one is generated [Fig. 4].

1. Select one individual using tournament selection
2. Mutation operator
   (a) connection : Each node branch is re-connected to the different node with the probability of P_{mc}.
   (b) the number of functions : Each node i is selected with the probability of P_{mn}, and the number of functions m_i is changed to $1, \ldots,$ or M. If m_i becomes greater than the previous one, then one or more new functions selected in the LIBRARY are added to the node. If m_i becomes smaller than the previous one, then one or more functions are deleted from the node.

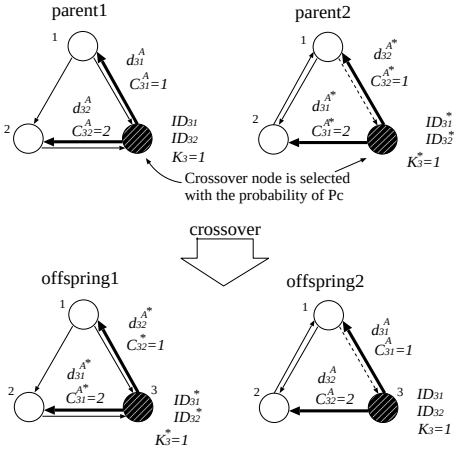

Fig. 3. Crossover

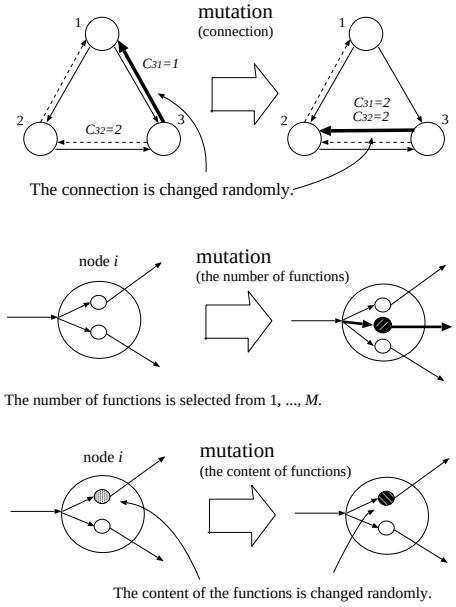

Fig. 4. Mutation

(c) the content of functions : Each function is selected with the probability of P_{mf}, and it is changed to the other function, i.e., ID_{ip} and d_{ip} are changed to the other ones.

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

2.5 Learning phase

In learning phase, node transitions are carried out and Q values are updated. Since RL of GNP is done when agents are carrying out their task, GNP can search for better solutions every judgement/processing besides the evolutional operation executed every generation. The aim of the combination between RL and evolution is to take advantage of the sophisticated search ability of evolution and online learning of RL.

Sarsa(λ) In this paper, Sarsa(λ) algorithm is used for the learning phase of GNP because the method with eligibility traces has good characteristics of both Monte Carlo and TD methods, i.e., using real experience and bootstrap simultaneously. Eligibility traces are useful for dealing with non-Markov tasks and long-delayed rewards.

Generally, state s is determined by the information an agent can get, and action a means the actual action it takes. In GNP learning, however, a state means a current node and an action means a selection of a function. Fig. 5

shows states, actions and an example of node transition. Since sarsa(λ) is used in learning phase, replacing traces e_{ip} are assigned to all state and action pairs. Here an example of node transition is explained using Fig. 5.

1. At time t, GNP refers to all Q_{ip} and select one of them based on ε-greedy. We suppose that GNP selects Q_{ip_1} and the corresponding function ID_{ip_1}.
2. Then GNP executes the function ID_{ip_1}, gets the reward r_t and the next node j becomes $C^A_{ip_1}$.
3. At time $t+1$, GNP selects one Q_{jp} in the same way as step 1. Here we suppose that Q_{jp_2} is selected.
4. Then the following procedure is executed.
   $\delta = r_t + \gamma Q_{jp_2} - Q_{ip_1}$, $e_{ip_1} = 1$
   For all i and p,
   $Q_{ip} = Q_{ip} + \alpha\delta e_{ip}$
   $e_{ip} = \gamma\lambda e_{ip}$
5. $t = t+1$, $i = j$, $p_1 = p_2$ then return step 2.

In this example, node i is a processing node, but if it is a judgement node, next current node is selected among $C^A_{ip}, C^B_{ip}, \ldots$ according to the judgement result.

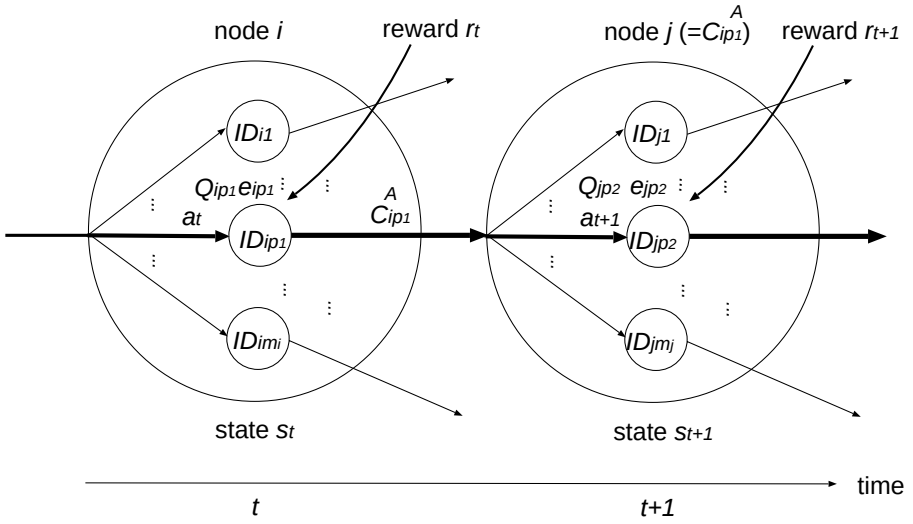

Fig. 5. An example of node transition

3 Simulations

To confirm the effectiveness of the proposed method, the simulations using tile-world and maze problem are done in this section.

Table 1. Function Set

Judgement node		Processing node	
symbol	content	symbol	content
JF	judge FORWARD	MF	move forward
JB	judge BACKWARD	TR	turn right
JL	judge LEFT side	TL	turn left
JR	judge RIGHT side	ST	stay
TD	direction of the nearest TILE from the agent	RD	random (MF, TR or TL)
HD	direction of the nearest HOLE from the agent		
THD	direction of the nearest HOLE from the nearest TILE		
STD	direction of the second nearest TILE		

3.1 Tileworld

The tileworld[10] used in the simulations is shown in Fig. 6 which is the 2D grid world including multi-agents (three agents in this paper), obstacles, tiles, holes and floors. Agents have some sensors and action abilities, and they aim to push and drop many tiles into holes as fast as possible. Since the given sensors and simple actions are not enough to achieve a task, agents should make the clever combinations of the sensor information (judgements) and actions (processings). The nodes used by agents are shown in Table 1, but RD (random) is not used in the tileworld environment. The judgement nodes { JF, JB, JL, JR } return { tile, hole, obstacle, floor or agent } and { TD, HD, THD, STD } return { forward, backward, left, right or nothing } as judgement results like $A, B, \ldots$ in Fig. 1.

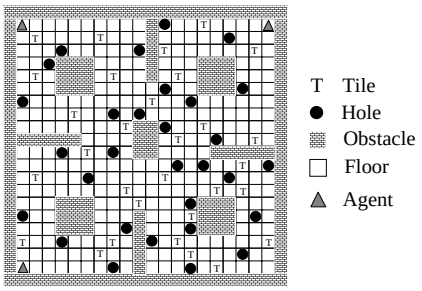

Fig. 6. Tileworld problem

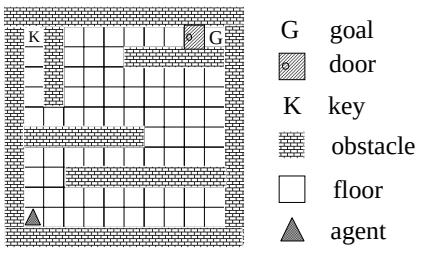

Fig. 7. Maze problem

Fitness and Reward A trial ends when the time step reaches the predefined step (300), and then fitness is calculated. "Fitness" is used in evolution phase

and "Reward" is used in learning phase.

Fitness = the number of dropped tiles

Reward =1 (when an agent drops a tile into a hole.)

Results We used the proposed method (GNP with RL), standard GNP (GNP), standard GP and GP with ADFs for the comparisons. Standard GNP uses only evolution to make its programs. The simulation conditions are shown in Table 2. Only GNP with RL preserves 5 elite individuals instead of 1 in order to make the system stable because the programs of GNP with RL are changed during task execution unlike the other methods. GP uses the judgement nodes of GNP as root and function nodes of the trees, and processing nodes are used as terminal nodes. A program of GP is executed from the root node, and GP repeats selecting branches at function nodes. Finally the content of the reached terminal node is executed. This procedure is defined as 1 step of GP. GP uses one-point crossover described in [11] and half-and-half initialization methods [8] for fixing the maximum depth of trees and producing trees with various sizes and shapes in an initial population. GP with ADFs has three ADFs in each individual.

Fig. 8 shows the fitness curves of GNP with RL, GNP and GP averaged over 30 simulations. The result of GP is based on the tree of maximum depth 6 because it shows the best results, and that of GP with ADFs is based on the main program (tree) of maximum depth 5 and the ADFs of maximum depth 4.From the results, GNP with RL shows the best fitness value. Although it seems to be natural that the method using RL can obtain better solutions than the other methods without it, the aim of developing GNP with RL is to solve the problems faster than the others in the same time limit of actions. In other words, GNP with RL aims to make full use of the information obtained during task execution for its learning.

In this problem, the arity of the function nodes of GP is relatively large (five), so the total number of nodes of GP becomes quite large as the depth becomes large. Although GP programs can have higher expression ability with the increase of the number of nodes, it takes much time to execute the programs and much memory is needed. For example, GP (depth6) takes too much time to execute the evolutionary operations and GP (depth7) cannot be executed because of the lack of memory in our machine (Pentium4 2.5GHz, DDR-SDRAM PC2100 512MB). GNP can obtain the good results using relatively small number of nodes (60 nodes).

3.2 Maze problem

Fig. 7 shows the environment used in the simulations. An agent should go to the goal but there is a door in front of the goal, so agent must take the key put on the upper left cell first in order to open the door. The nodes used by an agent are shown in Table 1, and an agent uses the processing nodes { MF, TL, TR, RD } and the judgement nodes { JF, JB, JL, JR } which return { floor, obstacle or goal } as a judgement result.

Table 2. Simulation conditions

		GNP with RL	GNP	GP
the number of individuals	crossover	120	120	120
	mutation	175	179	179
	elite	5	1	1
the number of nodes		60	60 (5 per each)	GP: max 19,531 GP+ADFs: 6,249
Crossover rate P_c		0.1	0.1	——
Mutation rate	P_{mc}, P_{mn}, P_{mf}	0.1, 0.01, 0.01	P_m=0.1	P_m=0.1
Learning parameters	step size α	0.1		
	discount rate γ	0.9	——	——
	trace decay parameter λ	0.9	——	——
	ε	0.1		

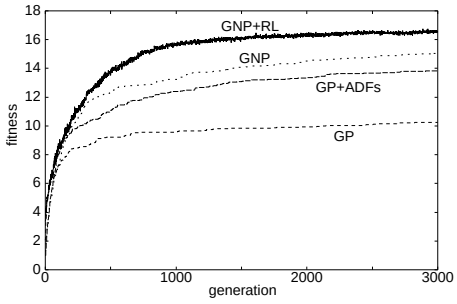

Fig. 8. Tileworld problem

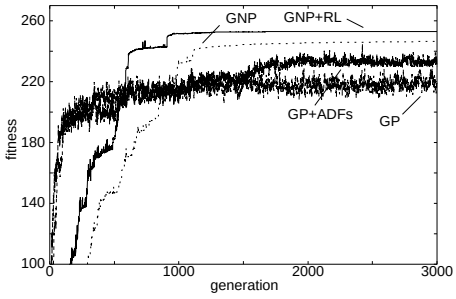

Fig. 9. Maze problem

Fitness and Reward The fitness and reward in maze problem are as follows and the time limit is also 300 steps.

Fitness = remaining time (if arriving at the goal), otherwise 0.
Reward =1 (when an agent arrives at the goal)

Result Fig. 9 shows the fitness curves averaged over 30 simulations.

From the results, GP(max depth 6) can learn faster than the other methods in the early generation. However, the characteristics of GNP are to reuse nodes and memory function, so GNP can find better solutions finally although it takes a little long time to make the complicated graph structure. In addition, GNP with RL can improve the learning speed and find better solution than standard GNP, and find the optimal policy (fitness=253, 47 steps) in all 30 simulations. GP can obtain the optimal policy some times, but there are some cases which show the fitness values around 200, so it degrades the average fitness.

4 Conclusion

In this paper, in order to enhance the performance of GNP, a new algorithm of GNP using reinforcement learning is proposed. Since GNP with RL can make full use of the information obtained during task execution, it is clarified from the simulations that the proposed method can improve the learning speed and find better solutions than standard GNP. In addition, the proposed method assigns a state to each node, so the number of states is that of nodes. As a result, the size of the Q-table becomes very small.

In a future, various problems will be solved in order to confirm the effectiveness of GNP with RL and comparisons with other methods such as GP with ADFs and other graph-based evolutionary methods like EP with FSMs will be done.

References

1. 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*, pp. 219-226, (2001).
2. A. Teller, and M.Veloso: "PADO, Learning Tree-structured Algorithm for Orchestration into an Object Recognition System", Carnegie Mellon University Technical Report Library, (1995)
3. Richard S. Sutton and Andrew G. Barto, "Reinforcement Learning - An Introduction", MIT Press Cambridge, Massachusetts, London, England, (1998).
4. S. Mabu, K. Hirasawa, J. Hu and J. Murata, "Online learning of Genetic Network Programming", *in Proc. of 2002 Congress on Evolutionary Computation*, pp. 321-326, (2002).
5. S. Mabu, K. Hirasawa and J. Hu, "Genetic Network Programming with Learning and Evolution for Adapting to Dynamical Environments", *in Proc. of 2003 Congress on Evolutionary Computation*, pp. 69-76, (2003).
6. Keith L. Downing, "Adaptive Genetic Programming via Reinforcement Learning", *in Proc. of the 3rd Genetic and Evolutionary Computation Conference*, pp. 19-26, (2001).
7. H. Iba, "Multi-agent reinforcement learning with genetic programming", *in Proc. of the Third Annual Conference of Genetic Programming*, pp. 167-172, (1998).
8. John R. Koza, "Genetic Programming, on the programming of computers by means of natural selection", Cambridge, Mass., MIT Press, (1992).
9. John R. Koza, "Genetic Programming II, Automatic Discovery of Reusable Programs", Cambridge, Mass., MIT Press, (1994).
10. M. E. Pollack and M. Ringuette, "Introducing the tile-world: Experimentally evaluating agent architectures", *in Proc. of the conference of the American Association for Artificial Intelligence*, pp. 183-189, (1990).
11. Riccardo Poli and William B. Langdon, "Schema Theory for Genetic Programming with One-point Crossover and Point Mutation, Evolutionary Computation", vol. 6, no. 3, pp. 231-252, (1998).