Performance of Genetic Network Programming for Learning Agents on Perceptual Aliasing Problem

Tadahiko Murata **Takashi Nakamura** **Sho Nagamine**

Department of Informatics, Kansai University
2-1-1, Ryozenji, Takatsuki, Osaka 569-1095, Japan
murata@res.kutc.kansai-u.ac.jp

Abstract – *In this paper, we examine the performance of genetic network programming (GNP) for learning agents on perceptual aliasing problems. Perceptual aliasing problems (PAP) are known as the problem where a learning agent can not distinguish between differing states of the world due to the limitation of its sensors. In order to cope with this problem, a genetic programming approach called Adaptive Genetic-Programming Automata (AGPA) has been proposed. While it effectively tackled to PAP, too many rules are generated that are not used to control the agent due to its tree-based structure. Using GNP, we can reduce the number of rules for PAP since it has network architecture but tree architecture as used in adaptive GP automata. We compare the performance of GNP and AGPA on a maze problem in which a learning agent tries to reach a goal. Simulation results clearly show that the number of rules can be reduced by GNP.*

Keywords: Perceptual aliasing problem, genetic network programming, adaptive GP automata, maze problems.

1 Introduction

When we try to tackle a learning agent problem, we should consider a problem that is called perceptual aliasing problem [1]. Perceptual aliasing problem is caused when sensors of an agent is limited, and the agent can not distinguish different states with the same inputs from its sensor. In order to solve these problems, GP-Automata has been proposed [2]. In GP-Automata, the agent can select its action according to the difference of its internal states. The number of internal states in an agent should be fixed in advance. If the number of states is specified appropriately, the agent can solve the problem. If the number is smaller than the required number, however, the agent can not solve the problem. On the other hand, if the number is larger than the required number, the search efficiency declines. That is, the performance of GP-Automata depends on the number of the internal states prepared beforehand. In order to design the number of internal states adaptively, Adaptive GP-Automata (AGPA) has been proposed [3]. In AGPA, the number of the internal states is optimized in evolutionary process. This method does not need trial and error for decision of the number of the internal states. While AGPA are more effective for perceptual aliasing problems in comparison with GP-Automata, many rules generated in a tree architecture are not used to attain the goal in the problem. In GP-Automata or AGPA, the number of generated rules increases because they employed decision tree representation to describe rules to control an agent.

Instead of using decision tree representation, GNP [4] employed network architecture. Using the network architecture, it can directly represent the flow of decision rules. We have already developed a variant of GNP, and showed its effectiveness in comparison with GP in multi-agent problems [5,6]. We found that GNP has more flexible representation to describe action rules for multiple agents with different roles. In this paper, we employ GNP to find decision rules for a learning agent in a maze problem with perceptual aliasing problem.

The remain of this paper is organized as follows: Section 2 explains the basic architecture of GNP [4-6]. In Section 3, we briefly explain about Adaptive GP-Automata [3]. Section 4 describes a method to count the number of rules generated by GNP in order to compare the readability of generated rules. Then computational results on maze problems are shown in Section 5. We conclude this paper in Section 6.

2 Genetic Network Programming

2.1 Basic Architecture of GNP

In this section, we describe the basic architecture of GNP [4-6]. GNP uses a network architecture instead of using a tree architecture. Fig. 1 shows the basic structure of GNP. In GNP, we have three types of nodes: start node, judgment node and processing node. In Fig. 1, the start node, the judgment node and the processing node are denoted by a square, a diamond and an open circle, respectively. The start node is like a root node in GP. The other two nodes, the judgment node and the processing node, correspond to the function node and the terminal node, respectively. Therefore we can employ the same function node and the terminal node used in GP as the judgment node and the processing node in GNP. Main difference between GP and GNP lies in the terminal node

or the processing node. As shown in Fig. 2, the terminal node of GP can not have a further connection. That is the reason why it is called as "terminal" node. On the other hand, the processing node in GNP can connect another node.

The difference between GNP and GP lies only in their architectures, but it brings great difference between them. For example, because the action of an agent is determined only in the terminal node in GP, the agent should return to the root node to take a next action. On the other hand, an agent in GNP acts when it finds a processing node, and it does not have to return to the start node after making its decision.

As for another disadvantage of GP, the performance of the tree may change greatly if the sub tree is exchanged by genetic operations at the node near to the root node in GP. On the other hand, the node exchange in GNP does not have such a great influence on the performance of the network.

2.2 Chromosome Representation

In order to generate a network as an individual in GNP, we assign $N+1$ nodes for one network randomly. The representation of each node is shown in Fig. 3. Each node has its ID number i ($i = 0,1,2,...,N$). As shown in Fig. 3, each node consists of two parts: node gene and connection gene. In the node gene, NT_i shows the type of the node i: "0" denotes the processing node and "1" denotes the judgment node. ID_i indicates the function of the node. If $NT_i = 0$ and there are P types of the processing node, ID_i varies from 0 to $P-1$. In the case of $NT_i = 1$ and there are J types of the judgment node, ID_i varies from 0 to $J-1$. According to the values of NT_i and ID_i, the function of the node i is defined. As for the start node, we do not assign NT_0 and ID_0 in the node gene.

In the connection gene, C_{ij} indicates the j-th connection from the node i, and n_i^{ark} shows the number of connections from the node i. If $NT_i = 0$, $n_i^{ark} = 1$ because the processing node has only one connection. When $NT_i = 1$, $n_i^{ark} \geq 2$ because a judgment node has several connections according to its condition. The value of the C_{ij} indicates the ID number of the node connected from the node i.

In the initial generation, we first generate the N nodes by assigning NT_i and ID_i for $i = 1,...,N$. After that we copy the set of generated N nodes for N_{pop} individuals. To the N nodes in each individual, we randomly assign the connection gene C_{ij} for each individual to form a population with the specified number of individuals. Therefore, the node gene of each individual has the same NT_i and ID_i, but the connection gene has different connections among N nodes.

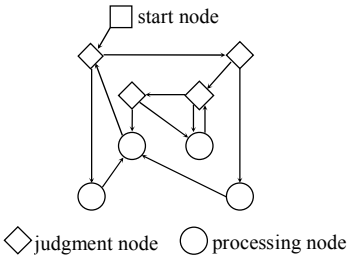

Figure 1. Basic Structure of GNP.

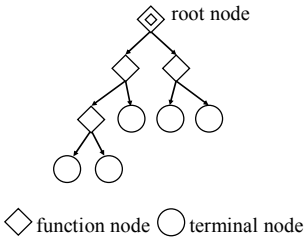

Figure 2. Basic Structure of GP.

	Node Gene		Connection Gene				
node i	NT_i	ID_i	C_{i1}	C_{i2}	----	----	$C_{in_i^{ark}}$

Figure 3. Representation for the node.

2.3 Genetic Operations for GNP

In order to create various connections among nodes, GNP has the two types of genetic operations: crossover and mutation. Fig. 4 shows a crossover operation in this paper. The procedure of the crossover is as follows:

[Crossover in GNP]

Step 1: Using the tournament selection, select two networks, and apply the crossover to them according to the crossover probability P_c.

Step 2: Select nodes randomly in one network.

Step 3: Exchange the selected nodes between two networks.

Step 4: Repeat Step 1 through 4 until the prespecified number of offspring is generated.

As shown in Fig. 4, the connections of the randomly selected nodes are exchanged by this crossover.

Fig. 5 shows a mutation operation for connection gene. The procedure of the mutation for the connection gene in every individual is as follows:

[Mutation in GNP]

Step 1: According to the mutation probability P_m, select a connection.

Step 2: Turn the value of the selected connection to a rondomly specified node ID.

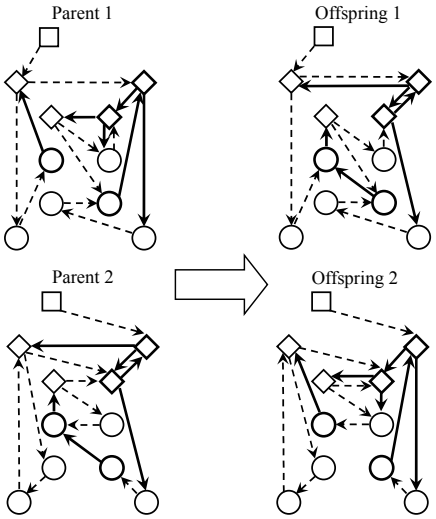

Figure 4. Crossover in GNP.

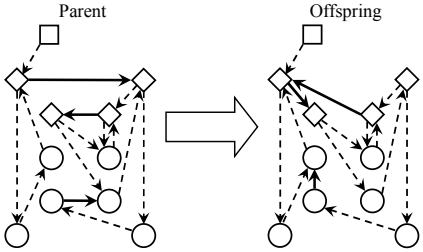

Figure 5. Mutation for connection gene in GNP.

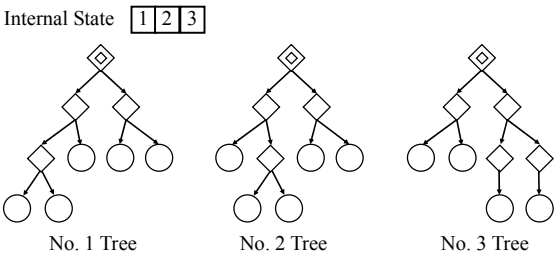

Figure 6. GP-Automata.

2.4 Algorithm of GNP

Using the initialization process and the genetic operations in Subsections 2.2 and 2.3, we form the following algorithm for GNP.

[GNP Algorithm]

Step 1 (Initialization)
Initialize the population with N_{pop} individuals.

Step 2 (Fitness evaluation)
Calculate the fitness value of each individual, and find an elitist individual with the best fitness value in the population.

Step 3 (Genetic operations)
Step 3-1 (Selection): Select parents for crossover by the tournament selection.
Step 3-2 (Crossover): Apply the crossover operator to the selected parents.
Step 3-3 (Mutation): Apply the mutation operator to the connection genes.
Step 3-4 (Elite strategy): Preserve the elitist individual found in Step 2 or Step 5.

Step 4 (Replacement)
Replace the newly generated population with the previous population.

Step 5 (Fitness evaluation)
Calculate the fitness value of each individual, and find an elitist individual with the best fitness value in the population.

Step 6 (Termination Condition)
Terminate the algorithm if the specified condition is satisfied. Otherwise return to Step 3.

3 Adaptive GP-Automata

When we apply a GP for learning agent to some perceptual aliasing problem, an agent should select appropriate action even if its sensor receives the same signal. In order to do that, the agent should have internal state to distinguish the same signal from the sensor as different states. However, if it should face a complicated problem, it becomes difficult to specify appropriate decision trees corresponding to internal states in advance. Ashlock [2] proposed his GP-Automata to cope with this problem. Fig. 6 shows a basic structure of GP-Automata. The finite number of internal states is prepared in advance, then a decision tree is developed by genetic operations for each internal states. The agent takes its action according to its internal state number and the corresponding decision tree. For example, If the current state is 2, the agent selects No. 2 decision tree, and follow the function node according to the signal from its sensor. It takes the corresponding action of a terminal node. Terminal nodes should include actions which transit its internal states.

While GP-Automata is effective to perceptual aliasing problems, the number of internal states should be appropriately specified in advance. If the number is smaller than the required number, the agent can not solve the problem. On the other hand, if the number is larger than the required number, the search efficiency declines. That is, the performance of GP-Automata depends on the number of the internal states prepared beforehand. For an adaptive design of the number of internal states, Adaptive GP-Automata (AGPA) has been proposed [3]. In AGPA, the number of the internal states is optimized in evolutionary process as well as decision trees. This method adaptively specify the number of the internal states.

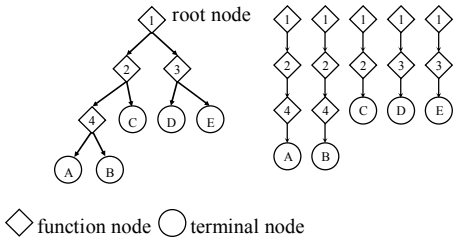

Figure 7. Rules in a tree.

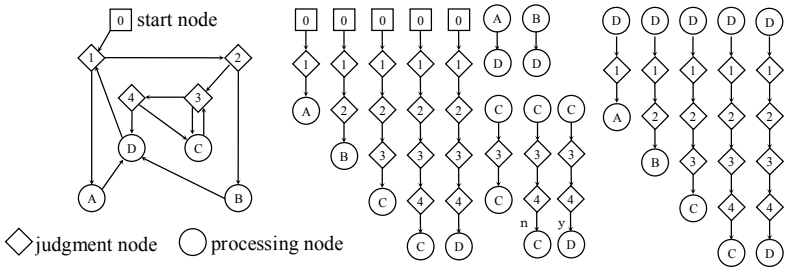

Figure 8. Rules in a network.

4 Number of Rules in GP & GNP

In this paper, an agent takes its action according to a network generated by GNP or trees with an internal state coded by AGPA. A processing node in GNP or a terminal node in AGPA can be regarded as a concluding part of each decision rule. For example, as shown in Fig. 7, if a tree consists of four function nodes (i.e., Nodes 1 through 4), and five terminal nodes (i.e., Nodes A through E), the tree has the following five rules:

If "1" & "2" & "4" are true then take Action "A".
If "1" & "2" are true & "4" is false then take Action "B".
If "1" is true & "2" is false then take Action "C".
If "1" is false & "3" is true then take Action "D".
If "1" is false & "3" is false then take Action "E".

Though it is easy to take rules from a tree structure, we should define to take rules from a network structure for GNP. When we have one start node, four judgment nodes and four processing nodes as shown in Fig. 8, we divide a network to several rules from the start node as follows:

Step 0: Start from the start node.
Step 1: Follow the network until a processing node is found. Repeat until all processing nodes from the start node are found.
Step 2: Start from the processing nodes found in Step 1, follow the network until an unfound processing node is obtained. Repeat until all processing nodes are found.
Step 3: Terminate the procedure.

Using this procedure, we find 15 rules from the network in Fig. 8. In this procedure, we regard a processing node as a condition. For example, to take Action "D", there are 5 routes (or rules) as follows:

If "1" & "2" & "3" & "4" are true
then take Action "D".
If Action "A" is taken
then take Action "D".
If Action "B" is taken
then take Action "D".

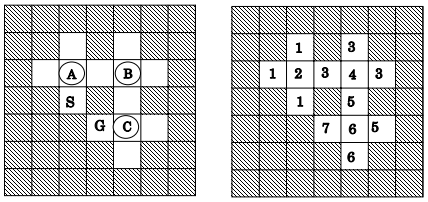

Figure 9. Maze A.

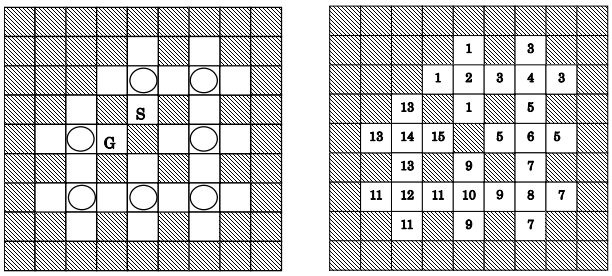

Figure 10. Maze B.

If "C" is taken & "3" & "4" are true
then take Action "D".
If "D" is taken & "1" & "2" & "3" & "4" are true
then take Action "D".

We count the number of rules in a network using this procedure.

5 Simulations on Maze Problems

We apply our GNP and the AGPA to maze problems with perceptual aliasing problems. Using GNP or AGPA, we develop decision rules for an agent to reach a goal in a maze. We apply them to two maze problems shown in Figs. 9 and 10. Maze A is a 7×7 grid world. An agent is set at a start cell denoted by "S". It can move one cell in the grid world at each step. The goal in the grid is shown by "G" in these figures. Circles in a cell shows that an agent can not distinguish the difference of the current state. That is, in that cell, the agent receives a signal that there are no walls in the north, the south, the east, and the west of it. However, the agent should move to the east when it is in the cell just above the start cell (Cell A). On the other hand, it should move to the south in Cell B, and should move to the west

in Cell C. The agent should distinguish the current state not only its sensor but also the internal state or the sequence of the decisions. While AGPA utilizes the information on internal states, GNP utilize the information on the sequence of the decisions. We show computer simulation results of these algorithms in the next section.

In Fig. 9, the agent should take the following actions from "S" to reach "G": Move North, Move East, Move East, Move South, Move South, and Move West. Each numeral depicted in the right figure in Fig. 9 indicates a reward obtained in the corresponding cell. If an agent can take the optimal actions in Maze A, it receive the following rewards: $1+2+3+4+5+6+7=28$. In this paper, we allow an agent to find the goal 9 steps. Therefore the agent continues to receive reward after reaching the goal until 9 steps are past. If the agent reach the goal in sixth step, it receive 21 rewards more at the goal. Thus the optimal rewards becomes 49 in this maze problem. We employ this reward as a fitness function in GNP and AGPA.

In Maze B, we allow an agent to find the goal in 17 steps. In that case, the optimal total reward becomes 165.

After preliminary experiments to define the value of parameters in GNP and AGPA, we specified the parameters as follows:

Common Parameters in Maze A & Maze B:
- The number of terminate generation: 150,
- Crossover Probability: 0.7
- Crossover Operation:
  - Uniform crossover in GNP,
  - One point crossover in AGPA,
- Mutation Probability: 0.01,
- The number of elite solutions: 10,
- The number of trials: 100

Maze A:
- The number of individuals in a population: 200
- Tournament size: 5.

Maze B:
- The number of individuals in a population: 400,
- Tournament size: 10.

6 Simulations Results

We apply GNP and AGPA using the above parameter specifications to Maze A and Maze B. Figs. 11 and 12 show the average fitness (i.e., rewards) of 100 trials over 150 generations. From Fig. 11, we can see that both algorithms, GNP and AGPA, could find decision rules that enable an agent to obtain the maximum fitness (i.e., 49) among all the 100 trials. From Fig. 12, we can see that GNP could obtain the better average fitness than AGPA in Maze B. These figures show that GNP can search the better results faster than AGPA.

Table 1 shows the statistics obtained in the simulations on Maze A. The first row indicates the number of trials in which the optimal solution is found. In Maze A, both the algorithms could obtain the maximum fitness among all the 100 trials as we can see in Fig. 11 and Table 1. The second row shows the average generation to obtain the optimal solution. From these statistics, we can see that GNP could obtain the optimal solution faster than AGPA.

From Table 2, we can see that the optimal solution could not be obtained in all the trials. But GNP could mark the better number of trials in which the optimal solution is obtained than AGPA. We average the number of generations in which the optimal solution is found. From this result, we can see that GNP can find the optimal solution faster than AGPA.

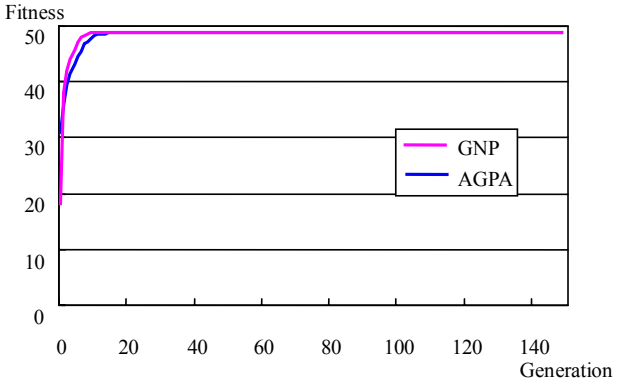

Figure 11. Average Fitness in Maze A.

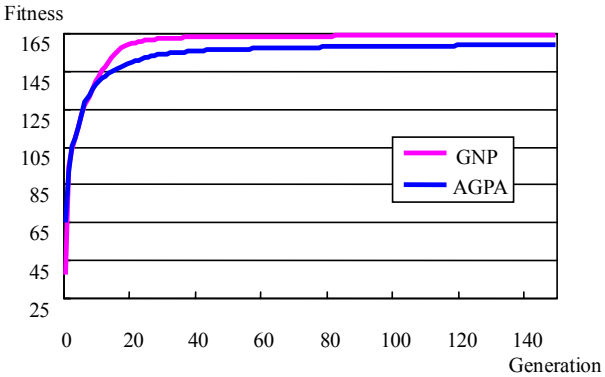

Figure 12. Average Fitness in Maze B.

Table 1. Statisitcs in Maze A.

	GNP	AGPA
The number of trials in which the optimal solution is obtained	100	100
The average generation to obtain the optimal solution	12.3	28.6

Table 2. Statisitcs in Maze B.

	GNP	AGPA
The number of trials in which the optimal solution is obtained	95	56
The average generation to obtain the optimal solution	24.5	62.5

Table 3. The number of rules obtained and used (Maze A).

	GNP	AGPA
# of rules in the best individual	12.8	32.0
Used rules	8.0	8.1

Table 4. The number of rules obtained and used (Maze B).

	GNP	AGPA
# of rules in the best individual	39.6	56.3
Used rules	15.9	15.7

Tables 3 and 4 show the average number of rules in the best solution generated by GNP and AGPA in Maze A and Maze B, respectively. The first row shows the number of rules obtained. We count the number of rules in the network obtained by GNP using the method described in Section 4. The second row shows the number of rules used by an agent to reach the goal.

From Table 3, we can see that while the average number of used rules is almost the same in GNP and AGPA, the number of rules in the network or the decision trees is different. AGPA produced more rules than GNP since it develops several trees to cope with the perceptual aliasing problem. The same observation can be done in Table 4. In AGPA, function nodes may produce unnecessary rules in order to generate necessary rules. On the other hand, GNP can concatenate processing nodes (i.e., action nodes), it can reduce the number of rules to enable an agent to reach the goal.

7 Conclusion

In this paper, we examine the effectiveness of GNP by applying it to perceptual aliasing problems. By comparing GNP with Adaptive GP-Automata, we can find that GNP can produce a small number of rules that can attain the goal. We pointed out the weakness of AGPA that it has the tendency to generate more rules than necessity. In order to cope with this problem, we can employ a function node like "prog *N*" that can concatenate *N* terminate nodes. Using such function node, the number of generated rules may be reduced in AGPA. We should try to improve the architecture of AGPA to compare it with GNP.

Acknowledgement

This research is supported by "Collaboration with Local Communities" Project for Private Universities: Matching fund subsidy from MEXT (Ministry of Education, Culture, Sports, Science and Technology), 2005-2009.

References

[1] S. D. Whitehead and D. H. Ballard, "Learning to perceive and act," *Technical report of Computer Science Department, University of Rochester*, 1990.

[2] D . Ashlock, "GP-Automata for Dividing the Dollar," *Proc. of Genetic Programming 1997*, pp. 18-26, 1997.

[3] H. Kataoka, A. Hara, and T. Nagao, "Action Control of an Agent Using Adaptive GP-Automata," *Information Processing Society Journal*, Vol. 44, No. 9, pp. 2390-2400, Sep. 2003.

[4] H. Katagiri, K. Hirasawa, J. Hu, and J. Murata, "Network structure oriented evolutionary model -Genetic Network Programming- and its comparison with Genetic Programming," *Proc. of 2001 GECCO*, p. 219, 2001.

[5] T. Murata and T. Nakamura, "Developing Cooperation of Multiple Agents Using Genetic Network Programming with Automatically Defined Groups," *Proc. of Late Breaking Papers in GECCO 2004*, 12 pages in CD-ROM, 2004.

[6] T. Murata and T. Nakamura, "Genetic Network Programming with Automatically Defined Groups for Assigning Proper Roles to Multiple Agents," *Proc. of 2005 GECCO*, pp. 1705-1712, 2005.