A Structural Optimization Method of Genetic Network Programming for Enhancing Generalization Ability

Shingo Mabu, Yamaguchi University, Japan, mabu@yamaguchi-u.ac.jp
Shun Gotoh, Yamaguchi University, Japan
Masanao Obayashi, Yamaguchi University, Japan, m.obayas@yamaguchi-u.ac.jp
Takashi Kuremoto, Yamaguchi University, Japan, wu@yamaguchi-u.ac.jp

Abstract: Genetic Network Programming (GNP), one of the evolutionary algorithms, has been proposed as an extension of Genetic Algorithm (GA) and Genetic Programming (GP), and the distinguished evolutional abilities have been verified. The program of GNP is represented by a directed graph structure and the node transition of the graph structure represents action rules. In practical applications, the robustness and generalization ability are very important, thus, we propose a method of structural optimization for enhancing the generalization ability of GNP in this paper, where the connections of initial nodes of graph structures are changed by mutation more frequently than the conventional method in order to use/evaluate variety of node combinations to create effective action rules. The generalization ability is evaluated by the Tile-World problem that is one of the benchmark problems of multi-agent systems, where the fitness obtained in the testing (unknown) environments is measured and the generalization ability of the proposed method is clarified.

Key-Words: *evolutionary computation, genetic network programming, graph structure, generalization, structural optimization*

1. Introduction

Recently, soft computing techniques such as Neural Networks, Fuzzy reasoning, and reinforcement learning are applied to various fields, and their usefulness has been shown. The advantage of the soft computing is that humans do not need to prepare complete control rules, but computers can obtain the flexible solutions.

Each of these approaches may require robustness and generalization ability, that is, the ability to output the appropriate solutions not only in the training environments, but even in unknown environments. The robustness and generalization ability are required for any approaches when we consider the practical applications.

On the other hand, Genetic Network Programming (GNP), one of the evolutionary algorithms, has been proposed as an extension of Genetic Algorithm (GA) [1] and Genetic Programming (GP) [2], and the distinguished evolutional abilities have been verified [3] comparing to GP, GP with automatic defined functions, and evolutionary programming (EP) [4]. Originally, GNP was proposed because graph structures basically have better representation abilities to make programs than string and tree structures. For example, node transitions in graph structures can make some repetitive processes like subroutines because some nodes can be repeatedly used by the node transition. In addition, after starting the node transition in a graph structure, GNP executes judgment nodes (if-then functions) and processing nodes (action functions) according to the connections between nodes without any terminal nodes. Thus, the node transition implicitly memorizes the history of judgments and processing, which contributes to the decision making in dynamic environments because GNP can make decisions based not

only on the current, but also the past information [3]. GNP has been applied to decision making systems such as stock trading systems [5], elevator group supervisory control systems [6], robot navigation systems [7], reduction of driving cost for hybrid electric vehicles [8], etc.

GNP has been also applied to data mining [9, 10], where a large number of important association rules can be extracted from databases. The data mining method was applied to network intrusion detection systems [11] where misuse detection and anomaly detection were realized with high detection rates.

In practical applications, the generalization ability has to be enhanced to adapt to various kinds of situations. Therefore, we propose a new structural optimization method of GNP in this paper. In the conventional GNP, even if a large number of nodes are prepared in a graph structure, only small parts of the nodes are used by evolution to create programs. The advantage of such evolutional result is the efficiency of the use of nodes, that is, only important nodes are selected to make rules. Thus, compact programs with effective action rules can be created. However, the disadvantage is that various kinds of action rules for adapting to various situations cannot be created only by the small parts of the program. To enhance the generalization ability, enough experiences in the training environments have to be stored in the evolved programs. Therefore, in this paper, we propose a method that sets mutation rates of initial nodes and other nodes at different values, respectively. In more detail, the mutation rate of initial nodes are set at high value to start the node transition from various places of graph structures, which contributes to use various nodes and more effective action rules can be created.

The performance of the proposed method is evaluated by the Tile-World problem [12] that is one of the benchmark problems of multi-agent systems, where the fitness obtained in the training and testing environments are compared between the proposed method and the conventional method.

This paper is organized as follows. Section 2 explains the evolution and the structure of GNP. Section 3 explains the algorithm of the proposed method. Section 4 explains the Tile-World problem, simulation conditions, and experimental results. Finally, section 5 is devoted to conclusions.

2. Genetic Network Programming, GNP

2.1 Basic structure and nodes

The program of GNP consists of one initial node, several judgment nodes and processing nodes, and it can create complex rules by connecting the nodes as a directed graph as shown in Figure 1.

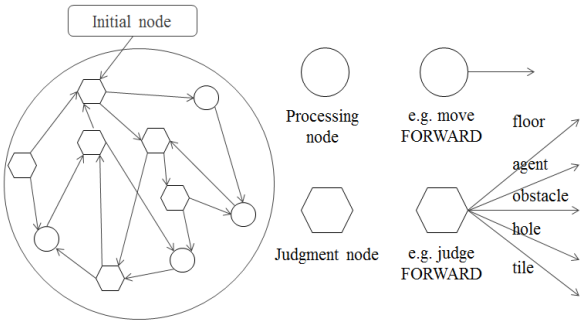

Figure 1 The structure of GNP

An initial node has no judgment or action function, but a connection to another node to be executed. A judgment node has several connections to other nodes, and one of them is selected by if-then type branch decision function. The judgment function of each judgment node is assigned by the designer at the initialization of the population. A processing node has an action function and only one connection to another node, therefore, after executing the action, the node transition transfers to the next node according to the connection.

The program execution (node transition) starts from the initial node, and the connections of judgment and processing nodes create effective if-then rules. Therefore, the role of evolution is to optimize connections between nodes. The previous research did not pay enough attention to the evolution of the connections of initial nodes, and the connections of initial nodes were changed with low probability in the evolutional process. However, in this paper,

the mutation frequency of initial nodes is changed to larger in order to execute enough exploration of graph structures for enhancing the generalization ability, which is described in Section 3.

2.2 Initialization of the population

The procedure of creating initial individuals is as follows:

(1) Prepare judgment nodes and processing nodes. The number of nodes in one individual is determined arbitrarily.
(2) Randomly determine the connection of the initial node.
(3) Assign judgment or processing function to each node. The number of functions and total number of nodes are determined depending on the problem.
(4) Randomly determine the connections of judgment and processing nodes. But, the self-loop is prohibited.
(5) Set the delay time [3], which is determined by the designer in advance.
(6) Repeat (1)–(5) to generate the total number of individuals set in advance.

2.3 Genetic Operation

Some individuals for the next generation are created by crossover and others are created by mutation, which means that there are no individuals created by both genetic operations. In fact, if both operations are carried out, destructive effect on the graph structures may occur.

2.3.1 Selection

The elite individual is taken over the next generation by the elite selection. Tournament selection is also used for selecting parent individuals in crossover and mutation.

2.3.2 Crossover

Crossover is operated by the following procedure:

(1) Two parent individuals are selected by executing tournament selection twice.
(2) The nodes in the parent individuals are randomly chosen by the crossover rate.
(3) Connections and node functions of the selected nodes are exchanged between the two parent individuals.

2.3.3 Mutation

Mutation is operated by the following procedure:

(1) One parent individual is selected by executing tournament selection once.
(2) Connections of each node in the parent individual are randomly selected by the mutation rate.
(3) The selected connections are re-connected to other nodes randomly.

3. The proposed Method

In the conventional GNP, we found that there is a tendency of over-fitting to the training environments. Since it is necessary to cope with various situations as well as some specific situations in order to have good generalization ability, various kinds of rules should be learned during evolution. Thus, we propose a method of structural optimization for the improvement of generalization ability of GNP.

3.1 Algorithm of the proposed method

We focus on the evolution of the initial node and propose a method that changes the connections of initial nodes more frequently than the conventional method, which is operated by the following procedure:

(1) When initializing individuals, the connection of the initial node in each individual is connected to one of the nodes in the graph structure randomly. Thus, the destinations of the connections from the initial nodes are different individual by individual.
(2) In the mutation, the connection of the initial node in a parent individual is selected according to the mutation rate of initial node. In section 4, the mutation rate of initial node is experimentally determined and fixed every generation. In detail, the mutation rates of initial node {0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0} are evaluated and that showing the best result (0.8) is used in the simulations.
(3) The connections of the initial nodes selected in step (2) are re-connected to another node randomly.

Actually, in the previous research, the effect of mutation rate of initial node was not analyzed in detail, however, it will

be clarified from the simulation results in Section 4 that relatively large mutation rate of initial node is effective for enhancing the generalization ability. In this paper, the mutation rate of initial node is experimentally determined, but the optimization of the mutation rate can be considered in the future.

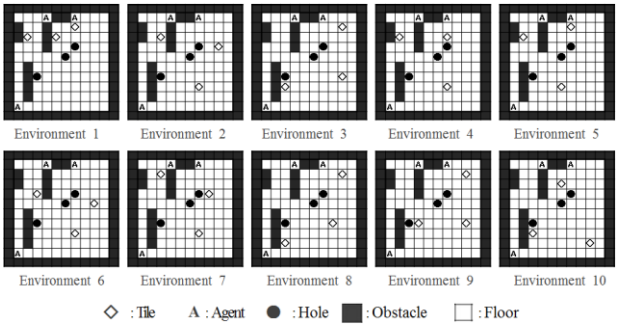

Figure 2 10 training environments

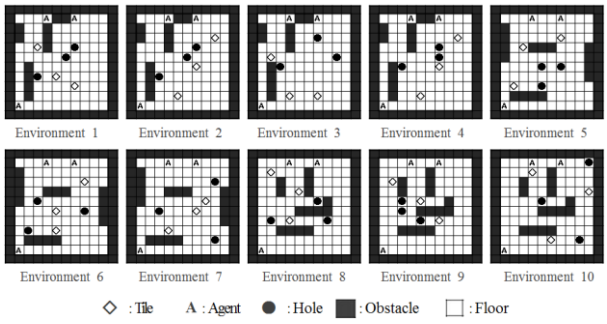

Figure 3 10 testing (unknown) environments

Table 1 Node functions

Processing nodes	Judgment nodes
(1) move forward	(1) judge forward
(2) turn right	(2) judge backward
(3) turn left	(3) judge left side
(4) stay	(4) judge right side
-	(5) direction of the nearest tile from the agent
-	(6) direction of the nearest hole from the agent
-	(7) direction of the nearest hole from the nearest tile
-	(8) direction of the second nearest tile from the agent

4. Simulations

The generalization ability is evaluated by the Tile-World problem that is one of the benchmark problems of multi-agent systems, where the fitness obtained in the training and testing (unknown) environments is measured and the effectiveness of the proposed method is clarified.

4.1 Tile-World

Tile-World consists of floor, obstacles, three agents, three holes and three tiles. The task of the agents is to drop all the tiles into the holes. When a tile is dropped into a hole, the tile and hole will disappear, that is, the hole is filled with the tile and becomes floor. Fitness of each individual is measured by the following formula.

$$\begin{aligned}\text{Fitness} &= 100\times(\text{the number of dropped tiles})\\ &+10\times(\text{how many cells the agents make the tiles close to the holes})\\ &+(\text{remaining time steps}),\end{aligned}$$

where, the remaining time steps is added to the fitness only when all the tiles are dropped into holes before reaching the time step limit. We use 20 training environments (Figure 2), which are 10 manually made environments and 10 randomly generated environments with some conditions/restrictions, e.g., the conditions guaranteeing the possibility of dropping tiles into holes. After measuring the fitness in the training environments, the elite individual in each generation is tested in 20 testing environments (Figure 3). The testing environments also consist of 10 manually made environments and 10 randomly generated environments.

4.2 Nodes

Four types of processing node functions and eight types of judgment node functions are prepared in the simulations, therefore, totally 12 kinds of functions are used (Table 1). Each individual is evolved by optimizing node connections. In the simulations, the number of nodes is set at 60, where five nodes for each kind of function are prepared, thus the total number of nodes is $60\,(=5\times12)$.

4.3 Parameters

The parameter settings used in the simulations are shown in Table 2. Each value in Table 2 is determined referring to the previous work [3] and also adjusted experimentally through the simulations.

4.4 Simulation results

The average fitness obtained in the training environments is shown in Figure 4, and that in the testing environments is shown in Figure 5. As it can be seen from the figures, the proposed method obtains higher fitness than the conventional method with lower mutation rate in both training and testing environments. In particular, in the testing environments, the conventional method shows the convergence of fitness improvement around 100th generation, while the fitness of the proposed method still continues to improve after that. The larger mutation rate of initial node accelerates the evolution of the entire structure, which improves the fitness not only in the training environments but also in the testing environments. By starting program execution from different nodes, the optimization of the connections has been carried out efficiently.

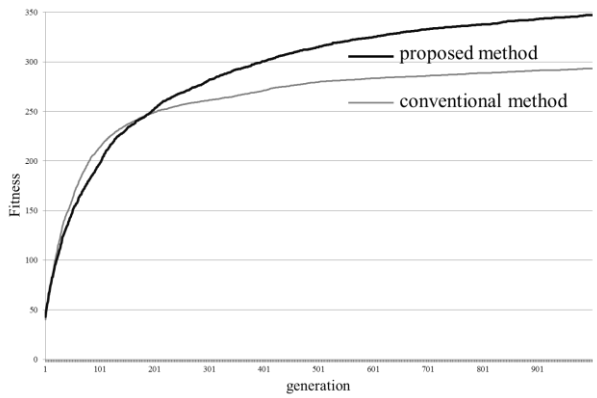

Figure 4 Fitness curves in the training environments

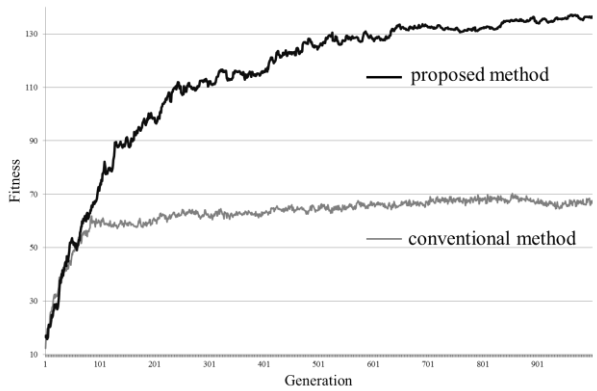

Figure 5 Fitness curves in the testing environments

4. Conclusion

In this paper, we focused on analyzing the effect of mutation for initial nodes, and proposed a method that sets the mutation rates for initial nodes and other nodes at different values. In the comparison of the fitness between the proposed and conventional methods in the testing environments, it could be seen that the generalization ability was improved by introducing the proposed method. In the future, we will consider applying the proposed method to distributed Genetic Network Programming (DGNP), which is an extension of GNP [13] for dealing with much more complicated problems, to improve the generalization performance of DGNP.

References:

[1] J. H. Holland, Adaptation in Natural and Artificial Systems, University of Michigan Press (1975)

[2] J. R. Koza, Genetic Programming, On the Programming of Computers by Means of Natural Selection, MIT Press (1992)

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

[4] D. B. Fogel, An introduction to simulated evolutionary optimization, IEEE Trans. on Neural Networks, Vol. 5, No. 1, pp. 3-14 (1994)

[5] S. Mabu, K. Hirasawa, M. Obayashi, and T. Kuremoto, Enhanced decision making mechanism of rule-based genetic network programming for creating stock trading signals, Expert Systems with Applications, Vol.40, No. pp. 6311-6320 (2013)

[6] A Double-Deck Elevator Group Supervisory Control System Using Genetic Network Programming, K. Hirasawa, T. Eguchi, J. Zhou, L. Yu, S. Markon, IEEE Trans. on Systems, Man and Cybernetics, Part C, Vol. 38, No. 4, pp. 535-550, 2008/7

[7] S. Mabu, A. Tjahjadi and K. Hirasawa, Adaptability Analysis of Genetic Network Programming with Reinforcement Learning in Dynamically Changing Environments, Expert Systems with Applications, Vol. 39, No. 16, pp. 12349-12357 (2012)

[8] K. Ishikawa, K. Nakazawa, T. Yamazaki, S. Furugori, T. Suetomi

and Y. Matsuoka, Reduction of the Driving Cost for Capacitor-Battery Combined HEV by Using Genetic Network Programming, Trans. of the Japan Society of Mechanical Engineers Ser. C, Vol. 79, No. 803, pp. 2259-2272 (2013) (in Japanese)

[9] K. Shimada, K. Hirasawa and J. Hu, Genetic Network Programming with Acquisition Mechanism of Association Rules, Journal of Advanced Computational Intelligence and Intelligent Informatics, Vol. 10, No. 1, pp. 102-111 (2006)

[10] K. Shimada, K. Hirasawa, and J. Hu, Class association rule mining with chi-squared test using genetic network programming, Proc. of the IEEE International Conference on Systems, Man and Cybernetics, pp. 5338-5344 (2006).

[11] S. Mabu, C. Chen, N. Lu, K. Shimada and K. Hirasawa, An Intrusion-Detection Model Based on Fuzzy Class-Association-Rule Mining Using Genetic Network Programming, IEEE Trans. on Systems, Man, and Cybernetics, Part C, Vol. 41, No. 1, pp. 130-139,

[12] 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)

[13] S. Mabu, K. Hirasawa, M. Obayashi and T. Kuremoto, Variable size mechanism of distributed graph programs and its performance evaluation in agent control problems, Expert Systems with Applications, Vol. 41, No. 4, pp. 1663-1671 (2014)