Developing Cooperation of Multiple Agents Using Genetic Network Programming with Automatically Defined Groups

Tadahiko Murata and Takashi Nakamura

Faculty of Informatics, Kansai University
2-1-1 Ryozenji, Takatsuki 569-1095, Osaka, Japan
murata@res.kutc.kansai-u.ac.jp
http://www.res.kutc.kansai-u.ac.jp/~murata/

Abstract. In this paper, we propose Genetic Network Programming (GNP) Architecture using Automatically Defined Groups. GNP is a kind of new evolutionary method inspired from Genetic Programming (GP). While GP has a tree architecture, GNP has a network architecture, with which an agent works in the virtual world. Because only one network architecture is evolved for agents in a system in previous works, every agent takes actions in the same way. In this paper, we apply a coevolution model called Automatically Defined Groups (ADG) to an evolutionary process of GNP, so that several GNP architectures are evolved in order to develop a cooperation among multiple agents. By computer simulation, we show that multi-agent cooperation can be developed by our GNP architecture with the ADG model.

1 Introduction

Recently many researchers have investigated on automatic design of complex systems using evolutionary computation (EC) techniques. Genetic Programming [1] is one of well-known EC techniques that uses a tree structure to describe solutions of numeral problems, combination optimization problems, and action control problems for agents. However, GP is known to have difficulty to converge in one near-optimal solution or to search an optimal structure efficiently because it has a large solution space. When it is applied to dynamic problems such as action control problems, some actions should be selected in a chain of actions. That is, an action should be selected not only by the current situation but also the actions already taken. A designer of GP can define a node for previous information, but it is difficult to define the number of previous actions to be considered to take an action.

In order to cope with such problems, a new architecture called Genetic Network Programming (GNP) has been proposed [2]. It is inspired from GP, but it does not have a tree architecture, but has a network architecture. While Evolutionary

Programming (EP) [3] also has a network architecture, it may be difficult to predefine all transitions in advance. GNP can work with only problem-dependent nodes like GP so that the designer predefine only a small number of nodes.

In GNP, we apply the Michigan approach where each action rule is represented by an individual. That is, each agent takes actions according to a set of action rules developed by GNP. In GNP [2], every agent takes actions using an identical set of rules. It can be called a homogeneous model. In GP research works, however, various methods have been proposed for heterogeneous model [4-6]. Luke and Spector [4] showed that the heterogeneous model performs better than the homogeneous model because the ability of agents becomes higher and agents performs more complex cooperative behaviors, while the search efficiency for GP becomes worse as its searching space increases. In the previous studies [4,5], the number of agents was restricted to two to four.

In order to obtain multiple roles for a number of agents, Hara and Nagao [7,8] have proposed a combined model of the homogeneous and the heterogeneous model. In their model, there are several roles for agents. They develop several trees by GP, and the agents with the same role refer the same tree. They proposed a model called Automatically Defined Groups (ADG) [7]. In their ADG model, the number of roles does not have to be predefined, but it is obtained automatically through evolutionary process. They showed their effectiveness for a load transportation problem [7,8] and the tile world problem [9].

Because the ADG model was proposed mainly for GP, we apply that model to develop networks of GNP in this paper. Using the load transportation problem [7,8], we show that the GNP can obtain better rules than the GP, and the ADG model for the GNP can develop appropriate roles for cooperative multiple agents.

2 GNP: Genetic Network Programming

2.1 Basic Architecture of GNP

In this section, we describe the basic architecture of GNP [2,10,11]. 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 why it is called as “terminal” node. On the other hand, the processing node in GNP can connect other nodes.

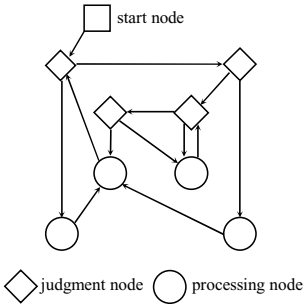

Fig. 1 Basic Structure of GNP.

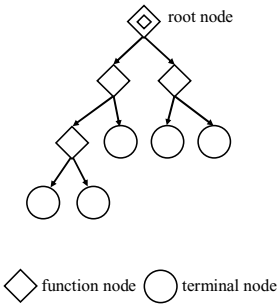

Fig. 2 Basic Structure of GP.

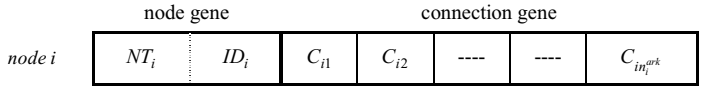

Fig. 3 Representation for the node.

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 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 $i = 1,...,N$ randomly. To the generated N nodes, 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.

2.3 Genetic Operations for GNP

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 for crossover.
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.

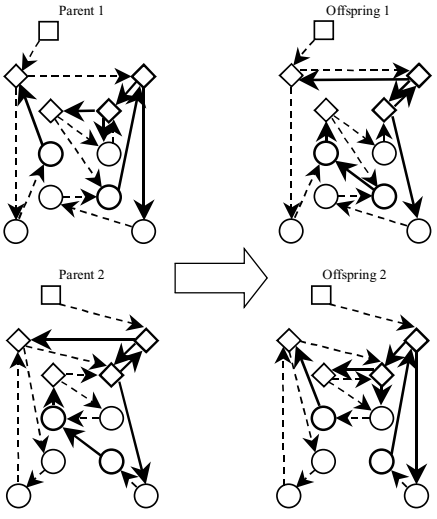

Fig. 4 Crossover in GNP.

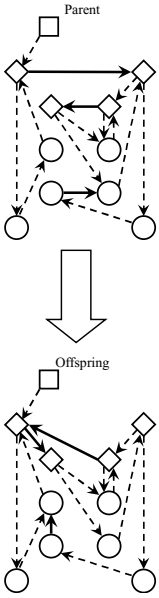

Fig. 5 Mutation for connection gene in GNP.

Fig. 5 shows a mutation operation for connection gene [12]. The procedure of the mutation for the connection gene is as follows:

[Mutation in GNP]
Step 1: Select a network randomly.
Step 2: According to the mutation probability P_m, select a connection.
Step 3: Turn the value of the selected connection to another node ID.

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 GNP in the ADG Model

3.1 ADG: Automatically Defined Groups

We employ the Automatically Defined Groups (ADG) model [7,8] for developing several roles of agents. According to its role, each agent refers to the action control rules described by GNP. In the concept of the ADG model, each agent has its own role, and the agents with the same role take actions under the identical set of rules. This is likened to social insects such as bees and ants. They have several specialized classes

among them such as queens, drones, workers and so on. It is considered that a number of workers take action under the identical rules. While these rules of a class is developed by GP in [7,8], we develop rules for a class by GNP in this paper. In the ADG model, a set of several classes is considered as one individual to be governed by genetic operations. That is, a set of several networks of GNP is considered as an individual. We refer this individual including several networks as an ADG individual in this paper.

In our ADG model, each agent belongs to one of the classes of the network. Therefore each ADG individual has three information: the class structure according to the number of roles of agents, a network structure for each class in the ADG individual, and the class IDs for all agents.

3.2 Genetic Operations for the ADG model

Genetic operations for the ADG model aim to acquire several classes according to the number of roles of agents. There are two tasks to attain this aim:

1) Acquire the number of roles, and develop a network of GNP for each role.
2) Acquire the correspondence between each agent and each role.

In order to attain these tasks, Hara and Nagao [7,8] proposed a group mutation and a crossover for ADG individuals. They proposed a group mutation to change the members of a class (or a group) as follows:

[Group Mutation in ADG]

Step 1: According to the group mutation probability P_{gm}, select an ADG individual.
Step 2: For the selected ADG individual, select an agent, and identify the class (or the group) of the selected agent.
Step 3: Change the class ID of the selected agent randomly.
Step 4: If the class ID randomly specified in Step 3 is the same class ID to which the agent belonged, generate a new network only for the selected agent. The structure of the newly generated network is the same one to which the agent belonged. Then go to Step 5. If the class ID chosen in Step 3 is different, go to Step 5.
Step 5: Return to Step 1 until the mutation is applied to all the ADG individuals.

By this mutation, the class structure is modified before the crossover operation.

After applying the group mutation to each ADG individual, the following crossover is applied to the population of the ADG individuals.

[Crossover in ADG]

Step 1: According to the crossover probability P_{gc}, select two ADG individuals among the ADG population.
Step 2: Select one agent randomly.
Step 3: Identify the network of GNP to which the selected agent refer in each of the selected ADG individuals. Let Net and Net' denote the networks of the

selected ADG individuals.

Step 4: Identify the set of agents $A(Net)$ that refer to the network Net as action control rules. Identify $A(Net')$, too.

Step 5: Apply the crossover operator for GNP (see Subsection 2.2) to the selected networks Net and Net'.

In Step 5, the following three relations between $A(Net)$ and $A(Net')$ are examined and the merger or the division of the classes is implemented before applying the crossover operator.

(Type a) $A(Net) = A(Net')$,
(Type b) $A(Net) \subset A(Net')$ or $A(Net) \supset A(Net')$,
(Type c) Otherwise.

In the case of Type a, the crossover for GNP is applied to Net and Net' without the merger or the division of the classes. If $A(Net)$ and $A(Net')$ is in the relation of Type b, the division of the class is occurred in the larger set. On the other hand, the merger of the classes is taken place when $A(Net)$ and $A(Net')$ are neither the same set nor the inclusion relation. Using Figs. 6 - 8, we explain these three cases.

In Figs. 6 - 8, two ADG individuals are already selected for the crossover operator, and Agent 2 is selected in Step 2 of the crossover operation for the ADG model.

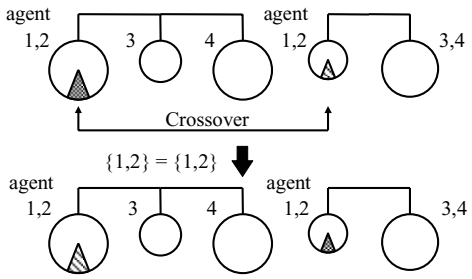

Fig. 6 Crossover for ADG in the case of $A(Net) = A(Net')$ (Type a).

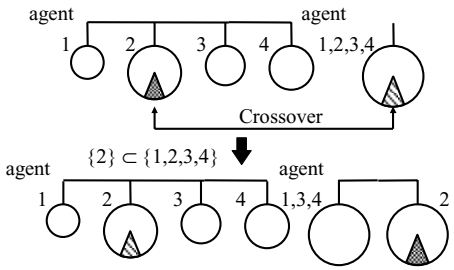

Fig. 7 Crossover for ADG in the case of $A(Net) \subset A(Net')$ or $A(Net) \supset A(Net')$ (Type b).

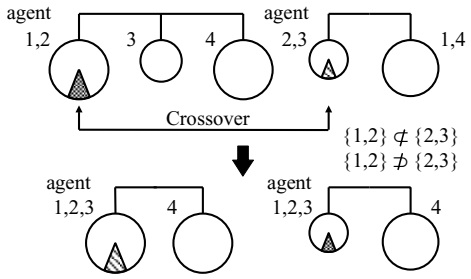

Fig. 8 Crossover for ADG in the case of no inclusion of $A(Net)$ and $A(Net')$ (Type c).

(Type a)
In this case, there is no change among the class structure in an ADG individual since Agent 2 is a member of the set {1, 2} in both the ADG individuals in Fig. 6. Apply the crossover for GNP in Subsection 2.3 to the selected networks.

(Type b)
When Agent 2 is selected in Step 2, it is a member of the set {2} in the left parent, and a member of the set {1, 2, 3, 4} in the right one in Fig. 7. If the network for the set {1, 2, 3, 4} in the right parent is modified with the network in the left by the crossover, Agents 1, 3, 4 are also influenced by the modification. In order to restrict the influence of the network for Agent 2 in the left parent, the same network of {1, 2, 3, 4} is newly generated in the right and Agent 2 is assigned to the newly generated network. After that, apply the crossover to the network of the left parent and the newly generated network in the right parent.

(Type c)
In this case, Agent 2 is included in {1, 2} in the left parent, and in {2, 3} in the right one. Since these two sets are not included each other, the union of these two sets is assigned to the networks generated by the crossover as shown in Fig. 8.

3.3 Overall Algorithm of GNP with the ADG Model

Using the ADG model, we develop networks of GNP by the following algorithm:

[Algorithm for GNP with the ADG Model]
Step 1 (Initialization)
Initialize the population with N_{pop}^{ADG} ADG individuals.
Step 2 (Fitness evaluation)
Calculate the fitness value of each ADG individual, and find an elitist individual with the best fitness value in the population.
Step 3 (Genetic operations)
Step 3-1 (Group Mutation): Select $N_{pop}^{ADG}-1$ ADG individuals by the tournament selection, and apply the group mutation to ADG individuals

according to the group mutation probability.

Step 3-2 (Crossover): Apply the crossover operator in Subsection 3.3 to the selected parents.

Step 3-3 (Mutation): Apply the mutation operator for GNP 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 ADG 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.

4 Computer Simulations on Load Transportation Problem

In this chapter, we employ a simple load transportation problem [7,8] to show the effectiveness of GNP comparing to GP. Later, the effectiveness of the ADG model is shown by comparing it with the heterogeneous model using the same problem.

4.1 Load Transportation Problem

In the load transportation problem, there are two types of loads that differ in weight. Each of the two types loads is placed in a point in a two-dimensional grid world. That is, the heavy loads are placed in the point H in the environment, and the light ones are placed in the point L. The number of agents is 20, and there are only 5 agents that can bring either load among them. The other 15 agents can carry only a light load.

The aim of this problem is to carry as many loads as possible to the goal point in a constant time period. The fitness of a team of 20 agents is measured by how many loads are transported to the goal point within the allotted time. The score of a heavy load is 5 and that of light one is 1. We allow 100 time steps for each agent. During that time steps each agent can bring a load back to the goal point three times if it moves in a shorter way. Thus $120 = (5 \times 5 + 1 \times 15) \times 3$ is the best fitness of a team.

4.2 Comparison between GNP and GP

In this section, we compare the performance of GNP and GP using the load transportation problem. We employed the homogeneous model. That is, the algorithm in Subsection 2.4 is used for GNP. In order to compare the performance of GNP with GP, we employ the same judgment (function) and processing (terminal) nodes in both the algorithms. Table 1 shows two judgment nodes and four processing nodes.

Table 1 Nodes used for GNP and GP.

Name	Description
if_carrying_load	Carry load or not
if_load_here	There is a load or not
Pick_up	Pick up load at the current position
Move_goal	Move to the goal point
Move_heavy_load	Move to the heavy load
Move_light_load	Move to the light load

Table 2 Average results of GNP and GP.

Fitness	GNP	GP
Max	120	75
Average	105.8	75.0
Min	75	75

When we generate the initial population for GNP, we used five nodes for each node type. Therefore each network has 30 nodes with a start node. In order to use the same number of nodes in GP, we specified the maximum depth of the tree was five.

We applied the genetic operations with 500 generations in each trial of GNP and GP. Table 2 shows the average result of 100 trials. According to the results, we can see that GP could not find the tree structure that enables agents to get the highest fitness value while GNP could obtain the network to attain the maximum fitness value. Figs 9 and 10 show the tree and the network structure obtained by GP and GNP, respectively. In these figures, the nodes that have no effect in the decision making of an agent are omitted. From Fig. 9, we can see that GP could produce the rule tree which enables only five agents to carry heavy loads. On the other hand, GNP could produce the network if an agent can not carry the heavy load, it will move to the place with light loads. This result clearly shows the effectiveness of GNP.

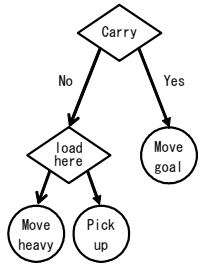

Fig. 9 The best tree obtained by GP

Table 3 Average results of the ADG and the heterogeneous model.

Fitness	ADG	Hetero
Max	120	118
Average	120.0	110.9
Min	120	99

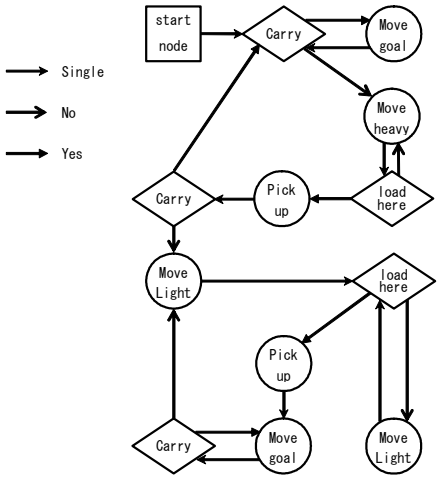

Fig. 10 The best network obtained by GNP

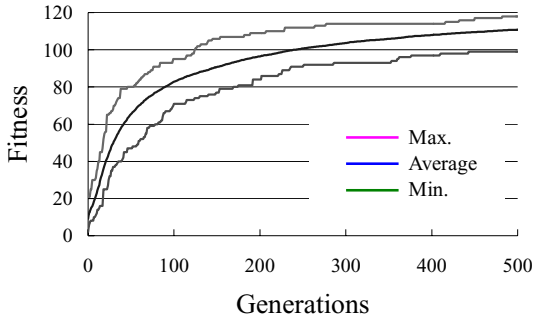

Fig. 11 Average over 100 trials by the Heterogeneous model.

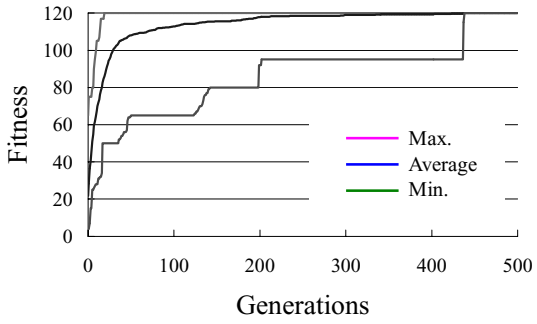

Fig. 12 Average over 100 trials by the ADG model.

4.3 Effectiveness of GNP Using the ADG Model

In this section, we compare the performance of GNP using the ADG model with that of GNP using the heterogeneous model. In the heterogeneous model, every agent has its own network for its action control rule.

Table 3 shows the average result of 100 trials of GNP using the ADG model and the heterogeneous model. Table 3 shows that GNP using the ADG model obtained the highest fitness value in every trial. On the other hand, GNP using the heterogeneous model could not find the network with the highest fitness value. These results clearly show that the ADG model is effective to this problem.

Figs. 11 and 12 show the average fitness over 500 generations obtained by the heterogeneous model and the ADG model. From these figures, we can see that the ADG model can find the best fitness (120) in the 20th generation in the best trial. On

the other hand, the heterogeneous model could not find the network with the best fitness value in 100 trials. From these figures, we can see the effectiveness of GNP with the ADG model to evolve the cooperation of multiple agents.

5 Conclusion and Future Works

In this paper, we propose the GNP architecture with the ADG model. By computer simulations, we clearly show the better representation ability of GNP than that of GP. We also showed that GNP with the ADG model can find appropriate roles of agents according to their ability by the load transportation problem.

As for further research topics, we need to investigate a way to reduce the redundant nodes in the networks developed in GNP. In Fig. 10, we showed the network where the nodes that are not work are removed. Without such a manipulative remove, the network has a more redundant structure. We will improve the performance of GNP in such way.

References

[1] J. R. Koza, *Genetic Programming On the Programming of Computers by means of Natural Selection*, MIT Press, 1992.

[2] 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.

[3] L. J. Fogel, A. J. Owens and M. J. Walsh, *Artificial Intelligence through Simulated Evolution*, Wiley, 1996.

[4] S. Luke and L. Spector, "Evolving teamwork and coordination with genetic programming," *Genetic Programming 1996*, pp. 141-149, 1996.

[5] H. Iba, "Multiple agent learning for a robot navigation task by genetic programming," *Genetic Programming 1997*, pp. 195-200, 1997.

[6] T. Haynes and S. Sen, "Crossover operation for evolving a team," *Genetic Programming 1997*, pp. 162-167, 1997.

[7] A. Hara and T. Nagao, "ADG: Automatically Defined Groups for multi-agent cooperation," *Proc. of 2nd Japan-Australia Joint Workshop on Intelligent and Evolutionary Systems*, pp. 91-98, 1998.

[8] A Hara and T. Nagao, "Emergence of cooperative behavior using ADG; Automatically defined groups," *Proc. of 1999 GECCO,* pp. 1039-1046 (1999).

[9] M.E. Pollack and M. Ringuette, "Introducing the tile world: Experimentally evaluating agent architectures," *Proc. of 8th National Conf. on Artificial Intelligence*, pp. 183-189, 1990.

[10] K. Hirasawa, M. Okubo, H. Katagiri, J. Hu and J. Murata, "Comparison between genetic network programming and genetic programming using evolution of ant's behaviours," *Trans. on Institute of Electrical Engineers of Japan*, Vol. 121-C, No. 6, pp. 1001-1009, 2001 (in Japanese).

[11] H. Katagiri, K. Hirasawa, J. Hu and J. Murata, "Variable size genetic network programming," *Trans. on Institute of Electrical Engineers of Japan*, Vol. 123, No. 1, pp. 57-66, 2003 (in Japanese).