Emergence of cooperative behavior using ADG; Automatically Defined Groups

Akira Hara **Tomoharu Nagao**
Imaging Science and Engineering Laboratory
Tokyo Institute of Technology
4259, Nagatsuta-cho, Midori-ku, Yokohama, 226-8503 Japan
ahara@isl.titech.ac.jp nagao@isl.titech.ac.jp

Abstract

In producing a multi-agent team which solves the problem cooperatively by means of Genetic Programming(GP), it seems that a heterogeneous team performs better than a homogeneous team. In a heterogeneous team, however, as the number of agents increases, the size of search space becomes vast and the efficiency of search decreases. One of the solutions of this problem is to divide a team into the proper number of groups, and to provide the same program for the all agents belonging to the same group. However it is difficult to know the adequate team structure beforehand. In order to solve these problems, we have proposed a method called *ADG; Automatically Defined Groups*. In this paper, we applied this method to a simple load transportation problem and a modified Tile World problem, and confirmed that the optimal team structure was acquired in each problem.

1 Introduction

Solving problems by multiple agents have attracted increasing attentions in recent years. A number of attempts that multiple agents having an identical action rule emerge the beneficial higher-level collective behavior have been made in the domain of evolutionary computation. In order to deal with difficult tasks, however, the distinct group behavior and the division of labor in the team are needed. In producing a multi-agent team, which solves the problem cooperatively, by means of Genetic Programming(GP), the breeding strategy is important. The main topic developed in this paper is automatic acquisition of optimal team structure using GP. We have proposed a new breeding strategy of GP, ADG; Automatically Defined Groups (Hara and Nagao, 1998). This strategy can exhibit the emergence of cooperative behavior for a large number of agents. In this section, we briefly present researches in the field of multi-agent. In section 2, we describe our breeding strategy, ADG. Next, in section 3, we apply the proposed method to a simple transportation problem and show several experimental results. Moreover, in section 4, we apply ADG to a modified Tile World problem. Finally, in section 5, we describe conclusions and future work.

1.1 Multi-Agent Cooperation

The methods for acquisition of action control rules of multi-agent by GP are mainly classified into two common ways. The first is to use co-evolution: Each agent is controlled by an individual genetic program. We perform evaluation of the individual based on his actions, and improve performance of the team that consists of respective agents. However it causes the credit assignment problem. Another way to generate a team is to regard a team that consists of all agents as an individual of GP. We perform evaluation of the individual based on the performance of the team. This way facilitates the evaluation and can produce the complex behavior such as the division of labor, since all agents are evaluated equally by the team's performance regardless of respective roles. Especially, when we consider the whole agents as one GP individual, various models (homogeneous model, heterogeneous model, etc.) have been proposed, and experiments for the comparison of the models have been performed (Luke and Spector, 1996; Iba, 1996).

1.1.1 Homogeneous Model

When all agents in the environment take actions under the identical rules, this team is called the *homogeneous* team. In GP, each agent refers to the same tree. All

agents decide their movements according to the same rules derived from the GP tree. However, since each agent is situated in a different environment, it is possible that each agent takes different actions according to the conditions and solves the problem by cooperation with each other.

1.1.2 Heterogeneous Model

When each agent in the environment takes actions under the distinct rules, this team is called the *heterogeneous* team. In GP, an individual maintains multiple branches, each of which are referred to by the corresponding agent. In the heterogeneous model, the various breeding strategies (Restricted Breeding, Free Breeding and so on) have been proposed (Luke and Spector, 1996; Iba, 1996; Haynes and Sen, 1997). Free breeding allows any member of a team to freely breed with any other member of another team. In restricted breeding, crossover operations are restricted to corresponding branch pairs. For instance, restricted breeding allows team member 1 to breed only with another team member 1, and team member 2 to breed only with another team member 2, etc. Generally restricted breeding works better than free breeding from a viewpoint that the restriction further promotes preservation of diversity and specialization of each agent since it divides team members in separate breeding pools (Luke and Spector, 1996).

2 ADG; Automatically Defined Groups

It is considered that each agent in heterogeneous model is specialized according to the each role in the environment and the team can perform more complex cooperative behavior than the homogeneous team. However an individual in GP needs to maintain multiple branches as many as the number of agents. Therefore the search space size becomes vast as the number of agents increases. It causes a decline of search efficiency.

By the way, bee and ant colonies in nature are regarded as large-scale multi-agent systems. Social insects such as bees and ants are divided into several classes according to the roles for their survival. For example, bee colonies consist of several specialized classes (queens, drones, workers, etc.). Each class consists of many hundreds of identical individuals. It is considered that a number of workers take action under the identical rules. In case we regard this colony as a GP individual, the structure of an individual becomes a hybrid structure between homogeneous and heterogeneous model as shown in Figure 1. If we apply this structure to GP, the system can prevent the increase of search space and search efficiently. That is to say, one of the solutions to previous problems is to divide a team into the proper number of groups, and to provide the same program for the whole agents belonging to the same group (Masunaga and Nagao, 1997). However it is difficult to know the proper team structure beforehand. Therefore we propose a method that the proper team structure is acquired automatically in the process of evolution. By this method the system can prevent the increase of search space and the search works efficiently. On the other hand, the acquired group structure becomes a clue for understanding the cooperative behavior and the necessary division of labor for solving the problem.

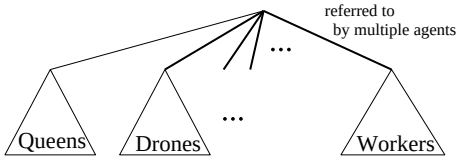

Figure 1: Classes of bee colony.

2.1 Proposed Breeding Strategy

We define a group as the set of agents referring to the same tree for the determination of their actions. The proposed strategies are as follows:

A GP individual maintains multiple branches, each of which functions as a specialized program for a distinct group. All agents belonging to the same group use the same program.

- Generating the initial population

  We generate the initial population consisting of GP individuals. Agents in each individual are divided into the random groups.

- Crossover operations

  Basically, crossover operations are restricted to corresponding branch pairs. For example, a tree referred to by an agent 1 in a team breeds with a tree referred to by an agent 1 in another team. The concrete processes are as follows.

1. We choose an agent arbitrarily to two parental individuals. A tree referred to by the agent in each individual is used for crossover. We use T and T' as expressions of these trees, respectively.

2. In each parental individual, we decide a set $A(T)$, the set of agents which refer to the selected tree T.

3. We arbitrarily choose a subtree for crossover in each selected tree, and perform crossover. When we perform a crossover operation on the tree T and T', there are the following three cases.

   (Type a) If the relation of the sets is $A(T) = A(T')$, the structure of each individual is unchanged.

   For example, Figure 2 shows the case that the trees referred to by the agent 2 are used for crossover. In this case, the sets of agents that refer to the trees are equal. So we perform a crossover operation as the restricted breeding.

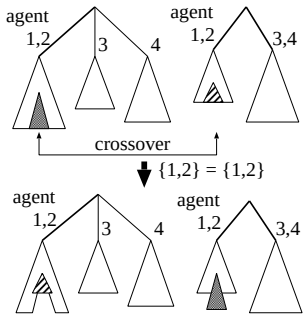

Figure 2: An example of crossover(Type a).

(Type b) If the relation of the sets is $A(T) \supset A(T')$, the division of groups takes place in the individual,which maintains T, so that the only tree referred to by the agents in $A(T) \cap A(T')$ can be used for crossover. The individual which maintains T' is unchanged.

For instance, Figure 3 shows the case that the trees referred to by the agent 2 are used for crossover. When we examine the sets of agents which refer to the trees, the set for the first parent is $\{2\}$ and the set for the second parent is $\{1,2,3,4\}$. The set for the second parent includes the set for the first parent. The subtree used for crossover in the first parent is considered as the special program to the agent 2. If we perform the crossover operation simply, this operation affects not only the program of the agent 2 but also the program of agents 1,3,4 in the second parent. It is considered that such influence is unfavorable. Therefore in the second parent we generate a new tree which is the same tree as agent 2 refers to, and shift agent 2 to the new group referring to the new tree. As a result the division of groups takes place so that the only tree referred to by the agent 2 can be used for crossover, and then we perform a crossover operation.

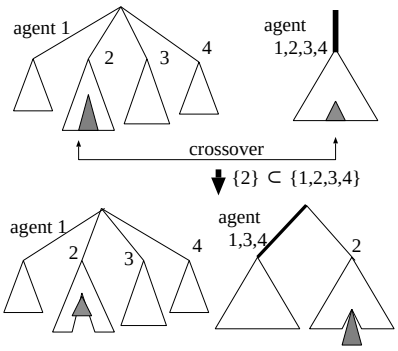

Figure 3: An example of crossover(Type b).

(Type c) If the relation of the sets is $A(T) \not\supset A(T')$ and $A(T) \not\subset A(T')$, the unification of groups takes place in both individuals so that the agents in $A(T) \cup A(T')$ can refer to an identical tree.

For example, Figure 4 shows the case that the trees referred to by the agent 1 are used for crossover. When we examine the sets of agents which refer to the trees, the set for the first parent is $\{1,2\}$ and the set for the second parent is $\{1,3\}$. One set does not contain another set each other. The agents 1 and 2 have the same action control rules in the first parent, and the agents 1 and 3 have the same action control rules in the second parent. We consider that the agents 1,2,3, which are elements of the union of these sets, had better have the same action control rules in offspring of the parents. Therefore in both parents we perform the unification of groups so that the agents 1,2,3 can refer to the same tree, and then we perform a crossover operation.

- Group mutation operations

  As mentioned above, the difference between the sets of agents referring to the trees used for crossover causes the division or unification of groups. Therefore, if all individuals converge to the same group structure, the changes of group structure do not take place. So, group mutation operations are introduced into this method. The group mutation operation shifts an agent to the arbitrarily selected group, with a probability equal to the group mutation rate. If the selected group is the same as the current group,

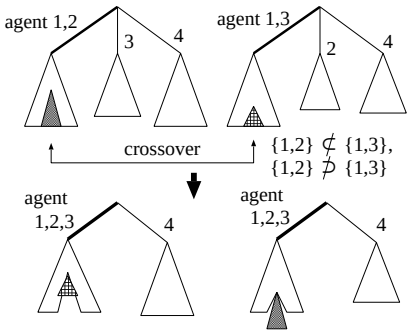

Figure 4: An example of crossover(Type c).

an identical tree is generated and the agent is shifted to the new group referring to the new tree. This operation works as an accelerator for the change of group structure by crossover operations. So, we perform group mutation operations before crossover operations.

- Selection

  We use tournament selection. In selection, if several individuals have the same fitness, the individual with lower number of groups survives. This is because this condition prevents the redundant division of groups. For example, consider the case where whole trees are identical in a heterogeneous team. The fitness of the team might be the same as a homogeneous team's. In this case, we consider that the trees in this heterogeneous team are redundant. Therefore we choose the individual with lower number of groups when several individuals have the same fitness.

We expect that by using this method the search works efficiently and the adequate team structure is acquired.

3 Load Transportation Problem

In order to confirm the emergence of the division of labor among the agents, we defined a simple load transportation problem. In this section, we apply the proposed method to this problem and show several experimental results.

The defined environments are as follows.

- Two types of loads that differ in weight are placed in the environment. We define two points where loads are placed. In one point, only light loads are placed. While in the other point, only heavy loads are placed. We assumed that large quantities of loads are placed in each point and that there is no case where loads run out.

- Multiple agents that differ in ability to carry are placed. The number of agents is 20. The agents 0,4,8,12,16 can carry either load, and the rest of agents can carry only light load. On each time step of the simulation, each agent evaluates its own GP tree to determine the action.

- A single goal point where loads should be gathered is set. The distances from the goal to respective load points are the same.

- 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 is measured by how many loads are transported to the goal point within the allotted time, *Max_Step*. When an agent arrives at the goal point with load, the load is automatically released and counted. The score on a heavy load is higher than that on a light load.

- The domain of action is an 11×10 grid. The goal point is placed at the grid location $(5, 0)$. The light load point is at $(0, 9)$, and the heavy load point is at $(10, 9)$. All agents start from the goal point. The movement of each agent is orthogonal. All objects are allowed to occupy the same grid.

Table 1: GP Functions and Terminals.

Name	Description
`If_carrying_load( , )`	carrying load?
`If_load_here( , )`	is there any loads here?
`Pick_up`	pick up a 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: GP Parameters.

population size	1000
selection	tournament + elitist strategy
tournament size	5
crossover rate	0.9
mutation rate (a rate of individuals which mutation is applied to)	0.9
group mutation rate	0.01
Max_Step	100
score for a heavy load	5
score for a light load	1

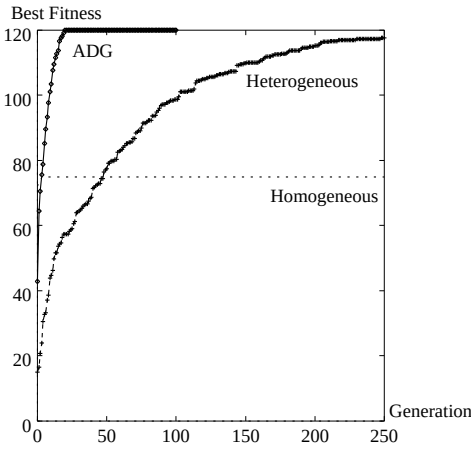

Figure 5: Comparison of the search performance between three methods.

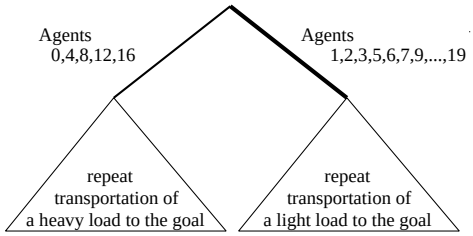

Figure 6: Acquired tree structure.

In order to apply GP to evolving agents' programs in this problem, we use the terminal and nonterminal sets shown in Table 1. The parameters about GP and environment are shown in Table 2. In mutation of tree nodes, individuals are selected with a probability of a mutation rate. The mutation operation is performed at only one random point in each selected individual.

We performed experiments with a proposed method under the environment mentioned above. Figure 5 shows the result of the experiments. The figure plots the best fitness value with generations averaged over 10 runs. We can see the best fitness was acquired at the generation about 20 by ADG.

The best individuals in respective runs had the same group structure. Figure 6 shows acquired tree structure by means of this proposed method. The tree was divided into the two groups. One group consisted of agents 0,4,8,12,16, which were defined so that they were also able to carry the heavy load. The other group consisted of the rest of agents, which were able to carry only a light load. Moreover the roles were separated so that the agents were able to carry the load according to their ability. Thus, it is easy to understand a form of cooperation by seeing the acquired tree structure. This is one of the advantages of ADG.

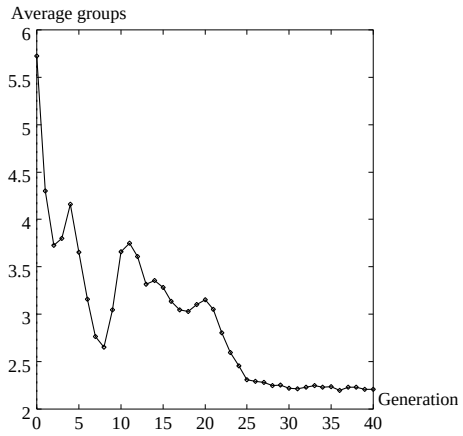

Figure 7: The change of the average number of groups.

The changes of the average number of groups in the whole individuals every generation in a certain run are shown in the Figure 7. Although each individual is assigned the random group in the initial population, we see from this graph that the individuals search solutions as their numbers of groups approach the optimal number of groups (= 2) gradually.

In order to confirm the effectiveness of the proposed method, we also ran the same experiments with conventional models, heterogeneous model and homogeneous model. For the sake of comparison, the curves for the homogeneous and heterogeneous models are also shown in Figure 5. In case of the heterogeneous model, optimum action of agents could be acquired. However, the time, which was needed for evolution of heterogeneous teams, was considerably longer than the time, which was needed by ADG. This difference results from the increase of the search space. Because in the heterogeneous model we have to optimize twenty tree structures as many as agents in the environment. Furthermore, in the heterogeneous model, it is difficult to know the necessary division of labor for solving the problem by the acquired tree structure. In case of the homogeneous model, the search space was small and as a result the fitness converged in a few generations. However, the optimum action of agents could not be acquired. This failure results from the lack of cooperation. Because in the homogeneous model the agents, which differ in ability, act on an identical rule and each agent can not work according to its ability.

Moreover, in order to confirm the effectiveness of the group operations by crossover in ADG, we also ran the

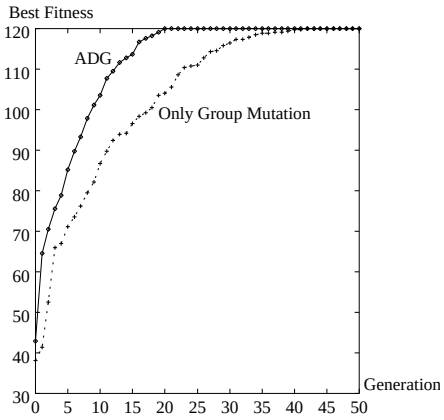

Figure 8: ADG vs. Only Group Mutation.

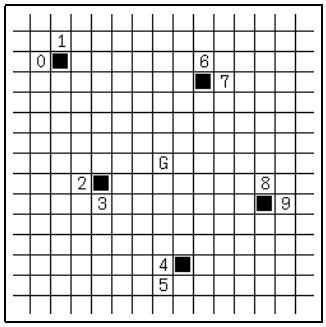

Figure 9: The environment (initial position) for the Tile World.($0, \ldots, 9$:Agent, ■ :Tile, G:Goal)

experiment with two methods, ADG and only group mutation. In the latter method, the change of the group structure is caused only by group mutation. In crossover operation, the group structure of each individual is unchanged though the subtrees are exchanged. In this experiment, the parameters shown in Table 2 are used in both methods. Figure 8 shows the result of this experiment. The figure plots the best fitness value with generations averaged over 10 runs. We can see from this graph that the performance of the ADG is superior to the only group mutation method. Moreover, in the only group mutation method, the optimal group structure as shown in Figure 6 was acquired only once in 10 runs, and the structures with the redundant division of groups were acquired in the other runs. Thus it is found that the division or unification of groups by crossover in ADG is effective.

4 Tile World Problem

In section 3, we showed that ADG worked efficiently due to preventing the increase of the search space and that the acquired group structure becomes a clue for understanding the necessary division of labor. In this section, we examine how the group structure affects the team's performance.

The Tile World (Pollack and Ringuette, 1990) is well known as a multi-agent test bed. We consider a modified version of the Tile World. The world is a toroidal, 2-dimensional landscape. The world consists of a rectangular grid on which agents, some tiles, and a goal are placed. Each tile occupies one cell of the grid. However, multiple agents are allowed to occupy the same cell in this problem. The agent can move up, down, left, and right. When a tile is in a cell adjacent to the agent, the agent can push the tile by moving in its direction. The most important part of this problem is that the tile is too heavy to be transported by a single agent. Each agent can apply a force vector with a size of 1 to the tile by pushing it. If the magnitude of the resultant vector to the tile is bigger than a constant value, *Power_Radius*, agents can carry the tile to the direction of the vector. That is to say, in order to move a tile, multiple agents must push the tile to the same direction at the same time. The aim of this problem is to push all tiles to the goal point in a constant time period.

In order to apply GP to evolving agents' programs in this problem, we use the terminal and nonterminal sets shown in Table 3. The usage of these symbols is motivated by the studies reported by (Luke and Spector, 1996; Iba, 1996). For each step, each agent evaluates its own GP tree in order to determine the action to be taken. GP tree always returns a vector. The vector is transformed into the action. The actions are STAY or move RIGHT, UP, LEFT or DOWN. The mapping between vectors and actions is determined as follows: If the magnitude of the vector $\vec{v}$ is less than or equal to the parameter *Radius*, the action of the agent is STAY. Otherwise, move 1 step RIGHT, UP, LEFT or DOWN according to the direction of $\vec{v}$.

The fitness of a team is measured by how efficiently the task can be solved within the allotted time, *Max_Step*. So we have to take into account 3 factors. The first is *ft*: the number of tiles pushed into the goal after execution. The second is t_F: the number of time steps required pushing all tiles into the goal, and the third is the distances to the goal of the tiles left when the task could not be completed. If the task is not completed, then t_F is equal to *Max_Step*. These three factors are weighted by the parameters *Bonus*, *Speed*, and *Dist*, respectively. Thus, the fitness f is given as follows:

$$f = Bonus \times ft + Speed \times (Max_Step - t_F) + Dist$$

Table 3: GP Functions and Terminals for the Tile World.

Name	#Args.	Description
+	2	Add two vectors.
−	2	Subtract two vectors.
∗2	1	Multiply the magnitude of a vector by 2.
/2	1	Divide a vector by 2.
−>90	1	Rotate a vector clockwise 90 degrees.
`inv`	1	Invert a vector, i.e. if the input is $\vec{v}$, then return $-\vec{v}$.
`ifdot`	4	For the arguments, $\vec{x_1}, \vec{x_2}, \vec{x_3}, \vec{x_4}$, if $(\vec{x_1} \cdot \vec{x_2} \geq 0)$ return $(\vec{x_3})$ else return $(\vec{x_4})$
`if>=`	4	For the arguments, $\vec{x_1}, \vec{x_2}, \vec{x_3}, \vec{x_4}$, if $(\|\vec{x_1}\| \geq \|\vec{x_2}\|)$ return$(\vec{x_3})$ else return$(\vec{x_4})$
`Last`	0	The last output vector of the GP tree for the agent. If this is the first move, then returns a zero vector.
`Goal`	0	The vector from the agent to the goal.
`Nearest-Tile`	0	The vector from the agent to the nearest tile.
`Same-Group`	0	The vector from the agent to the nearest agent which belongs to the same group. (except the agents which are in the same position.)

Table 4: GP Parameters for Tile World.

population size	1000
selection	tournament + elitist strategy
tournament size	5
crossover rate	0.8
mutation rate	0.9
group mutation rate	0.075
Radius	1.0
Power_Radius	3.0
Max_Step	200
Bonus	2500
Speed	80
Dist	100

$$\times \sum_{t \in LT} \{MD(og(t),\ Goal) - MD(cr(t),\ Goal)\}, \quad (1)$$

where LT is the set of remained tiles. $MD(x,y)$ means the manhatttan distance of x and y. $og(t), cr(t)$ are the original position, and the current position. Thus, $\{MD(og(t),\ Goal) - MD(cr(t),\ Goal)\}$ equals to the movement of a tile t toward the goal. This is summed over all remained tiles. For these definitions of the fitness and the mapping from vectors to actions, we refer to the study reported by (Iba, 1996).

In our experiments, as shown in Figure 9, ten agents and five tiles are placed and a specific location is designated as the goal. In order to carry a tile, three agents have to push the tile to the same direction at the same time.

We performed experiments with ADG under the environment mentioned above. The Parameters about GP are shown in Table 4. Figure 10 shows acquired tree structure by means of ADG. We can see from this figure that the tree was divided into the three groups and each group contains three agents in the finally evolved program. Figure 11 shows a snapshot using the finally evolved program. When we observe the movement of agents using this program, agents which belong to the same group assemble and then move in herd in order to push tiles to the goal.

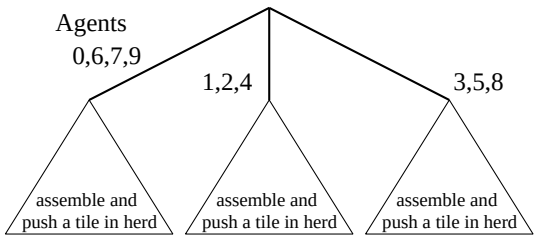

Figure 10: Acquired tree structure.

In order to confirm the effectiveness of ADG, we also ran the same experiments with conventional models, homogeneous model and heterogeneous model. We consider the homogeneous team as the team in which whole agents belong to the same group. On the other hand, we consider the heterogeneous team as the team in which each agent belongs to the distinct group. In the experiment with heterogeneous model, for the sake of the fair comparison, we use `Different-Group` instead of `Same-Group`. `Different-Group` is the vector from the agent to the nearest agent (except the agents which are in the same position) which belongs to the distinct group.

Figure 12 shows the result of the experiments. In ho-

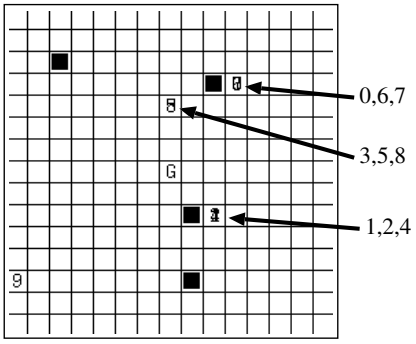

Figure 11: A snapshot of behavior of agents controlled by an evolved program.

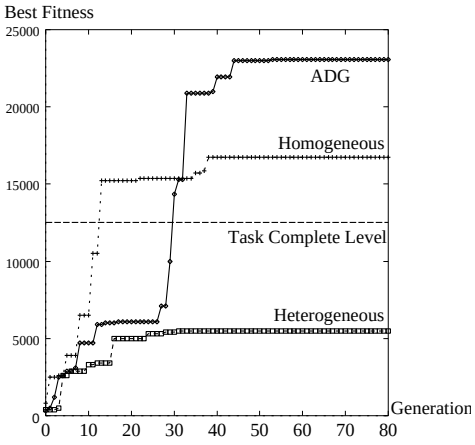

Figure 12: Comparison of the search performance between three methods for the Tile World.

mogeneous model, the behavior that agents assemble and transport tiles to the goal by using `Same-Group` could be acquired. However, the team could not realize the parallel processing by the independent multiple groups. Consequently, compared with the team acquired by ADG, it took more time steps to complete the task, i.e. to push all tiles into the goal. Therefore, the performance of the homogeneous team is inferior to the team acquired by ADG. In case of the heterogeneous model, the team which completes the task could not be acquired. This failure results from the increase of the search space. Because in the heterogeneous model we have to optimize multiple trees as many as agents in the environment.

Consequently, it was found through this experiment that in order to solve the problem efficiently, we need to divide the team into the adequate groups and that can be realized by ADG.

5 Conclusions and Future Work

We presented a method to acquire the optimal team structure in the process of evolution by means of GP. This method employs the restricted breeding for promoting the emergence of cooperative behavior such as the division of labor among the agents, and it optimizes the team structure by dividing or unifying the groups according to the relation of the sets of agents.

We applied this method to multi-agent problems, and confirmed that the optimal team structure in each problem was acquired by our method. We also compared the proposed method with the conventional methods, homogeneous and heterogeneous methods, in the evolutionary speed and performance. As a result we showed that the proposed method worked efficiently due to preventing the increase of the search space and due to forming the adequate team structure.

We are now planning to apply this method to the problems which require more large-scale and more complex cooperation of multiple agents, and confirm whether the optimal team structure can be acquired.

References

Hara,A. and Nagao,T. (1998). ADG; Automatically Defined Groups for multi-agent cooperation. In Katai,O. et al (eds.), *Proceedings of The Second Japan-Australia Joint Workshop on Intelligent and Evolutionary Systems*, pp.91-98.

Luke,S. and L.Spector (1996). Evolving Teamwork and Coordination with Genetic Programming. In J.R.Koza et al (eds.), *Genetic Programming 1996*, pp.141–149, Cambridge MA:MIT Press.

Iba,H. (1996). Emergent Cooperation for Multiple Agents Using Genetic Programming. In H.-M.Voigt et al (eds.), *Parallel Problem Solving from Nature IV*, pp.32–41, Berlin:Springer.

Haynes,T. and Sen,S. (1997). Crossover Operation for Evolving A Team. In J.R.Koza et al (eds.), *Genetic Programming 1997*, pp.162–167, San Francisco, CA:Morgan Kaufmann.

Masunaga,S. and Nagao,T. (1997). An Action Control Model of Agents Based on Evaluation of Group Behaviors. In *1997 IEEE International Conference on Systems,Man,and Cybernetics*, pp.3343–3348, IEEE Press.

Pollack,M.E. and Ringuette,M. (1990). Introducing the Tileworld: Experimentally Evaluating Agent Architectures. In *Proceedings of the 8th National Conference on Artificial Intelligence*, pp.183–189, MIT Press.