ELSEVIER Journal of Information Sciences 108 (1998) 181–205

INFORMATION SCIENCES
AN INTERNATIONAL JOURNAL

Evolutionary learning of communicating agents

Hitoshi Iba [1]

Machine Inference Section, Electrotechnical Laboratory (ETL), 1-1-4 Umezono, Tsukuba Science City, Ibaraki 305, Japan

Received 1 February 1997; received in revised form 1 September 1997; accepted 26 October 1997

Abstract

This paper presents the emergence of the cooperative behavior for communicating agents by means of Genetic Programming (GP). Our experimental domains are the pursuit game and the robot navigation task. We conduct experiments with the evolution of the communicating agents and show the effectiveness of the emergent communication in terms of the robustness of generated GP programs. The performance of GP-based multi-agent learning is discussed with comparative experiments by using different breeding strategies, i.e., homogenous breeding and heterogeneous breeding. © 1998 Elsevier Science Inc. All rights reserved.

Keywords: Genetic programming; Multi-agent system; Distributed artificial intelligence

1. Introduction

Recently intelligent agents and multi-agent systems have garnered much interest in Distributed Artificial Intelligence (DAI). Genetic Programming (GP) and its variants have been applied to multi-agent learning ([1], Ch. 12; [2–8]). They have introduced the following evolutionary strategies:

1. Homogeneous breeding (Fig. 1(a)). All agents use the same program evolved by GP. Individuals breed as in an ordinary GP.

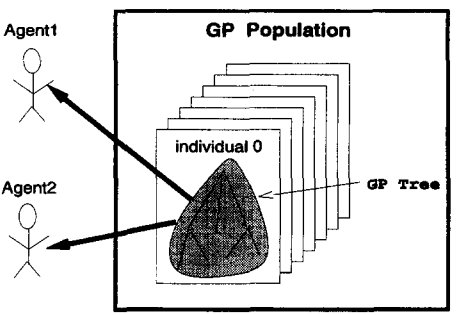
[1] E-mail: iba@etl.go.jp.

PII: S0020-0255(97)10055-X

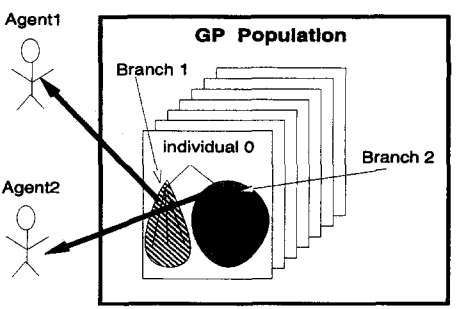
2. Heterogeneous breeding (Fig. 1(b)). Each agent uses a distinct program. A GP individual maintains multiple branches, each of which serves as a specialized program for a distinct agent. The breeding proceeds in the same way as the ADF version of GP [9], i.e., crossover operations are applied to the correspondent branch pairs.

They showed that GP could evolve the cooperative behavior for a specific problem solving. However, in most of these studies, the agents had no communicating mechanism. The communication is an essential factor for the emergence of cooperation. This is because a collaborative agent must be able to handle situations in which conflicts arise and must be capable of negotiating with other agents to reach an agreement [10]. Benda et al. [11] introduced the following three types of relationship between agents:

1. Communicating agents (Type A), i.e., one agent is capable of requesting data from another agent.
2. Negotiating agents (Type B), i.e., in addition to the above data request, agents can negotiate with each other about their movements.
3. Controlling agents (Type C), i.e., an agent can control over another agent.



(a) Homogeneous Strategy.



(b) Heterogeneous Strategy.

Fig. 1. Homo-/heterogeneous strategies.

This paper applies GP for evolving communicating agents and shows that the robustness of the generated program is increased with the communication in the following problem domains:

1. The robot navigation task (Fig. 2), in which communication commands, such as SEND or RECEIVE, are used so that the agent can sig-

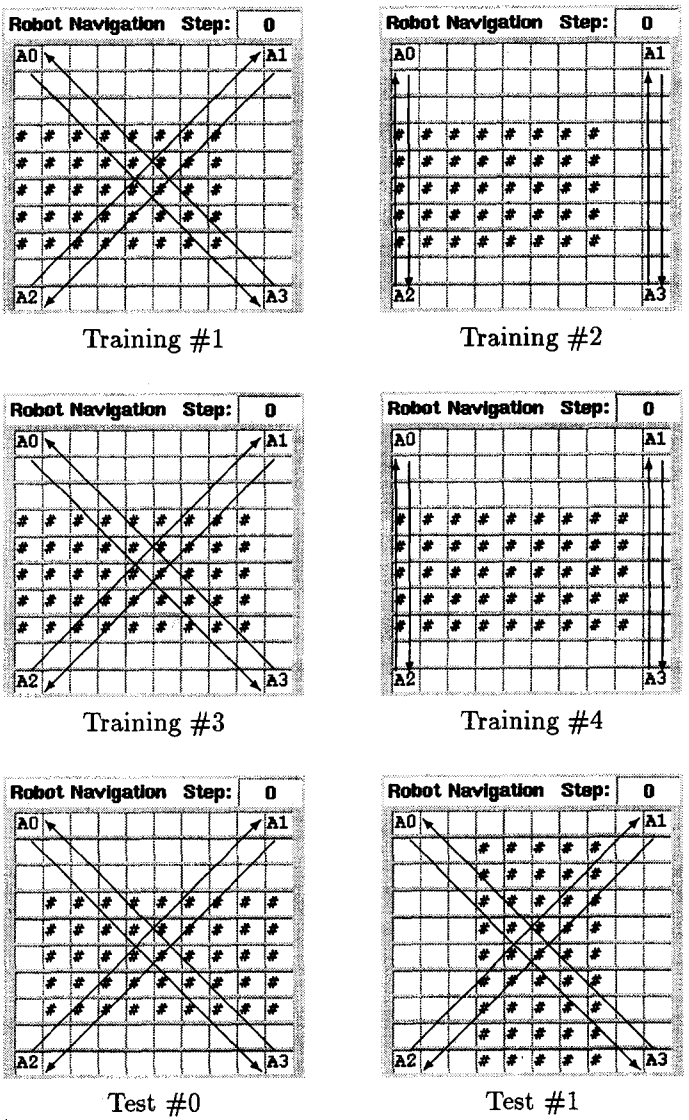


Fig. 2. Training and test cases.

nal what it sees to a remote agent and sometimes can tell it to stop (i.e., Type C).

2. The pursuit game (Fig. 8), in which we introduce COM?'s commands. These commands take one argument and request data from the agent? (i.e., Type A).

The robustness is an important feature of a program evolved by GP [12]. It is defined as the ability to cope with noisy or unknown situations. In the robot navigation, the robustness could be examined by testing an evolved program for another navigation task. In pursuit of the robustness, we verify the validness of an evolved program for testing data, which are different from the training data. We show that GP-based multi-agent learning evolves robust programs through the usage of communication.

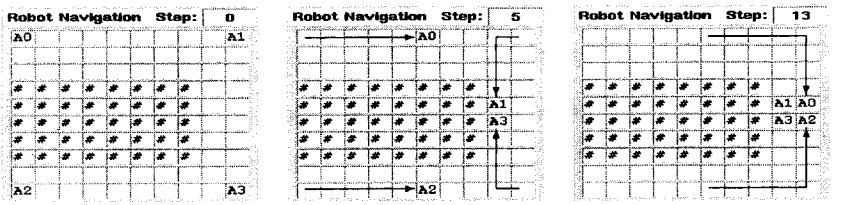
The rest of this paper is structured as follows. GP is applied to a robot navigation task in Section 2. Section 2.1 describes the experimental set-up of the robot navigation. Section 2.2 introduces communication commands for our tasks. In Section 2.3, we show the experimental results and compare the performance. We use the above-mentioned two breeding strategies (Fig. 1) for comparison [2]. Section 3 describes another multi-agent problem, i.e., the pursuit game. Section 3.2 shows some experimental results in order to test the robustness of a generated program. In Section 3.3, we apply our approach to evolving communicating agents. Section 4 discusses these results, followed by some conclusions in Section 5.

2. The robot navigation problem

The world of our robot navigation consists of a rectangular grid on which agents (denoted as Ai, $i = 0, 1, \ldots$) and some obstacles (#) can be placed (see Fig. 2). Each object occupies one cell of the grid. The agent can move up, down, left, and right unless doing so would cause it to run into the world's boundaries or an obstacle. The agents' goal is to find the optimal path in a grid world, from given starting locations to their respective goals (the agents' destinations are shown by arrows in the figure). In our robot navigation task, four agents have to pass through a narrow aisle to reach their goals. The deadlock problem often occurs when two agents try to move toward the opposite directions in the aisle. For instance, if agents move to their destinations without any

[2] We have proposed a new approach to co-evolving the multi-agent cooperation, i.e, the emergence of the job separation among the agents [5,6]. Experimental results have shown that this strategy seems to be promising for GP-based multi-agent learning, as an integration of both homogeneous and heterogeneous strategies. Actually, in the following experiments, we have often used this co-evolutionary breeding, instead of heterogeneous strategy. However, for the sake of the limitation of pages, we only report on the results by the heterogeneous strategy.

Agent0: Goal
Agent1: (if>= Ag1 Goal Goal Goal)
Agent2: Goal
Agent3: (inv (inv Goal))



Step0 Step5 Step13

Fig. 3. Behaviour of agents based on goal terminal.

consideration, then they are very likely to block each other (see Fig. 3 for the example behavior). Hence, but for the necessary cooperation, the navigation task could not be solved effectively.

Fig. 2 shows some of the training and testing cases. These cases are taken from various types of navigation scenarios, and solving these distinct tasks requires the effective generalization of training examples (i.e., the robustness of an evolved problem).

2.1. The robot navigation and genetic programming

In order to apply GP to evolving an agent's problem in the robot navigation, we use the terminal and nonterminal sets shown in Table 1 [3]. In the table, a symbol without any argument is a terminal symbol.

We have chosen vector operations for the GP tree representation. This is aimed at incorporating more precise directional information as to the environment surrounding the agents. We assume agents, i.e., robots, have a potential-based sensor [13]. Thus, the Goal terminal returns a directional vector, through which the agent draws nearer to the goal. The length of the vector is the distance to the goal. For instance, assuming that x and y axes are rightward and upward as usual, the Goal terminal returns a vector $\begin{pmatrix} -9 \\ 9 \end{pmatrix}$ for the agent A3 in Training #1 (Fig. 2), because the agent A3 and its goal are positioned at $\begin{pmatrix} 9 \\ 0 \end{pmatrix}$ and $\begin{pmatrix} 0 \\ 9 \end{pmatrix}$, respectively. In the same way, the Ag*i* terminal returns a vector from the agent to the *i*th nearest agent, through which the agent draws nearer to the *i*th nearest agent.

[3] The usage of these symbols is motivated by the study reported by [4]. Luke studied evolving teamwork by GP for a pursuit game, in which the world is a continuous 2-dimensional.

Table 1
GP terminals and functions (robot navigation)

Name	# Args.	Description
Goal	0	The directional vector by which to move the agent toward its goal.
Last	0	The last vector of the GP output for the agent. If this is the first move, then returns a random vector.
Ag*i*	0	The directional vector by which to move the agent toward the *i*th nearest agent.
V1	0	A unit vector, i.e., $\binom{1}{0}$.
Rand	0	A random vector.
+	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 v, then return $-v$.
if_dot	4	Evaluate the first and second arguments. If their dot product is greater than 0, then evaluate and return the third argument, else evaluate and return the fourth argument.
if> =	4	Evaluate the first and second arguments. If the magnitude of the first argument is greater than the magnitude of the second argument, then evaluate and return the third argument, else evaluate and return the fourth argument.

What is required by a GP tree program is to tell how to move an agent, i.e., right, left, up, down and stay. Thus, the wrapper (i.e., the mapping between the output of a parse tree and the action to be taken) is applied to the output of the GP tree so as to decide the agent's move. The mapping between vectors and actions is determined as follows: If the norm of the vector $\vec{v}$ is less than or equal to the parameter Radius, then STAY where you are. Otherwise, move 1 step RIGHT, UP, LEFT or DOWN depending on the direction of $\vec{v}$. i.e., when $\vec{v}$ is between $\left[-\frac{\pi}{4}, +\frac{\pi}{4}\right]$, $\left[+\frac{\pi}{4}, +\frac{3\pi}{4}\right]$, $\left[+\frac{3\pi}{4}, +\frac{5\pi}{4}\right]$ and $\left[+\frac{5\pi}{4}, +\frac{7\pi}{4}\right]$, respectively. This mapping is shown in Fig. 4. We set the Radius parameter to 1.0. For instance, if the output of a GP tree is a vector $\begin{pmatrix} 5 \\ 10 \end{pmatrix}$, then the agent's move is UP as the result of the wrapper.

Fitness functions need to be designed carefully so that they satisfy the following requirements:

Request 1: Give a high score to a GP program which moves an agent to its goal.

Request 2: Give a higher score to a GP program which finishes the tasks (i.e., moves all agents to their goals) quickly.

Request 3: If any agents have not reached the goals after the execution of a GP program, give a higher score when they have been moved nearer to their goals.

$$Wrapper(\vec{v}) = \begin{cases} \text{STAY}, & \text{if } \| \vec{v} \| < Radius \\ \text{RIGHT}, & \text{if } \| \vec{v} \| \geq Radius, \theta_{\vec{v}} \in [-\frac{\pi}{4}, +\frac{\pi}{4}] \\ \text{UP}, & \text{if } \| \vec{v} \| \geq Radius, \theta_{\vec{v}} \in [+\frac{\pi}{4}, +\frac{3\pi}{4}] \\ \text{LEFT}, & \text{if } \| \vec{v} \| \geq Radius, \theta_{\vec{v}} \in [+\frac{3\pi}{4}, +\frac{5\pi}{4}] \\ \text{DOWN}, & \text{if } \| \vec{v} \| \geq Radius, \theta_{\vec{v}} \in [+\frac{5\pi}{4}, +\frac{7\pi}{4}] \end{cases}$$

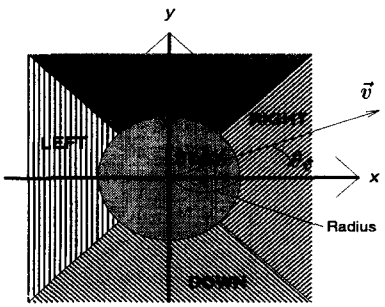


Fig. 4. Wrapper, i.e., mapping between the output of a parse tree and the action to be taken.

In order to meet the above requests, we use the following fitness for a GP tree T:

Step 1. Set $Step\_Time := 51, Fitness := 0.0$.

Step 2. Evaluate T and move agents according to the result of the wrapper.

Step 3. If an agent reaches its goal, then $Fitness := Fitness + Bonus \times FT$, where FT is the number of agents that have reached the goals at this step.

Step 4. If all agents reach their goals, then

$$Fitness := Fitness + Speed\_UP \times Step\_Time, \tag{1}$$

and return *Fitness*.

Step 5. $Step\_Time := Step\_Time - 1$.

Step 6. If *Step_Time* is zero, then

$$Fitness := Fitness + C_T \times \sum_{ag \in AG} \{\mathrm{d}(st(ag), gl(ag)) - \mathrm{d}(cr(ag), gl(ag))\} \ , \tag{2}$$

where AG is the set of remained agents. Return *Fitness*.

Step 7. Go to Step 2.

Initially, the maximum number of evaluations is set to 51 (Step 1). In Step 3, the value of *Bonus* is added to the fitness if an agent reaches its goal, which satisfies Request 1. If all agents are moved to their goals, i.e., the task is completed, the fitness value is increased with the remained *Step_Time* (Step 4). This meets the above Request 2. In Step 6, $d(x,y)$ means the distance of x and y. $st(ag), cr(ag)$, and $gl(ag)$ are the original position, the current position, and the goal position for an agent ag. Thus, $\{\mathrm{d}(st(ag), gl(ag)) - \mathrm{d}(cr(ag), gl(ag))\}$ equals to the moved distance of an agent ag toward its goal.

Therefore, the Eq. (2) means that the fitness is more increased if the remained agents have been moved nearer of their goals after the execution of the program, which satisfies the above Request 3.

We have chosen the *Bonus*, *Speed_UP*, and C_T parameters as 3000.0, 80.0 and 100.0, respectively. Since we used the tournament selection, (see Table 3), the absolute values of these parameters are not important. However, the *Bonus* parameter should be larger than the second term of the Eq. (2), so that GP searches for a program which completes the task at first.

2.2. Communication among agents

Communication is an essential factor for the emergence of cooperation. For example, Werner and Dyer [14] presented a simulation, in which a population of artificial organisms evolved simple communication protocols for mate finding. They showed how the organisms generated and interpreted meaningful signals as a result of evolution.

In this section, we use communication commands such as SEND or RECEIVE, by which an agent can tell another agent to stop or move. The SEND_*i* functional symbol takes two arguments and returns its second value (i.e. a two-dimensional vector). As a side effect, SEND_*i* sends its first argument to the *i*th nearest agent as a command. The RECEIVE function returns the evaluated result of the command message, if any, which has been sent to itself by the SEND_*i* command. The message list is an FIFO (i.e., first in, first out) queue. If no message is sent, then the RECEIVE function returns its argument by default.

For instance, consider the following GP trees for two agents:

Agent 0: (SEND_1 (inv Goal) Goal)

Agent 1: (if> = Ag1 Goal Goal (RECEIVE Goal))

Suppose that Agent 0 and Agent 1 compete with each other, e.g., A0 and A2 in Training #4 (see Fig. 2). That is, one's goal is another one's starting point and both agents have to pass through the same narrow path to reach their goals. At the first time step, the Agent 0 program returns the vector Goal. As a side effect, the SEND_1 function sends the command (inv Goal) to Agent 1. When evaluating the Agent 1 program for the first time, the RECEIVE function returns the Goal vector because there is no message in the queue. Thus, both agents draw near to their goals as a result. Next, at the second time step, the Agent 0 program sends (inv Goal) and returns the vector Goal as before. Then, in case of evaluating the Agent 1 program, if the Agent 0 is nearer to Agent 1 than its goal [4], the RECEIVE function returns the evaluated result of the pre-

[4] Note that when evaluating if > = function, if the magnitude of the first argument is greater than the magnitude of the second argument, then evaluate and return the third argument, else evaluate and return the fourth argument.

Table 2
GP Functions for communication

Name	# Args.	Description
Send_*i*	2	Send its first argument to the *i*th nearest agent as a command. Return its second argument.
Send_*i*Y	1	Send the YIELD command to the *i*th nearest agent. Return its argument.
Send_*i*S	1	Send the STOP command to the *i*th nearest agent. Return its argument.
Send_*i*R	1	Send the RANDOM command to the *i*th nearest agent. Return its argument.
Receive	1	Receive a message as a command. If no message is received, return its argument by default.

vious message (inv Goal), i.e., the inverse vector of Goal [5]. Otherwise, the Agent 1 tree returns the third argument of if >= function, i.e., the Goal vector. As a result, Agent 0 draws near to its goal, whereas Agent 1 moves in the direction opposite to its goal if Agent 0 is near enough, and moves toward its goal otherwise. Later on, the evaluation proceeds in the same way. Therefore, intuitively, the above communication realizes the following cooperation:

1. At first, both Agent 0 and Agent 1 approach their own goals.
2. When Agent 0 comes closer to Agent 1, Agent 1 gives way by moving in the direction opposite to its own goal.
3. When Agent 0 reaches its goal and stops sending a command, Agent 1 starts moving toward its destination.

As can be seen in the above example, through the usage of the communication commands, GP is expected to evolve a robust program, in the sense that agents can cooperate with each other more effectively in general situations.

The function symbols introduced for the communication are shown in Table 2. SEND_*i*Y, SEND_*i*S, and SEND_*i*R macros send commands such as YIELD (i.e., the receiver moves to one of its adjacent empty cell), STOP (i.e., the receiver stays at its current position) and RANDOM (i.e., the receiver moves randomly). These commands are commonly used for the motion control. Thus, in addition to the SEND_*i* and RECEIVE primitives, we introduce these macros for the sake of improving efficiency.

[5] Note that the terminal Goal is interpreted to each other. Thus, in this case, the received Goal message is interpreted as Agent 1's goal, not as Agent 0's goal.

Table 3
GP Parameters for sgpc1.1

max_generation	100	max_depth_after_crossover	17
population_size	2000	max_depth_for_new_trees	5
steady_state	0	max_mutant_depth	4
grow_method	GROW	crossover_any_pt_fraction	0.7
tournament_K	6	crossover_func_pt_fraction	0.1
selection_method	TOURNAMENT	fitness_prop_repro_fraction	0.1

2.3. Experimental results

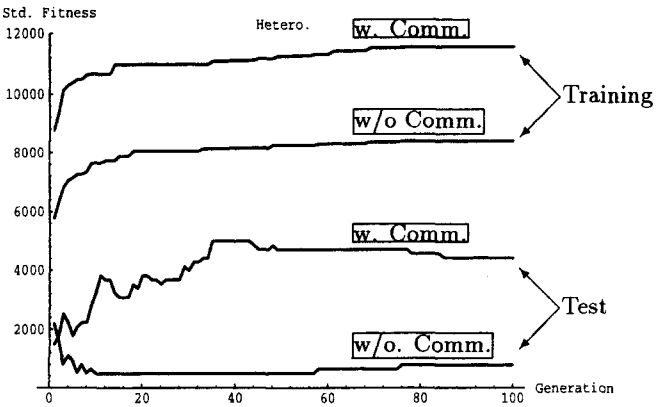
This section shows the experimental results of the robot navigation. We have implemented our GP based on sgpc1.1 [6], a simple GP system in C language. The used parameters are shown in Table 3, Fig. 2 shows some of the training and testing cases. We chose six training and three testing scenarios. They were modified to constitute a variety of examples in several ways, such as rotating or widening a passage. The total number of the training and testing data were 24 and 9, respectively. The fitness of a program is the averaged fitness over the various training cases.

Experiment 1 (Communicating agents vs. non-communicating agents): First, we have conducted comparative experiments so as to confirm the effectiveness of communication. The heterogeneous strategy was applied for evolving agents without communication (i.e., with GP functions and terminals shown in Table 1) and for evolving communicating agents (i.e., Tables 1 and 2).

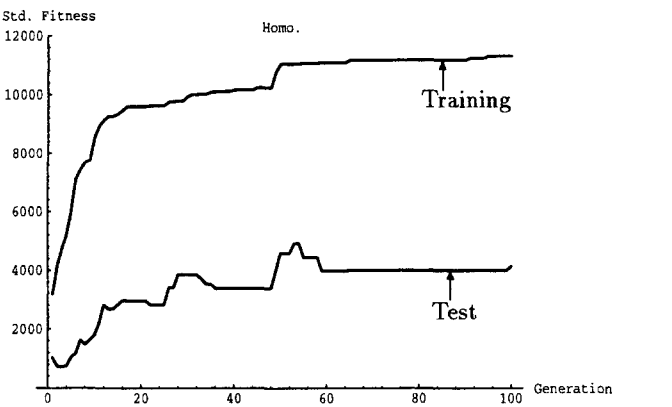
Fig. 5(a) shows the result of experiments. The figure plots the best fitness value with generations, averaged over 10 runs. The fitness value of $4 \times Bonus(= 12000.0)$ is given to a GP tree which completes the task i.e., moves all agents to the goals. Thus, on the average, GP reaches a solution around 60 generations for the communicating agents, whereas agents without communication could not solve the task after 100 generations. Note the superiority of the communicating agents for the testing cases as well as for the training cases.

The poor performance of non-communicating agents results from the lack of appropriate generalization. They could adapt to a certain situation and memorize it as a specialized cognitive map. But they failed to generalize it so as to cope with multiple cases. For instance, the trees acquired most often were a simple form of Goal terminals. As mentioned in Section 1, these trees tend to move agents to their goals. But the agents are liable to fall into a deadlock (see Fig. 3). The better programs described below were acquired in one run:

[6] sgpc1.1 is available by anonymous FTP to ftp.io.com. The directory is /pub/genetic-programming.



(a) Heterogeneous Strategy.



(b) Homogeneous Strategy.

Fig. 5. Experimental results (robot navigation task): (a) Heterogeneous strategy; (b) Homogeneous strategy.

```
Agent 0: (if>=Agl Vl Goal (inv Goal))
Agent 1: (if>=Agl (*2 Vl) Goal (inv Goal))
Agent 2: (if>=Agl (*2 (*2 Vl) Goal (inv Goal))
Agent 3: (if>=Agl Vl Goal (inv Goal))
```

These programs realize a form of cooperation, in the sense that an agent moves to its goal if the other agents are further than some threshold (i.e., (if >= Agl***)). If the nearest agent is close, it gives way by moving in the direction opposite to its goal (i.e., (inv Goal)). Although this strategy succeeded in some limited situations, the agents failed to solve a complicated task such as Training #1. Fig. 6 shows the resultant behavior of agents for this training case, in which only one agent reached its goal.

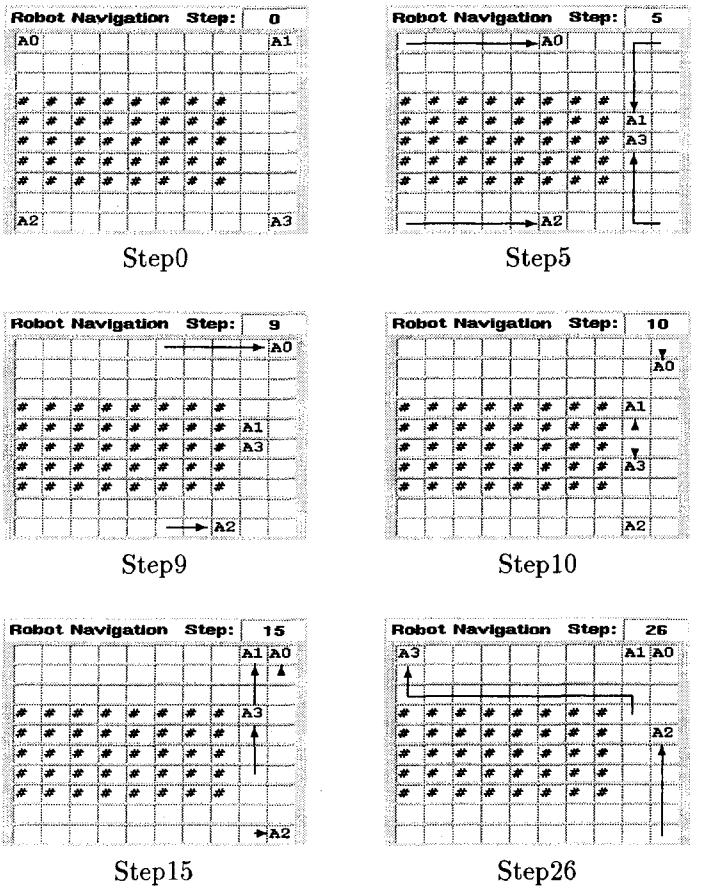


Fig. 6. Emergent behavior without communication.

On the other hand, agents with communication were able to solve the various tasks. For instance, the programs shown in Table 4 were acquired in one run at the generation of 56. These programs scored the standard fitness of

Table 4
Acquired trees for heterogeneous strategy

Agent 0	(Receive Goal)
Agent 1	(if >= Goal Ag1 (if >= (Receive Goal) (Send1_Y Goal) (if >= (if >= Goal Goal (Receive Goal) Goal) (Send1 Goal Goal) (*2 (Receive Goal)) (Receive Goal)) Goal) Goal)
Agent 2	(Receive Goal)
Agent 3	(if >= (if >= Ag1 V1 (Receive Goal) V1) V1 (Send1_Y Goal) (if >= Ag1 Goal Goal (Receive Goal)))

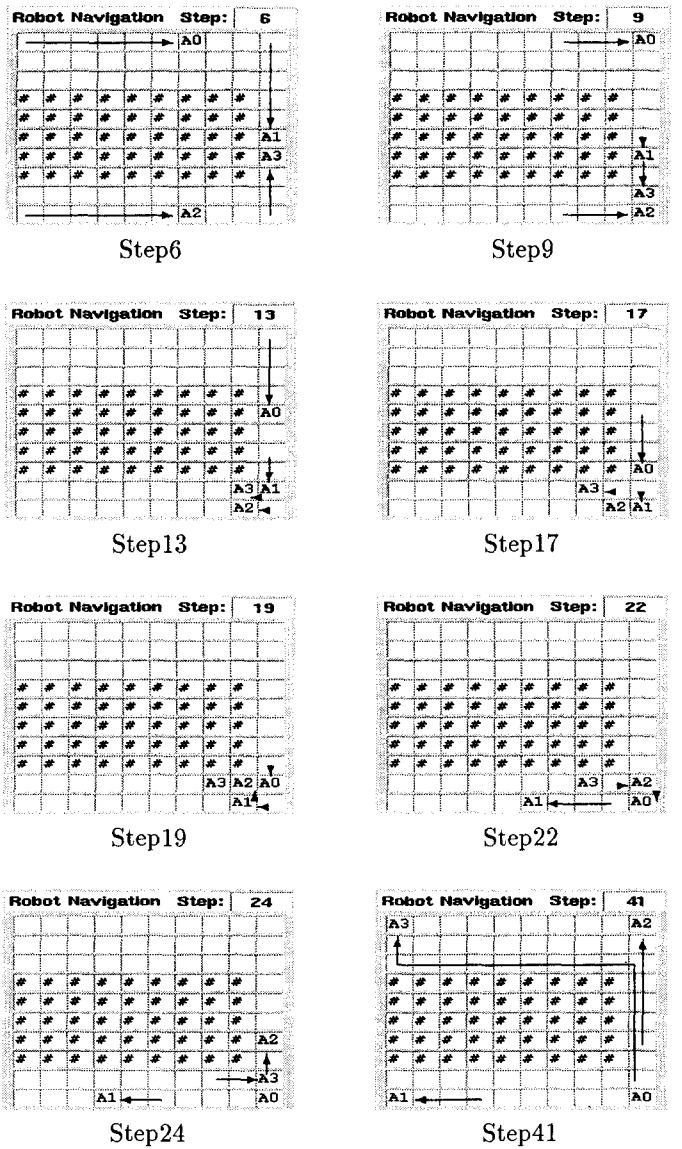


Fig. 7. Emergent behavior with communication.

13520.0, which means that they solved most of the training cases. Fig. 7 shows the emergent behavior of these agents for Training #3. The key features of these programs are as follows:

1. Agent 0 and Agent 2 are receivers, which always give way if they receive a message.

2. Agent 3 is a receiver if the nearest agent is far, i.e., $|Ag1| \geq |V1| \Rightarrow$ Receive. It sends a yield message otherwise, because (if >= Ag1 V1 (Receive Goal) V1) returns V1 and (if >= V1 V1(Send1_YGoal)***) always returns the third argument (Send1_Y Goal).
3. Agent 1 moves to its goal if the nearest agent is further than the goal. Otherwise, unless it has received any message, it sends a yield message to its nearest agent. If it has received any message, execute the received command.

As can be seen in the behavior of A3 from Step 9 to Step 19 (Fig. 7), if an agent receives a message, it stops moving or gives way. As a result of this effective communication, agents cooperate with each other to avoid the deadlock situation in the narrow path.

The above experimental results have shown that the communicating agents complete the training tasks more effectively, i.e., the evolved program is more robust. It is also suggested that the GP-based adaptive learning resulted in establishing the effective job separation among the communicating agents.

Experiment 2 (Homogeneous Strategy; Fig. 1(b)): For the sake of comparison, we have experimented with the evolution of homogeneous agents with communication. The performance of the homogeneous strategy was almost as good as the heterogeneous strategy. Although they were controlled by the same program, the homogeneous agents also evolved so as to cooperate with each other for the sake of solving the task. Fig. 5(b) shows the experimental result, in which the best fitness value is plotted with generations, averaged over 10 runs. Slightly poor performance seems to be due to the fact that homogeneous agents require a large tree structure. The acquired tree was much more complicated than those by the heterogeneous strategy. For instance, in one run, the following tree was acquired:

(if >= Goal (Receive Goal)(Receive Goal) (if >= (Receive Goal)
(*2(if >= (*2(if >= Goal Ag1 V1 Goal))Ag1 V1 Goal))(if >= Ag1
(Receive Goal) Goal(if >= (if >= (Send1_YGoal)(*2(Receive Goal))
(Send1_Rev Goal) Ag1)(*2goal)Ag1(Receive Goal))) Goal))

This program scored the standard fitness of 12250.0, which means that they solved most of the training cases. Considering the symmetry of the training and testing cases, the good performance of the homogeneous strategy may not be very surprising. For instance, the test cases #0 and #1 are identical to the rotated cases of the training case #3 by 90° and 180°, respectively. For four agents, who have to move to their own goals, the relative goal direction is more important than the absolute direction. Because the ultimate goal of this task is to acquire a generalized strategy for a given set of cases, we believe that the homogeneous agents have managed to do the job. The different setting, such as introducing agents with various sensors, will have shown the difficulty of the homogeneous strategy (see [8] for more details).

3. The pursuit problem

The Predatory-Prey pursuit problem is a test bed in Distributed Artificial Intelligence (DAI) research to evaluate techniques for developing cooperation strategies [15]. The problem domain is a grid world, in which four blue (predator) agents attempt to capture a red (prey) agent by surrounding it from four directions on a grid world (see Fig. 8(a)). In the figure, four agents are represented by white circles (i.e., 1,2,3,4), and the prey by a small black dot (i.e., •). Agent movements are limited to one horizontal or vertical step per time unit.

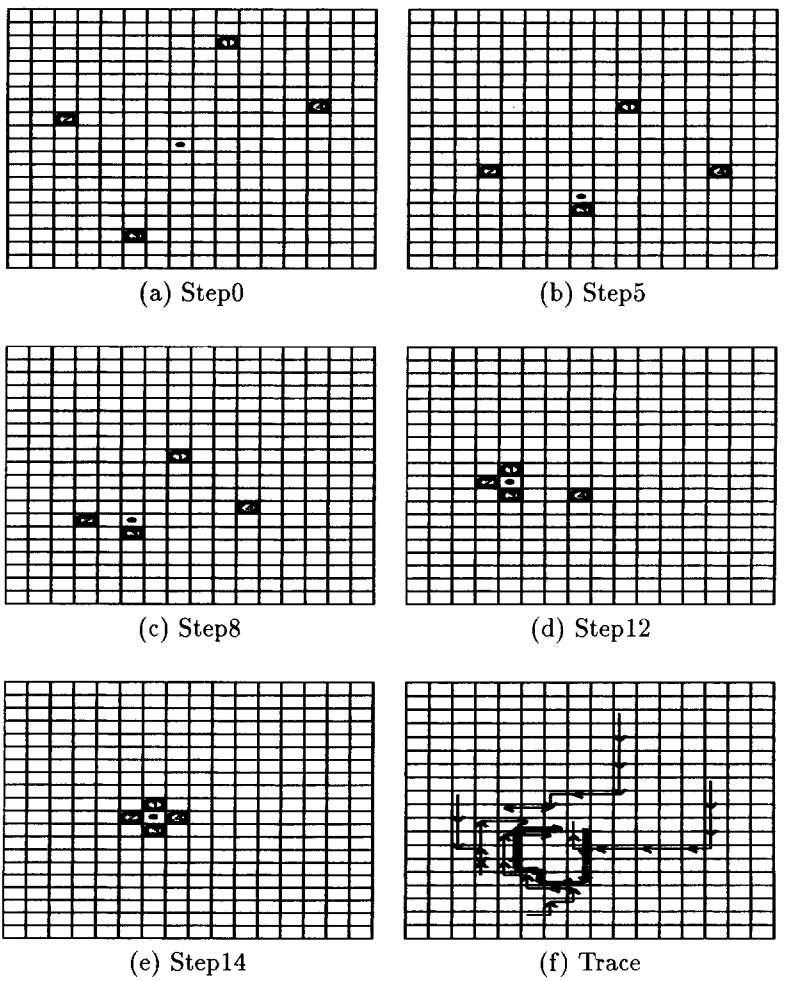


Fig. 8. Solving a pursuit game.

The approach undertaken by Gasser et al. [16] was a Lieb configuration, in which each predator occupies a different quadrant, where a quadrant is defined by diagonals intersecting at the location of the prey, while homing in on the prey (see [17] for details).

The research of the pursuit problem was aimed at showing the efficiency with which blue agents could capture the red agent. In other words, when agents with limited ability and partial information seek for the same goal, how do they cooperate with each other in order to solve the whole problem?

This section describes GP-based approach to evolving a solution for this problem.

3.1. The pursuit problem and genetic programming

In order to apply GP to evolving an agent's program in the pursuit problem, we use the terminal and nonterminal sets shown in Table 5. In the table, a symbol without arguments is a terminal symbol. The agents' actions are STAY or move RIGHT, UP, LEFT or DOWN.

We have chosen vector operations for the GP tree representation, in the same way as the robot navigation. For instance, assuming that x and y axes are rightwards and upwards as usual, the Target terminal returns a vector $\begin{pmatrix} 2 \\ 7 \end{pmatrix}$ for the agent A3 and $\begin{pmatrix} 5 \\ -2 \end{pmatrix}$ for A2, in the initial situation (Fig. 8(a)). The same wrapper is also used to map between the output of a parse tree

Table 5
GP Terminals and functions (pursuit problem)

Name	# Args.	Description
Target	0	The vector from the agent to the target, i.e., the prey.
Ag*i*	0	The vector from the agent to Agent*i*.
Rand	0	A random vector.
+	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 $-\vec{v}$.
if_dot	4	Evaluate the first and second arguments. If their dot product is greater than 0, then evaluate and return the third argument, else evaluate and return the fourth argument.
if >=	4	Evaluate the first and second arguments. If the magnitude of the first argument is greater than the magnitude of the second argument, then evaluate and return the third argument, else evaluate and return the fourth argument.

and the action to be taken, i.e., it is applied to the output of the GP tree so as to decide the agent's move (see Fig. 4).

At each time step, an agent is executing an action according the above mapping. During this period, the agents can either complete the task, i.e., they have captured the prey, or the prey gets away without being captured.

In order to assign a fitness *Fitness* to an agent's behavior, we use the following fitness calculation for a GP tree T:

Step 1. Set $Step\_Time := 30, Fitness := 0.0$.

Step 2. Evaluate T and move agents according to the result of the wrapper.

Step 3. $Fitness := Fitness + \sum_{i=0}^{3} d_i$ where d_i is the distance between the prey and the agent i.

Step 4. If the prey is captured, then return *Fitness*.

Step 5. $Step\_Time := Step\_Time - 1$.

Step 6. If *Step_Time* is zero, then return *Fitness*.

Step 7. Go to Step 2.

Initially, the maximum number of evaluations (i.e., *Step_Time*) is set to be 30 (Step 1). The above fitness calculation assures that the sooner the task is finished, the better (i.e., the smaller) the fitness is. We have used the heterogeneous strategy for this experiment.

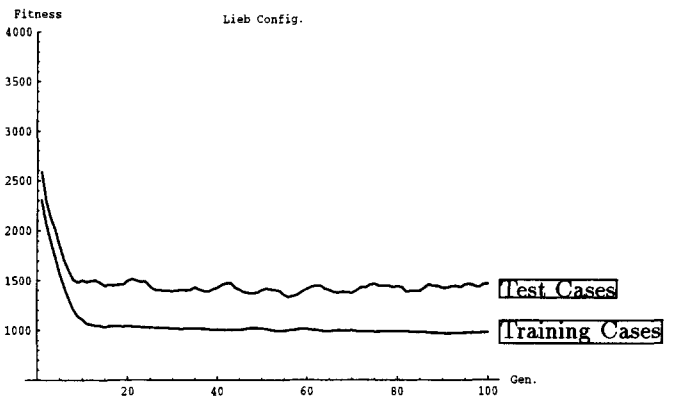
GP is using the fitness of an agent's behavior to evolve new and possibly better behaviors. The implementation chosen is sgpc1.1. We used the same parameters as that which was shown in Table 3, except that the population size was set to be 500.

3.2. Robustness of GP-based multiple-agent learning

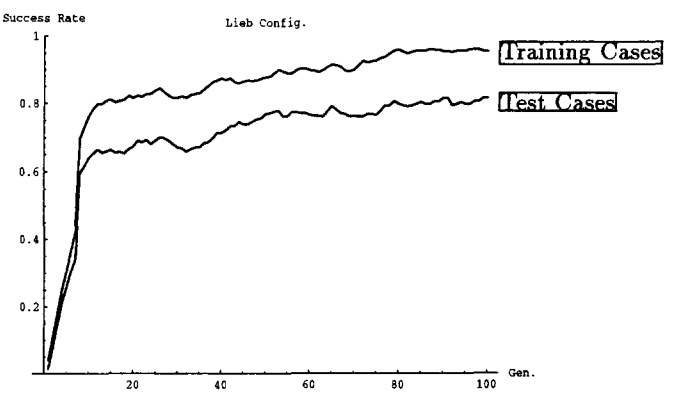
The robustness is defined as the ability of agents to cope with noisy or unknown situations. In the pursuit problem, the robustness of an acquired GP program is tested by examining the performance for the unknown test data. Thus, we conduct experiments under the following conditions:

1. The fitness of a GP individual is calculated by using 15 different training maps. These maps are randomly generated.
2. At each generation, the best GP individual is tested for validating its robustness, i.e., calculating the performance for 10 testing maps randomly generated.
3. For both training and testing maps, the initial position of agents and the prey constructs a Lieb configuration. This is to ensure that the problem is solvable, i.e., predators are able to capture the prey if they cooperate with each other effectively. The difficulty of the pursuit problem in a different setting is discussed in details in [17].
4. When evaluating the fitness, the prey attempts to get away from the agents. More precisely, the prey moves in the direction opposite to the gravity center of four agents.

Fig. 9 shows the experimental results for this problem. The figures plot the best fitness value with generations (Fig. 9(a)) and the ratio of success (i.e., the prey captured by four agents) for test and training data (Fig. 9(b)). The best fitness values were averaged over 30 runs. The non-monotonicity of the fitness transition is due to the randomness of the data generation. Fig. 8 illustrates a trace of agents for a typical run, in which four agents succeed in capturing the prey after 14 time steps. This trace resulted from GP programs acquired at the 83rd generation. Note that agents did not necessarily go towards the prey. This deviation seems to ensure the robustness for unknown situations, because the other agents and the prey often behave unpredictably in such a new situation. As can be seen in Fig. 9, we can confirm that the robustness is increased with generations as a result of GP-based learning.



(a) Fitness vs. Generations.



(b) Success Ratio vs. Generations.

Fig. 9. Experimental results (Lieb configuration): (a) Fitness vs. generations; (b) Success ratio vs. generations.

3.3. Evolving communicating agents

We have introduced the communication for the pursuit problem by using the following functional symbols:

$$F = \{\mathrm{COM0}_1, \mathrm{COM1}_1, \mathrm{COM2}_1, \mathrm{COM3}_1\}. \tag{3}$$

This COM? command takes one argument and requests data from the agent? If the agent? can see the target, i.e., within the range of scope, then the COM? function returns the vector from the calling agent to the target. Otherwise, the COM? returns its argument (i.e., by default).

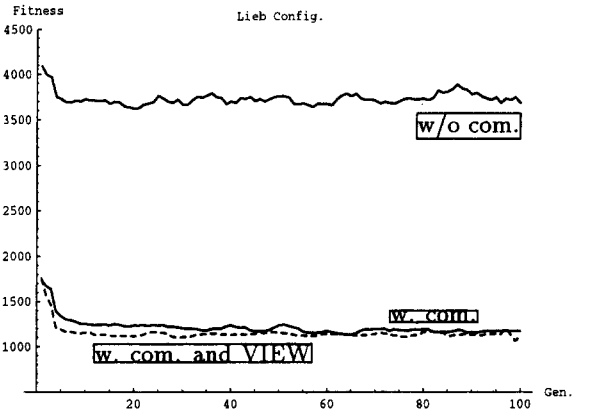
We conduct the experiments under the following conditions:

1. An agent has a finite scope area, i.e., an object is visible to an agent if it is within a circle whose radius is a given value V (i.e., visibility) and whose center is the agent.
2. Initially, agents construct a Lieb configuration and the prey is visible to at least one agent.
3. The prey makes a random motion.

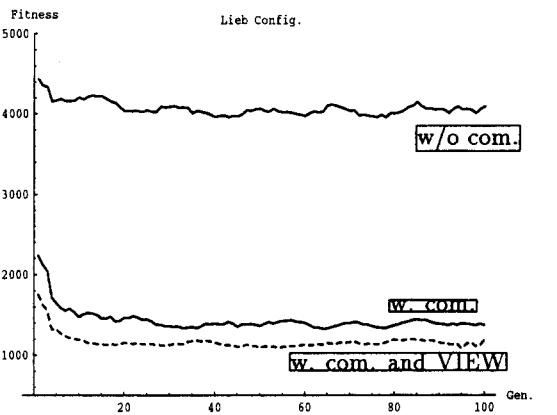
In our experiment, we set the visibility V to be 5. For the sake of confirming the effectiveness of communication, the following methods are compared:

1. No communication commands are used, i.e., the function set is the same as Table 5.
2. Communication commands are used (Eq. (3)). The communication cost is one time step. That is, if any communication command is used in evaluating a tree, then the agent has to wait one time step to make the motion based upon the wrapper result of the tree output.
3. Besides communication commands, a special macro VIEW is used. The communication cost is the same as the above. The VIEW macro takes one argument and evaluates its argument if and only if the prey is visible to the agent.

The experimental results are shown in Fig. 10, in which the fitness values for training and testing data are plotted with generations. The best fitness values were averaged over 30 runs. The non-monotonicity of the fitness transition is due to the randomness of the data generation. Table 6 summarizes the performance comparison of three methods. Although the superiority of communicating agents is clearly shown in terms of the fitness transition (Fig. 10), the averaged numbers of success are not remarkably different among these three methods. The agents without communication could capture the prey in some cases, because the prey moves randomly. But in most cases, they were unable even to draw near to the prey. On the other hand, the communicating agents succeeded in enclosing the prey. However, they did not always manage to capture it. In this experiment, agents were assumed to have a limited visibility. Thus, in earlier stage of pursuit, the communication is necessary. On the contrary, later when the prey is near enough, the communication is becoming a



(a) Fitness for Training Data.



(b) Fitness for Testing Data.

Fig. 10. Experimental results (communicating agents): (a) Fitness for training data; (b) Fitness for testing data.

Table 6
Performance comparison

Communication	×	○	○
VIEW Command	×	×	○
Avg. # of success (training)	2.71	4.83	5.00
Avg. # of success (test)	1.71	3.17	3.50
#. of COM?	–	653.48	363.11

burden because of the communication cost, i.e., the agent has to stop one time step for communication. Table 6 shows the averaged number of COM? commands when evaluating the training data for the best individual. As can be seen, the number is much smaller for the agents with VIEW macro. This shows that VIEW command is expected to work as a way of eliminating unnecessary communication. We are working on the further extension of this VIEW command in order to improve the success ratio of testing data.

4. Discussion

4.1. Related works

GP and its variants have been applied to multi-agent learning. For instance, Koza used GP to evolve sets of seemingly simple rules that exhibit an emergent behavior. The goal was to genetically breed a common computer program, when simultaneously executed by all the individuals in a group of independent agent, i.e., the homogeneous breeding, that causes the emergence of beneficial and interesting higher-level collective behavior ([1], Ch. 12).

Fogarty et al. [2] studied the evolution of the multiple communicating classifier systems in the heterogeneous environment of a distributed control system for a walking robot. They introduced the "symbiosis" analogy to realize a macro-level operator to the evolution of heterogeneous species and showed the effectiveness of their approach empirically. But they failed to observe the evolution of a "superorganism" by their experiments. They also investigated the evolution of multiple fuzzy controllers in the homogeneous environment of a distributed control system for a communication network.

Haynes proposed an approach to the construction of cooperation strategies based on GP for a group of agents [3]. He experimented in the predator-prey domain, i.e., the pursuit game, and showed that the GP paradigm could be effectively used to generate apparently complex cooperation strategies without any deep domain knowledge.

Luke examined three breeding strategies (clones, free and restricted) and three coordination mechanisms (none, deictic sensing, and named-based sensing) for evolving teams of agents in the Serengeti world, a simple predator/prey environment [4]. Our paper has been partly motivated by this experiment.

In our previous paper [6], we have applied GP-based multi-agent learning to the Tile World and proposed a co-evolutionary breeding scheme. Experimental results have shown the superiority of the co-evolutionary breeding over the two strategies, i.e., the homogeneous strategy and the heterogeneous strategy. In the co-evolutionary strategy, some individuals were expected to perform specialized tasks for different agents with generations. We will make an attempt at extending this scheme for the evolution of communicating agents.

4.2. The robustness against noise

In the previous experiments, the robustness was tested by validating the generated programs for some unseen cases. It also can be measured by the ability to cope with noise, which is an important and inevitable feature for real-world applications. Thus, we have been studying the robustness of the generated GP program against a noisy situation in the robot navigation. We used the best program evolved so far (i.e., that which was shown in Table 4) for the testing. The noisy situation was realized by reducing the sensor precision. More precisely, we introduced the parameter *ERROR_RATE* and follow the steps described below:

1. The Goal terminal returns a random vector with the probability of *ERROR_RATE*.
2. The A*gi* terminal returns a random vector with the probability of *ERROR_RATE*.

Fig. 11 shows the experimental result. The figure plots the fitness value of training and testing data with different error rates, averaged over 100 runs. As expected, the fitness became lower with higher error rates. However, the figure shows that the fitness value above 8000 was kept with the error rate of 20%. Half of the training cases were solved correctly with the error rate. Although we cannot make any concluding remarks with this small experiment, we think that the generated GP program is robust in terms of the graceful degradation against noise. This topic in connection with real-world applications is our future research concern.

4.3. Different coordination mechanism

Terminal and functional symbols chosen in the previous experiments were based on name-based sensing coordination described in [4]. Luke showed that

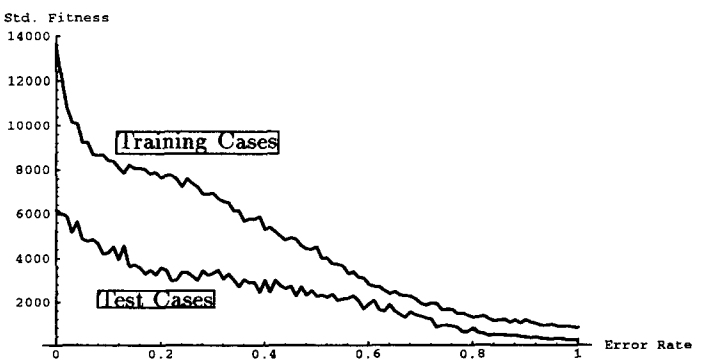


Fig. 11. Graceful degradation for a generated GP program.

the name-based sensing consistently outperforms the other two (i.e., none and deictic sensing) for his experiment. The effect of the agent's view range has been a key issue of DAI research. Thus, we also have studied the different coordination mechanisms, i.e., the different view ranges and step lengths for the robot navigation [5]. Experimental results showed that the homogeneous strategy gave a poor result. This is because, in spite of different coordination mechanisms, all agents were controlled by the same program, i.e., the same GP individual. The performance of the heterogeneous breeding strategy was much better. However, we think that using communication helps agents to cooperate with each other in a different and effective manner. We are currently working on the extension of our scheme in this direction.

4.4. Future research

The previous experiments have shown that GP was successfully applied to multi-agent tasks. The tasks were achieved by the effective job separation, i.e., the cooperation emerged among multiple agents. Then, the following questions arise. Which part of the resultant GP tree is the cooperation? Is the job separation explicitly represented in the acquired tree?

These topics remain to be seen, and we are currently researching on them. The cooperative behavior is usually implicitly described in a GP tree. However, it might be possible to find a useful subtree (i.e., subroutine) in an acquired tree. For instance, Koza [18] experimented in an artificial ant problem by using the ADF-version of GP and observed that the useful subroutine, i.e., a semicircular counterclockwise inspecting motion, was expressed in the ADF branch of the resultant tree. Some researchers focused on the extension of GP to select an effective subtree and add it to a new function [19,20]. The subroutine discovery based on GP is also our current research interest [21]. We have been trying to extract a useful subtree from a population of GP trees for the sake of interpreting the cooperative behavior as well as improving the efficiency.

5. Conclusions

This paper described the emergence of cooperative behavior based on GP. We have confirmed the following points by experiments:

1. GP was successfully applied to multi-agent test beds, i.e., the pursuit problem and the robot navigation.
2. The robustness of the acquired GP program was examined by testing data.
3. We have confirmed how agents cooperate with each other via communication as a result of the GP-based learning.

Our goal was to realize the emergence of the job separation among communicating agents. This paper has shown a feasibility study on GP-based multi-

agent learning, and we believe that it is a first step to the emergence of cooperation among communicating agents.

Acknowledgements

We are grateful to Walter Alden Tackett, for providing his Simple Genetic Programming in C ("sgpcl.1"), which we used to conduct our comparative experiments. We have profited from comments by anonymous reviewers.

References

[1] J. Koza, Genetic Programming, On the Programming of Computers by means of Natural Selection, MIT Press, 1992.
[2] T. Fogarty, L. Bull, B. Carse, Evolving Multi-Agent Systems, in: G. Winter, J. Pèriaux, M. Galàn, P. Cuesta (Eds.), Genetic Algorithms in Engineering and Computer Science, Wiley, Chichester, 1995.
[3] T. Haynes, R. Wainwright, S. Sen, Evolving a Team, in: Working Notes of the AAAI-95 Fall Symposium on Genetic Programming, AAAI Press, 1995.
[4] S. Luke, L. Spector, Evolving teamwork and coordination with genetic programming, in: Genetic Programming, MIT Press, 1996.
[5] H. Iba, Multiple-agent learning by genetic programming, in: ICML96 Workshop on Evolutionary Computation and Machine Learning, 1996.
[6] H. Iba, Emergent cooperation for multiple agents using genetic programming, in: Parallel Problem Solving from Nature IV (PPSN96), 1996.
[7] H. Iba, T. Nozoe, K. Ueda, Evolving communicating agents based on genetic programming, in Proc. of the IEEE International Conference on Evolutionary Computation (ICEC97), 1997.
[8] H. Iba, Multiple-agent learning for a robot navigation task by genetic programming, in: Proceedings of the Genetic Programming Conference (GP97), 1997.
[9] J. Koza, Genetic Programming II, Automatic Discovery of Reusable Programs, MIT Press, Cambridge, MA, 1994.
[10] J. Chu-Carroll, S. Carberry, Communicating for conflict resolution in multi-agent collaborative planning, in: Proceedings of the First International Conference on Multi-Agent Systems (ICMAS95), 1995.
[11] M. Benda, V. Jagannathan, R. Dodhiawalla, On optimal cooperation of knowledge sources, in: Proceedings of the Workshop on Distributed Artificial Intelligence, May 1988.
[12] T. Ito, H. Iba, M. Kimura, Robot programs generated by genetic programming, Japan Advanced Institute of Science and Technology, IS-RR-96-0001I, in: Genetic Programming, 1996.
[13] H. Yokoi, Y. Kakazu, Autonomous grasp control of link mechanism by vibrating potential method, in: Control of Engineering Practice, Elsevier, 1994, pp. 1031–1038.
[14] G.M. Werner, M.G. Dyer, Evolution of communication in artificial organisms, in: C.G. Langton, C. Taylor, J.D. Farmer, S. Rasmussen (Eds.), Artificial Life, vol. II, Addison-Wesley, New York, 1991.
[15] R. Levy, J. Rosenschein, A game theoretic approach to distributed artificial intelligence and the pursuit problem, in: E. Werner, Y. Demazeau (Eds.), Decentralized Artificial Intelligence, Elsevier, Amsterdam, 1992.

[16] L. Gasser, N.F. Rouquette, R.W. Hill, J. Lieb, Representing and using organizational knowledge in distributed AI systems, in: L. Gasser, M.N. Huhns, (Eds.), Distributed Artificial Intelligence, vol. 2, Morgan Kaufmann, Los Altos, 1989.

[17] T. Haynes, K. Lau, S. Sen. Learning cases to compliment rules for conflict resolution in multiagent systems, In: S. Sen, (Ed.), *Working Notes for the AAAI Symposium on Adaptation, Co-evolution and Learning in Multiagent Systems*, Stanford University, CA, 1996, pp. 51–56.

[18] J. Koza, Simultaneous discovery of reusable detectors and subroutines using genetic programming, in: Proceedings of the Fifth International Conference on Genetic Algorithms, Morgan Kaufmann, Los Altos, 1993.

[19] P.J. Angeline, J.B. Póllack, Evolutionary module acquisition, in: Proceedings of the Second Annual Conference on Evolutionary Programming, 1993.

[20] J.P. Rosca, D.H. Ballard, Hierarchical self-organization in genetic programming, in: Proceedings of the Eleventh International Conference on Machine Learning, 1994.

[21] N. Hondo, H. Iba, Y. Kakazu, COAST: An approach to robustness and reusability in genetic programming, ETL-TR-96-4, 1996.