Hitoshi Iba

This paper presents the emergence of the cooperative behavior for multiple agents by means of Genetic Programming (GP). Our experimental domains are multi-agent test beds, i.e., the robot navigation task and the Tile World. The world consists of a simulated robot agent and a simulated environment which is both dynamic and unpredictable. In our previous paper, we proposed three types of strategies, i.e, homogeneous breeding, heterogeneous breeding, and co-evolutionary breeding, for the purpose of evolving the cooperative behavior. We use the heterogeneous breeding in this paper. The previous Q-learning approach commonly used for the multi-agent task has the difficulty with the combinatorial explosion for many agents. This is because the state space for Q-table is so huge for the practical computer resources. We show how successfully GP-based multi-agent learning is applied to multi-agent tasks and compare the performance with Q-learning by experiments. Thereafter, we conduct experiments with the evolution of the communicating agents. 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. The effectiveness of the emergent communication is empirically shown in terms of the robustness of generated GP programs.

19.1 Introduction

This paper applies GP in order to evolve multiple agents and shows that the cooperative behavior emerges as a result of evolution. There are three main motivations for us to realize GP-based multi-agent learning.

First, there have been many approaches to adaptive agents. For instance, reinforcement learning, i.e., Q-learning, is often used for multi-agent tasks [Tan93], [Sen *et al.*95]. However, most of these straightforward approaches scale poorly to more complex multi-agent learning problems, because the state space for each learning agent grows exponentially in the number of its partner agents engaged in the joint task [Ono97],[Rosca96]. On the other hand, GP searches for the combination of input variables so as to reduce the computational complexity. We will explain the comparative study in Section 4.

Secondly, many breeding strategies, e.g. homogeneous strategy, heterogeneous strategy, and co-evolutionary strategy, are proposed and compared for the multi-agent learning method. Thus, we can use a different strategy for a particular task by using evolutionary learning. For instance, [Bull97] described the key feature of the co-evolving strategy as "genetic joining" i.e., hereditary endosymbioses, and observed that it worked best in terms of mean performance due to their reduction in the effects of the partner variance (see [Iba96],[Iba97] and [Bull97] for the details of the comparative studies).

The third motivation comes from the communication issue. 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 [Chu-Carroll *et al.*95]. [Benda *et al.*88] 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 exert control over another agent.

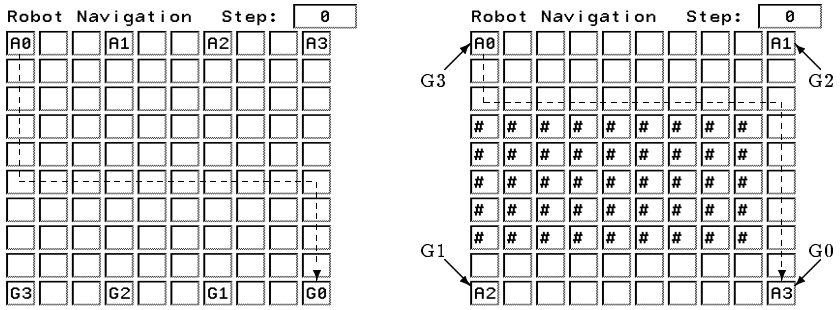
However, it is not easy to design agents with the above communication protocols by humans, because many factors, such as synchronization, communication costs, and transmission channel, have to be considered beforehand. There have been relatively few studies on the adaptive agents with such higher-level communication as types B and C. GP is suitable for representing these communication protocols. In Section 5, we use ACL (Agent Communication Language) to show the evolvability of communicating agents.

The rest of this paper is structured as follows. Section 2 describes the experimental domains used in this paper, i.e., the robot navigation task and the Tile World. Section 3 explains different breeding strategies for multi-agent learning. In section 4, GP performance is compared with Q-learning. We introduce communication commands in section 5 and conduct experiments with communicating agents. Section 6 discusses our approach, followed by some conclusions in Section 7.

19.2 Example Tasks

There are some benchmark problems for the multi-agent research. We use two target tasks in this paper.

First task is a robot navigation domain. Consider a robot navigation task for four agents (Fig.19.1). The world consists of a rectangular grid (i.e., 10×10), on which agents (denoted as Ai, $i = 0, 1, \cdots$) and some obstacles (#) can be placed. 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 (denoted as Gi in the figure). A possible optimal path for A0 is shown in the figure as a dotted line. The goal of the multi-agent team is for each agent to move quickly to their respective goal locations without



(a) Scenario RN1. (b) Scenario RN2.

Figure 19.1
The Robot Navigation.

colliding with other agents and walls. In Scenario RN1, the obstacles are the other agents alone. This is an example of loosely-coupled interactions [Sen *et al.*95].

In order to apply GP to evolving an agent's program in the robot navigation, we use the basic terminal and nonterminal sets shown in Table 19.1[1] . In the table, a symbol without any argument is a terminal symbol.

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 $[-\frac{\pi}{4}, +\frac{\pi}{4}]$, $[+\frac{\pi}{4}, +\frac{3\pi}{4}]$, $[+\frac{3\pi}{4}, +\frac{5\pi}{4}]$ and $[+\frac{5\pi}{4}, +\frac{7\pi}{4}]$, respectively. We set the Radius parameter to 1.0. For instance, if the output of a GP tree is a vector $\binom{5}{10}$, then the agent's move is UP as the result of the wrapper.

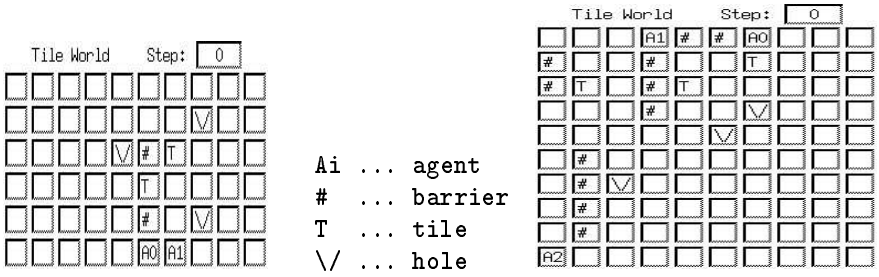
The second experimental domain is the Tile World, which is a multi-agent test bed that consists of simulated robot agents and a simulated environment which is both dynamic and unpredictable [Pollack *et al.*90], [Hanks *et al.*93]. The world also consists of a rectangular grid, on which agents (denoted as Ai, $i = 0, 1, \cdots$), some tiles (T), some obstacles (#), and some holes (\/) can be placed (see Fig.19.2(a)). 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. When a tile is in a cell adjacent to the agent, the agent can push the tile by moving in its direction. The agents' goal is to push all tiles into the holes. The

[1] The usage of these symbols is motivated by the study reported by [Luke *et al.*96]. Luke studied evolving teamwork by GP for a pursuit game, in which the world is a continuous 2-dimensional area.

Table 19.1
Basic 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.

simple interaction is shown in Fig.19.2(a). This is an example of strongly-coupled interactions (i.e. the constraints on movement are so severe) [Goldman *et al.*94]. For either agent to accomplish its goal, it would need to carry out a large number of movements to perform its own task. For example, for A0 alone to fill holes with two tiles, it needs to move 17 steps (assuming A1 is not on its way). Similarly, A1 alone would need to move 26 steps to finish its task. However, if they work together, they can finish their task by going 12 steps (see [Iba96] for how GP evolved this optimal behavior). We have chosen the same basic terminal/functional symbols shown in Table 19.1 for this task, except that Hole and Tile terminals are used in stead of Goal terminal. The Tile terminal returns the vector from the agent to the nearest tile and the Hole returns the vector from the agent to the nearest hole.



(a) Tile World 1 (TW1). (b) Tile World 2 (TW2).

Figure 19.2
The Tile World.

19.3 Fitness Assignment and Breeding Strategies

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

Requirement 1 Give a high score to a GP program which makes each agent achieve its own goal.

Requirement 2 Give a higher score to a GP program which finishes the whole task quickly.

Requirement 3 If any agents have not achieved the goals after the execution of a GP program, give a higher score when they have contributed to their goals more.

In order to meet the above requirements, we use the following fitness derivation for a GP tree T. This algorithm includes three user-defined parameters, i.e., $Bonus$, C_T, and $Speed\_UP$. The meaning of these parameters is explained later.

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 achieves its goal, then

$$Fitness := Fitness + Bonus \times FT, \tag{19.1}$$

where FT is the number of agents that have achieved the goals at this step.

Step 4 If all agents achieve their goals, then

$$Fitness := Fitness + Speed\_UP \times Step\_Time, \tag{19.2}$$

and return $Fitness$.

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

Step 6 If $Step\_Time$ is zero, then

$$\begin{aligned} Fitness \quad := \quad & Fitness \\ & + C_T \times \sum_{ag \in AG} \{d(st(ag), gl(ag)) - d(cr(ag), gl(ag))\}, \end{aligned} \tag{19.3}$$

where AG is the set of remaining 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 achieves its goal, which satisfies **Requirement 1**. If all agents achieve their goals, i.e., the task is completed, the fitness value is increased with the remaining $Step\_Time$ (**Step 4**). This meets the above **Requirement 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 destination (i.e., the goal for the robot navigation task and the nearest tile for the Tile World) for an agent ag. Thus, $\{d(st(ag), gl(ag)) - d(cr(ag), gl(ag))\}$ equals the motion distance of an agent ag toward its goal. Therefore, the equation (19.3) means that the fitness is more increased if the remaining agents have been moved nearer to their destination after the execution of the program, which satisfies the above **Requirement 3**.

We have chosen the $Bonus$, $Speed\_UP$, and C_T parameters as 3000.0, 80.0 and 100.0, respectively. Since we use the tournament selection, (see Table 19.2), the absolute values of these parameters are not important. However, the $Bonus$ parameter should be larger than the second term of the equation (19.3), so that GP searches for a program which completes the task at first. The task completion is defined as follows:

1. For the robot navigation task, each agent reaches its own goal.

2. For the Tile World task, all tiles are put into the holes.

19.4 Comparison with Reinforcement Learning

This section compares the performance of multi-agent Q-learning and GP for the robot navigation domain (see Appendix 19.A for the details of multi-agent Q-learning).

Let us take a straightforward approach for the navigation task (Fig.19.1(a)). The simple table entry x for Q-table, i.e., Q_large, would be as follows:

$$x \in \{(me, g_x, g_y, a_{1x}, a_{1y}, a_{2x}, a_{2y}, a_{3x}, a_{3y}, last)\}, \tag{19.4}$$

where me is the agent number (i.e., 0,1,2,3), $\binom{gx}{gy}$ is a relative vector to its own goal, and $\binom{a_{ix}}{a_{iy}}$ is a relative vector to the other three agents. The variables gx, gy, a_{ix}, a_{iy} range from -9 to +9. $last$ is the last move of the agent, i.e., STAY, RIGHT, LEFT, UP, and DOWN. Therefore, the size of this state space is

$$5 \times 4 \times 18^8 \approx 10^{11}. \tag{19.5}$$

This number is so huge for the practical computer resources that Q-learning often fails in the task for many agents. [Ono97] proposed a remedy called Modular Q-learning, in which each learning modular focused on one agent and its particular partner. However, the effectiveness of this method relies on the modular decomposability of the problem, which is not known beforehand.

An alternative way is to shrink the search space. For instance, since the minimum distance seems more important for avoiding the collision, we can choose the following state representation Q_small:

$$x \in \{(g_x, g_y, amin_x, amin_y, last)\}, \tag{19.6}$$

where $(amin_x, amin_y)$ is a relative vector to the nearest agent. The same sort of perceptual representation was chosen for a different problem [Tan93]. However, this state space representation requires the problem-specific knowledge, which is not always available.

During each trial, a sequence of actions is executed so as to update Q-table. The trial is terminated either when the task is completed, i.e., when all agents are moved to their goals, or when a fixed number of actions are tried. For the sake of comparison, we set this number to be 51, which is equal to the $Evals$ value described in Section 2. After each trial, all agents are displaced at the initial positions and the next trial is started again with the updated Q-table. One trial in Q-learning corresponds to one fitness evaluation of a GP individual. Thus, the Q-learning performance of N trials is comparable with that of GP at the generation of g, provided that the following equation is satisfied:

Table 19.2
The Experimental Setup for sgpc1.1.

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

$$N = pop_{size} \times g, \tag{19.7}$$

where pop_{size} is the population size of GP. We regard N/pop_{size} as the generation of Q-learning.

With these preparations, we have conducted the experiment with the previous scenario RN1 by using GP and Q-learning. The Q-learning parameters were set at $\beta = 0.8$, $\gamma = 0.9$, and $T = 0.4$ (see [Tan93] for details). Fig.19.3 plots the best standard fitness (i.e., that which was derived from eqs.(19.1),(19.2), and (19.3)) with generations, averaged over 20 runs. The fitness values of GP-based agents are also shown. We used the heterogeneous breeding by GP with the parameters shown in Table19.2. Q-learning (Homo.) represents that agents share the Q-table, whereas each agent uses a different Q-table in case of Q-learning (Hetero.). As can be seen from the figure, the agents gave poorer results with Q-learning. At most three agents learned to reach their goals with Q-learning. We played with Q-learning parameters in several other ways, only to observe the similar result.

The difficulty of Q-learning seems to be due to the above-mentioned representational issue. The appropriate design of the state space is difficult for this task because of its high dimensionality. Although the total set of sensor inputs, i.e., GP terminals, is provided, yet it is not easy to distinguish between essential terminals and useless ones. A part of terminals may be combined to construct an effective Q-table, but its combination is not known beforehand. On the other hand, in case of GP, essential terminals are expected to be adaptively chosen and functionally com-

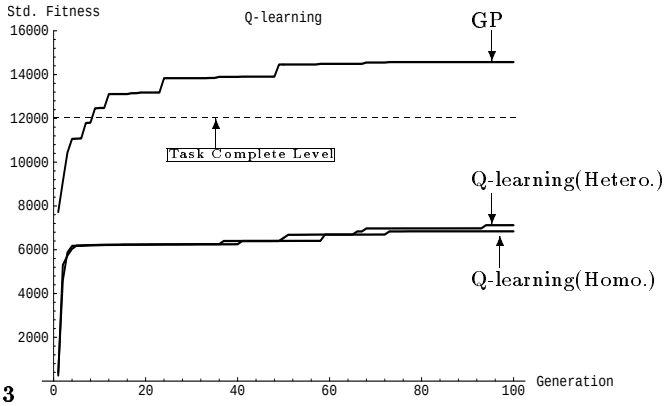


Figure 19.3
Experimental Result (Q-learning vs. GP).

bined as a result of the tree evolution. Thus, we think the superiority of GP-based multi-agent learning has been confirmed by this experiment.

19.5 Evolving Agents with Communication

19.5.1 Evolving Controlling Agents

For the navigation task, 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. The function symbols introduced for the communication are shown in Table 19.3. 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.

We have conducted comparative experiments so as to confirm the effectiveness of the communication. The heterogeneous strategy was applied for evolving agents without communication (i.e., with GP functions and terminals shown in Table 19.1) and for evolving communicating agents (i.e., Table 19.1 and Table 19.3). The used

Table 19.3
GP Functions for Communication (Controlling Agents).

Name	#Args.	Description
Send_i	2	Send its first argument to the i-th nearest agent as a command. Return its second argument.
Send_iY	1	Send the YIELD command to the i-th nearest agent. Return its argument.
Send_iS	1	Send the STOP command to the i-th nearest agent. Return its argument.
Send_iR	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.

parameters are shown in Table 19.2. We chose 6 training and 3 testing scenarios, which are similar to Fig.19.1(b). They were modified to constitute a variety of examples in several ways, by 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.

Fig.19.4(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 following programs were acquired in one run:

```
Agent0: (if>= Ag1 V1 Goal (inv Goal))
Agent1: (if>= Ag1 (*2 V1) Goal (inv Goal))
Agent2: (if>= Ag1 (*2 (*2 V1)) Goal (inv Goal))
Agent3: (if>= Ag1 V1 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>= Ag1 ***)).

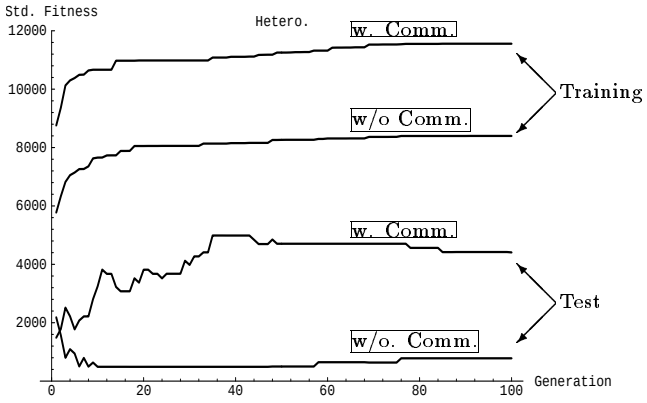


Figure 19.4
Experimental Results (Robot Navigation Task).

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.

On the other hand, agents with communication were able to solve the various tasks. For instance, the programs shown in Table 19.4 were acquired in one run at the generation of 56. These programs scored the standard fitness of 13520.0, which means that they solved most of the training cases. Fig.19.5 shows the emergent behavior of these agents for Training #3. The key features of these programs are as follows:

1. Agent0 and Agent2 are receivers, which always give way if they receive a message.

2. Agent3 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_Y Goal) ***) always returns the third argument (Send1_Y Goal).

3. Agent1 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 Step9 to Step19 (Fig.19.5), 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.

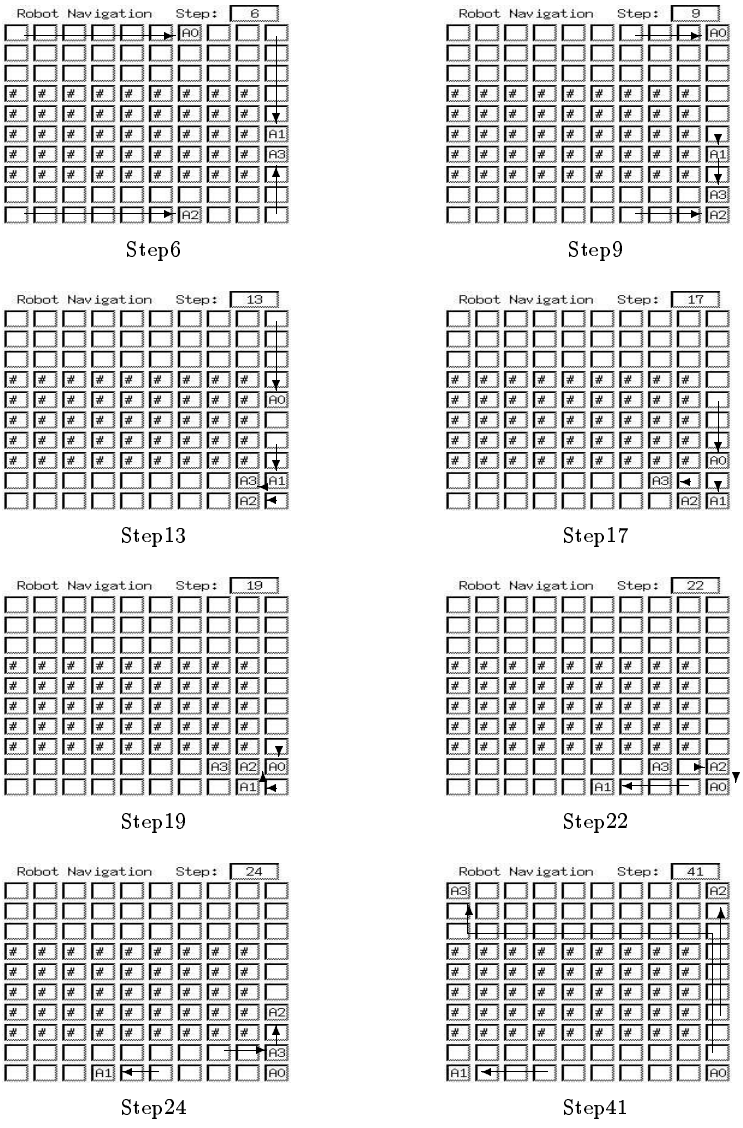


Figure 19.5
Emergent Behavior with Communication.

Table 19.4
Acquired Trees for Heterogeneous Strategy.

Agent Name	Tree
Agent0:	(Receive Goal)
Agent1:	(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)
Agent2:	(Receive Goal)
Agent3:	(if>= (if>= Ag1 V1 (Receive Goal) V1) V1 (Send1_Y Goal) (if>= Ag1 Goal Goal (Receive Goal)))

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.

19.5.2 Evolving Negotiating Agents

A number of researchers in DAI use various protocols developed in economics and game theory to evaluate the multi-agent interaction. In the simplest case, the agent requesting a service offers a specific reward for the completion of a task. The task may be completed by a set of agents, who need to negotiate how to divide the reward. Dividing the total amount equally might not be fair if the agents made different contributions. If there are many agents (or sets of agents) that may complete the task, the requester might try to minimize its cost by seeking multiple bids or holding an auction. There are a number of alternatives that have different properties and may be applicable or preferred in different situations [Genesereth *et al.*97].

We use ACL (Agent Communication Language) for the Tile World in order to evolve the negotiation among agents. ACL consists of three parts, i.e., its vocabulary, an inner language called KIF (Knowledge Interchange Format), and an outer language called KQML (Knowledge Query and Manipulation Language). The vocabulary of ACL is listed in a large and open-ended dictionary of words appropriate to common application areas. Each word in the dictionary has an English description used by humans in understanding the meaning of the word. The dictionary is open-ended to allow for the addition of new words within existing areas and in new application areas. Full specifications are available, and parts of the language are making their way through various standards organizations (see [Genesereth *et al.*97] for details).

Now consider the modified Tile World shown in Fig.19.2(b). Agent 0's goal is to fill hole H0, while agent 1's and 2's need to fill holes H1 and H2, respectively. To fill its hole, agent 1 needs to do 17 steps. Agents can cooperate and help each other to reduce the cost of achieving their goals. There are several kinds of joint plans that the agents can execute for the reduction.

We use ACL-like commands shown in Table 19.5 for agents to negotiate with each other. The experimental results have shown that the effective communication emerged among the agents so that they negotiate with each other to reduce the cost of achieving their own goals. For instance, the programs shown in Table 19.6 were acquired in one run. We observed that Agent0 and Agent2 negotiated with each other while Agent1 tried to achieve its goal by itself.

The above result shows that GP has formed a 3-agent coalition, i.e., {{Agent0, Agent2}, {Agent1}}. This coalition structure means that there are two coalitions, one consisting of Agent0 and Agent2, and the other consisting only of Agent1. When two agents form a coalition, they are coordinating their actions. The utility of an agent from a joint plan that achieves his goal is defined as the difference between the cost of achieving his goal alone and the cost of his part of the joint plan [Zlotkin *et al.*91]. In general, n-agent coordination mechanisms can be used when any sub-group of agents may engage in task exchange to the exclusion of others. [Zlotkin *et al.*94] showed that a simple and rational coalition was formed by means of cryptographic techniques. However, this method assumes certain characteristics of the task domain, i.e., sub-additivity. The sub-additive task domain means that, by combining sets of tasks, we may reduce (and can never increase) the total cost, as compared with the sum of the costs of achieving the sets separately. Unfortunately, many task domains do not have this property. In some case, there is no knowing whether the domain has this property or not. The above experimental result has shown that a certain coalition was established by means of GP, which requires no particular conditions for the coalition formation. Therefore, we believe GP-based multi-agent learning can provide an effective coordination method for a broader range of task domains. Further research will consider the applicability and feasibility of GP in this direction.

19.6 Discussion

19.6.1 Evolving Other Types of Communicating Agents

In this paper, we introduced communication commands of types B for the robot navigation domain (Section 5.1) and C for the Tile World (Section 5.2). Other types of communication can be defined as well. For example, [Werner *et al.*91] presented a simulation, in which a population of artificial organisms evolved low-

Table 19.5
GP Functions for Communication (Negotiating Agents).

Name	#Args.	Description
Propose_i	1	Send a proposal to the i-th nearest agent. The proposal is that the sender gives some of the sender's reward (the ratio is given by the first argument) to the receiver if the receiver achieves the sender's goal. Return the Tile vector.
Accept_i	0	Accept the i-th nearest agent's proposal and send the acceptance message. Return the Hole vector.
Reject_i	0	Reject the i-th nearest agent's proposal and send the rejection message. Return the Tile vector.

Table 19.6
Acquired Trees for Negotiating Agents.

Agent Name	Tree
Agent0:	(+ (inv (if>= last (- (+ (if>= (Propose_2 0.3259680) last (Accept_2) (if>= (Accept_1) (Propose_2 0.3260016) (Reject_2) Ag0)) (Accept_2)) (- Ag1 (if>= last (if>= Ag0 (Reject_0) (Reject_2) (Reject_0)) (Propose_0 0.3261200) (Propose_0 0.217209)))) (- (- (Accept_0) (Accept_2)) (Accept_0)) (+ Hole (+ Tile (+ Hole (- Ag1 (Reject_0))))))) (->90 Tile))
Agent1:	(if_dot Tile (if_dot last (->90 (if>= (+ Ag1 Ag0) Tile Hole Ag1)) (inv (Reject_1)) Hole) [-1.076370,-0.553187] last)
Agent2:	(if_dot (if_dot Ag1 Tile Ag2 (inv (if>= Ag0 (if>= (Propose_0 0.3563) Ag1 (+ (if>= (Reject_0) Ag0 Ag0 Ag1) Ag0) (if>= (if>= (Reject_0) (+ Ag1 (Propose_2 0.02550)) (- (Propose_0 0.979331) (Accept_0)) (- Ag0 (Accept_0))) (Accept_0) (if>= (Propose_1 0.541944) Ag0 (if>= (Accept_0) (Reject_0) (Propose_0 0.582183) (Accept_0)) (Propose_0 0.918706)) (Propose_2 0.235540)) (Accept_1)) (+ Ag1 (+ (if>= Ag1 (Propose_1 0.980090) (if>= (Propose_1 0.289795) Ag0 (Propose_1 0.000803) Ag0) (Reject_0)) (if>= (+ (Accept_1) (Accept_1)) (Reject_0) (if>= (Reject_0) (Propose_1 0.093556) (Accept_0) (Accept_0)) (if>= (Propose_1 0.966526) (Propose_1 0.947458) (Reject_0) Ag1))))))) Ag2 (*2 (/2 Tile)))

level communication protocols for mate finding. They showed how the organisms generated and interpreted meaningful signals as a result of evolution. For the pursuit problem, we used a communication command of type A and empirically confirmed the evolvability of the communicative cooperation of this type [Iba97]. In this experiment, agents were assumed to have a limited visibility. In most cases, agents without communication were unable even to get closer to the prey. On the other hand, the communicating agents succeeded in enclosing the prey.

The effective communication and its learnability for a specific task remain to be seen in DAI. We have shown the emergence of communication among agents so as to increase the robustness of a generated GP program. The evolvability of various types of communication for other problem domains is our future research concern.

19.6.2 Q-learning and Genetic Programming

The previous result has shown the difficulty with Q-learning approach for the multi-agent task. The state space for Q-table may become so huge in case of many agents. This paper presented a GP-based approach to avoid the above problem. We have also proposed an extension of multi-agent reinforcement learning with GP. This is to integrate GP-based adaptive search with Q-learning so as to shrink the search space. The established system is called QGP, i.e., Q-learning with Genetic Programming. The salient features of QGP are as follows:

1. GP searches for the combination of input variables, i.e., constructs an effective Q-table entry.

2. Q-learning updates Q-values for each Q-table, i.e., a GP individual. These values are inherited from generation to generation.

3. The fitness of GP is derived from the reinforcement reward given by Q-learning.

We showed how successfully QGP was applied to the robot navigation task [Iba98]. Our QGP is based upon a selectionist approach [Steels97], in the sense that a structure comes into existence by variation or construction and is tested as a whole for the fitness (i.e., the reinforcement reward) in the environment. We believe that QGP leads to the effective integration of two learning paradigms, i.e., Q-learning and GP.

19.6.3 Related Work

[Teller94] realized an indexed memory in GP and studied the evolution of agents with mental models. He evolved programs that could solve a problem of pushing blocks up against the boundaries of a world. [Andre95] extended this idea to realize MAPMAKER, a method for the automatic generation of agents that discovered

information about their environment, encoded this information for later use, and created simple plans utilizing the stored mental models. He applied his method to a "gold" collection problem, in which one part of a program made a map of the world and stored it in memory, and the other part used this map to find the gold. These works are closely related to our robot navigation domain, because both tasks require the effective map making and its usage. However, their work did not necessarily aim at the multi-agent cooperation.

As for multi-agent learning, 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 simultaneous 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 [Koza 92]. [Fogarty *et al.*95] 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 [Haynes *et al.*95],[Haynes *et al.*96]. 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 *et al.*96] 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. The terminal and function symbols in Table 19.1 have been partly motivated by the study reported by this work. They studied evolving a teamwork by GP for the pursuit game, in which the world is a continuous 2-dimensional area.

19.7 Conclusion

This paper described the emergence of cooperative behavior based on GP and showed the following points:

1. The effectiveness of GP-based method was shown by the comparative experiment with Q-learning.

2. GP was successfully applied to multi–agent test beds, i.e., the robot navigation task and the Tile World. We have confirmed that the cooperative behavior emerged

by means of GP.

3. The evolvability of communicative cooperation has been shown. The robustness of a generated GP program was increased with the emergent communication.

The experimental results showed the effective emergence of cooperation in many difficult situations. A more general way to validate the evolutionary scheme is our future concern. Another important topic is to apply our approach to a real-world problem. We have attempted to evolve a robust robot programming in a real world situation [Ito *et al.*96]. This research focused on the robustness of a single robot. We are currently working on the extension of this framework to realize the evolution of a group of robots.

19.A Multi-agent reinforcement learning

This appendix briefly explains multi-agent reinforcement learning (see [Tan93] for details).

Each agent uses the one-step Q-learning algorithm. Given a current state x and an available actions a_i, a Q-learning agent selects each action a with a probability given by the Boltzmann distribution:

$$p(a_i \mid x) = \frac{e^{Q(x,a_i)/T}}{\sum_{k \in actions} e^{Q(x,a_i)/T}}, \tag{19.8}$$

where T is the temperature parameter that adjusts the randomness of decisions. The agent then executes the action, receives an immediate reward r, moves to the next state y.

In each time step, the agent updates $Q(x,a)$ by recursively discounting future utilities and weighting them by a positive learning rate β:

$$Q(x,a) \Longleftarrow Q(x,a) + \beta(r + \gamma V(y) - Q(x,a)). \tag{19.9}$$

Where $\gamma (0 \leq \gamma < 1)$ is a discount parameter, and $V(x)$ is given by:

$$V(x) = \max_{b \in actions} Q(x,b). \tag{19.10}$$

$Q(x,a)$ is updated only when taking action a from state x. Selecting actions stochastically by $p(a_i \mid x)$ ensures that each action will be evaluated repeatedly. In general, the $Q(x,a)$ value is kept as a multi-dimensional table, which we call a Q-table. Two types of multi-agent reinforcement learning algorithms have been proposed [Weiβ93], i.e., heterogeneous learning and homogeneous learning. In the former case, each agent uses and updates a different Q-table, whereas all agents share a common Q-table in the latter.

For the robot navigation task, the reward r is defined by following the same principle described in Section 3. According to **Requirement 1**, the bonus reward is given when an agent is moved to a goal. When the task is completed, the reward proportional to the remaining time steps is provided, which meets **Requirement 2**. If an action causes an agent to get closer to its goal, a little reward is given, which satisfies **Requirement 3**.

Bibliography

Andre,D., The Automatic Programming of Agents that Learn Mental Models and Create Simple Plans of Action, in *Proc. of the 14th International Joint Conference on Artificial Intelligence*, 1995

Benda,M., Jagannathan,V., and Dodhiawalla,R., On Optimal Cooperation of Knowledge Sources, in *Proceedings of the 1988 Workshop on Distributed Artificial Intelligence*, May 1988

Bull,L., Evolutionary Computing in Multi-agent Environments: Partners, in *Proc. of the 7th International Conference on Genetic Algorithms (ICGA97)*, 1997

Chu-Carroll,J., and Carberry,S., Communicating for Conflict Resolution in Multi-Agent Collaborative Planning, in *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS95)*, 1995

Fogarty,T., Bull,L., and Carse,B., Evolving Multi-Agent Systems, in *Genetic Algorithms in Engineering and Computer Science*, Winter G., Pèriaux,J., Galàn,M. and Cuesta,P. (eds), John Wiley & Sons, 1995

Genesereth,R., and Ketchpel,S.P., Software Agents, 1997

Goldman,C.V. and Rosenshein,J.S. Emergent Coordination through the use of Cooperative State-Changing Rules, in *Proceedings of the 12th National Conference on Artificial Intelligence*, 1994

Hanks,S., Pollack,M.E., and Cohen,P.R., Benchmarks, Test Beds, Controlled Experimentation, and the Design of Agent Architectures, In *AI Magazine*, Winter, 1993

Haynes, T., Wainwright,R., and Sen,S., Evolving a Team, In *Working Notes of the AAAI-95 Fall Symposium on Genetic Programming*, AAAI Press, 1995

Haynes, T. and Sen,S., Learning Cases to Compliment Rules for Conflict Resolution in Multiagent Systems, In *International Journal of Human Computer Studies*, 1996

Iba,H., Emergent Cooperation for Multiple Agents using Genetic Programming, in *Parallel Problem Solving form Nature IV (PPSN96)*, 1996

Iba,H., Multiple-Agent Learning for Robot Navigation Task by Genetic Programming, *Genetic Programming 1997 (GP97)*, 1997

Iba,H., Multiple-Agent Reinforcement Learning with Genetic Programming, *Genetic Programming 1998 (GP98)*, 1998

Ito,T., Iba,H. and Kimura,M., Robot Programs Generated by Genetic Programming, Japan Advanced Institute of Science and Technology, IS-RR-96-0001I, in *Genetic Programming 96*, 1996

Rosca,J.., Ballard,D.H., Discovery of Subroutines in Genetic Programming, in *Advances in Genetic Programming II*, MIT Press, 1996

Koza, J., Genetic Programming, On the Programming of Computers by means of Natural Selection, MIT Press, 1992

Koza, J., Genetic Programming II, Automatic Discovery of Reusable Programs, MIT Press, 1994

Luke,S. and Spector,L., Evolving Teamwork and Coordination with Genetic Programming, in Genetic Programming 96, MIT Press, 1996

Steels,L., Perceptually Grounded Meaning Creation, in *Proc. Second International Conference on Multi-Agent Systems (ICMAS96)*, 1997

Ono,N., Fukumono,K., A Modular Approach to Multi-Agent Reinforcement Learning, in *Distributed Artificial Intelligence Meets Machine Learning*, Weiβ,G. (ed.), Springer, 1997

Pollack,M.E. and Ringuette,M., Introducing the Tileworld: Experimentally Evaluating Agent Architectures, in *Proceedings of the 8th National Conference on Artificial Intelligence*, 1990

Sen,S. and Sekaran,M., Multiagent Coordination with Learning Classifier Systems, in *Adaption and Learning in Multi-Agent Systems*, Weiβ,G. and Sen,S. (eds.), Springer, 1995

Tan,M., Multi-Agent Reinforcement Learning: Independent vs. Cooperative Agents, in *Proceedings of the Tenth International Conference on Machine Learning*, pp.330–337, 1993

Teller,A., The Evolution of Mental Models, in *Advances in Genetic Programming*, 1994

Weiβ,G., Learning to Coordinate Actions in Multi-Agent Systems, in *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pp.311–316, 1993

Werner,G.M. and Dyer,M.G., Evolution of Communication in Artificial Organisms, in *Artificial Life II*, Langton,C.G., Taylor,C., Farmer,J.D. and Rasmussen,S. (eds.), Addison-Wesley, 1991

Zlotkin,G., and Rosenschein,J.S., Cooperation and Conflict Resolution via Negotiation among Autonomous Agents in Noncooperative Domains, in *IEEE Transactions on Systems, Man and Cybernetics*, 21(6), pp.1317-1324, 1991

Zlotkin,G., and Rosenschein,J.S., Coalition, Cryptography, and Stability: Mechanisms for Coalition Formation in Task Oriented Domains, in *Proc. 12th National Conference on Artificial Intelligence (AAAI94)*, 1994