A Study of Genetic Programming on Cooperative Model for an Ant Colony

Yueh-Hung Chen
Institute of Computer Science and Information Education
National Tainan Teachers College
m87518@stu.ntntc.edu.tw

Koun-Tem Sun
Institute of Computer Science and Information Education
National Tainan Teachers College
ktsun@ipx.ntntc.edu.tw

Abstract

A cooperative model is the action strategy in which every individual cooperates with others in a team. Thus, the better cooperative model the team has, the higher performance it has. However, every individual in the team can not detect all state of others. It only detects the change of the surroundings, and all individuals in the team are act independently. So, it is difficult to find an efficient cooperative model by human being. The genetic programming (GP) can program automatically by simulating the evolutionary mechanism and process on computers. When a program evolves using the GP technique, an efficient program often can be generated for solving a particular problem that the human being doesn't understand in detail how to solve it. So, in this research, we try to use GP to evolve the cooperative model for the ant colony under both the single program architecture and the multi-agent architecture. Then, ants can efficiently transport the food to the nest under the generated model. It is proved that the GP technique can be used to evolve an efficient cooperative model. This research not only can be used to related research, but also can be applied to develop more complex cooperative models such as the cooperative models for robots and cooperative learning model.

1. Introduction

When observing the cooperative strategy of a good teamwork, it can be found that "emergent behavior" arises in the team when every individual in this team cooperates with others in certain strategy to complete a task efficiently. The "emergent behavior" is a complex and integrated behavior arising in a team consisting of many simple individuals when applying seemingly simple rules to every individual in this team repetitively [1]. So, an appropriate cooperative model is a set of rules on which the group of individuals can show expectable emergent behavior. However, it is sometime difficult to find a set of apropos rules for all individuals of the team such that expectable emergent behavior appear in it. Because all individuals in the team are act independently at the same time and they can only detect the change of the surroundings, consequently it is hard to predict the overall behavior of the team at the stage of plan. Nonetheless, there are many cooperative model and emergent behavior in nature and in computer science, for example, central place foraging behavior of the ant colony and majority classification problem of the cellular automata [1]. Therefore, when researching emergent behavior or cooperative model, it is important to develop an efficient strategy to find the appropriate rules on which the whole team can show expectable emergent behavior.

When developing the best cooperative models for the team, it is convenient to solve this difficulty for predicting the emergent behavior by using genetic programming (GP) [9, 10]. Genetic programming is a new research area of computer science. It provides a new approach for computer to solve the problem by itself. It has been proved by a variety of researches that the genetic programming can be used to evolve programs for solving different applications, and the performance of the programs evolved by GP are very well. Up to the present, the performance of many programs evolved by GP is supervisor to that of algorithms written by knowledgeable human investigator [1].

Genetic programming is based on genetic algorithm [6, 7, 12, 13], and its objective is to evolve a program to solve the given problem. The set of programs, called program space, in which all programs can be constructed from basic components, is similar to the search space of genetic algorithm. So, how to take advantage of the mechanics of natural selection and natural genetics, and how to make use of the operations of crossover, mutation and reproduction to find the fittest program in program space, are mainly investigated in the region of genetic programming. The characteristics of GP are that only a few related knowledge of the specific problem is necessary when a program evolved using the GP. So, when facing a difficult problem of which only a few knowledge is available (e.g., evolving a set of rules to make expectable emergent behavior appear, or automated discovery of an analog electrical circuit for a difficult-to-design asymmetric band-pass filter [11]), this

characteristic is very useful.

So, in this research, we try to use GP with simple functions and terminals to evolve the cooperative model for the ant colony (e.g., a program controlling each ant in the ant colony). As for the architecture of cooperative model, we try to respectively evolve the cooperative model under the single program architecture and the multi-agent architecture. The cooperative model of single program architecture is a cooperative model for the ant colony only consisting of one program tree. This program tree detects states of the surroundings and the ant itself, and than makes this ant do an appropriate action at every time step. The cooperative model of multi-agent architecture is a cooperative model for the ant colony consisting of more than one program tree (called agent). Every agent detects states of the surroundings and the ant itself, and then suggests an appropriate action. Then, the ant performs all actions under a specific order. For the multi-agent architecture, the interaction among all agents can be viewed as another kind of emergent behavior. In this research, we also compare these two architectures and suggest a better architecture for developing cooperative model. It will be very useful for developing the cooperative models and related searches by using GA.

2. Genetic Programming

Genetic programming (GP) is an extension of the conventional genetic algorithm in which each individual in the population is a computer program. To evolve the program for a specific application is the object of the genetic programming. The operations of GA: crossover, mutation and reproduction are also used in GP to find the fittest program for the specific application among all possible programs. So, the fitness of the program can be defined by the accuracy and the performance of it.

For a program, it can be represented with the tree structure (as shown in Figure 1). Terminal nodes are the variables or constants of the program, and internal nodes are operators or functions of the program. Internal nodes return the value computed with the values of its' children to the parent, or choose one of the sub-trees to execute.

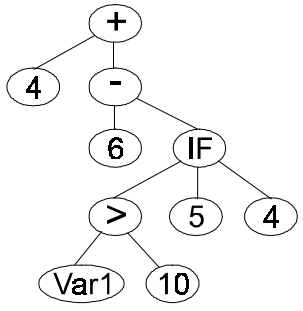

Figure 1. A simple example of program tree.

So, before evolving programs by using genetic programming, we should determine function set and terminal set of the program tree first. The functions contained in function set should operate with one or more arguments. They are always the internal nodes of the program tree and compute values returned from its' sub-trees or its' arguments. Terminals contained in terminal set are variables, constants or functions that do not need arguments, and they are always terminal nodes (e.g., leaves) of the program tree.

Just like the genetic algorithm, the fitness functions are different for various applications and they effect the direction of evolution of the population. The better definition of fitness function leads to a better performance of evolved programs and a shorter period of evolution.

The genetic programming contains three important operations to generate the offspring. They are crossover, mutation and reproduction. These operations are described below:

(1) crossover: This operation randomly select a sub-tree from two program trees respectively and then swap these two selected sub-trees and generate two new program trees (offspring). (as shown in Figure 2)

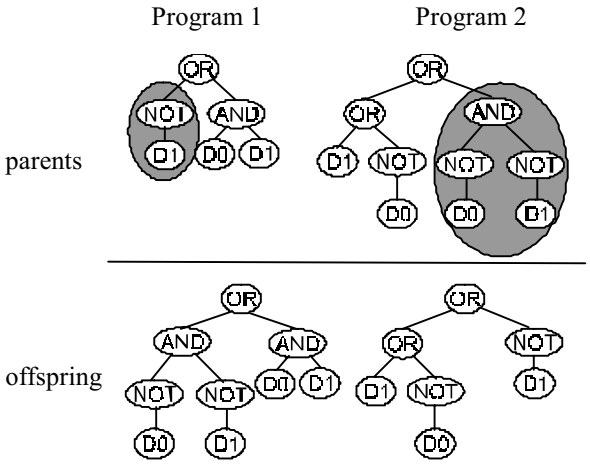

Figure 2. A simple sample of crossover for programs 1 and 2.

(2) reproduction: This operation directly copy a program tree to generate a new offspring. The parent and the offspring are all the same.

(3) mutation: This operation randomly select a sub-tree from the program tree, then replace this sub-tree with a new sub-tree generated randomly. (as shown in Figure 3)

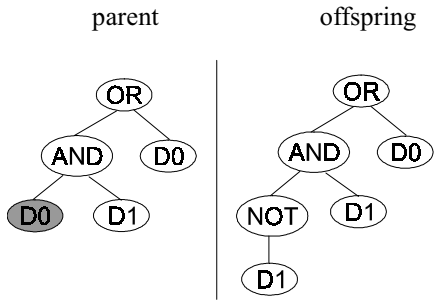

Figure 3. A simple sample of mutation on the terminal node "D0."

In summary, the process of genetic programming is composed of three steps:

(1) Initialize a population of programs, of which every program is randomly generated. The process of randomly generating a program tree is to select a function from the function set or a terminal from the terminal set randomly. If a function is selected, then it's sub-trees are recursively generated as before.

(2) Iteratively perform the following sub-steps until the termination criterion has been satisfied:
   (a) Execute each program in the population and assign it a fitness value using the fitness function.
   (b) Create a new population of computer programs by choosing the "parent" programs with a probability value and then applying the operations: crossover, mutation and reproduction for generating the offspring.

(3) Among all generations, the program having the best fitness is the result of the genetic programming for this run.

3. Multi-Agent Architecture

Multi-agent architecture is also a new research region of artificial intelligent. On this architecture, an intelligent individual is composed of many agents that can only do something simple independently. These agents consist in the individual on a special architecture so that the expectable overall behavior arises by the cooperation of all agents to show the intelligent behavior.

The definition of the whole multi-agent system is as follow [5]:

(1) An environment, E, that is, a space which generally has a volume.

(2) A set of objects, O. These objects are situated, that is to say, it is possible at a given moment to associate any object with a position in E. These objects are passive. That is, they can be perceived, created, destroyed and modified by the agents.

(3) An assembly set of agents, A, which are specific objects ($A \subseteq O$), representing the active entities of the system.

(4) An assembly set of relation, R, which link objects (agents) to each other.

(5) An assembly of operations, OP, making it possible for the agents of A to perceive, produce, consume, transform and manipulate objects from O.

(6) Operators with the task of representing the application of these operations and the reaction of the world to this attempt at modification, which we should call the lows of the universe.

In this research, not only the cooperative models for the ant colony under the single program architecture (i.e., every ant is controlled by a single program tree) are evolved, but it is attempted to evolve cooperative models under the multi-agent architecture. With the multi-agent architecture, a cooperative model consist of more then one program tree, and the behavior of an ant is the result from the interaction of all agents of the cooperative model. By using multi-agent architecture, we hope that the simulated ants can be more similar to the real ants. Moreover, when complex cooperative models are evolved, cooperative models under the multi-agent architecture can be more complex than that under the single program architecture.

4. Central Place Foraging Problem

For evolving cooperative models by using genetic programming, we choose central place foraging problem [3] as an example. In physical world, the real ants forage for foods cooperatively. The ant finds food source and leaves pheromone on the ground while returning to the nest, thereby creating a trail of pheromone between the food source and the nest. When other ants detect the odor of this pheromone, they respond by following the pheromone trail, finding the food source and join in transforming the foods to the nest [8]. This process is a complex cooperative behavior, but the operation of every ant is relative simple. The organization of real ants can lead to the cooperation of them by detecting the states of surroundings. Moreover, the relative simple rules of an ant in the ant colony bring about the complex emergent behavior. The objective of his research is to evolving a cooperative model for an ant colony, and by applying the evolved cooperative model to every ants of the ant colony, it is hoped that ants can forage for foods efficiently. To solving the central place foraging problem of this research, the ants can choose to complete the work alone rather then cooperate with others. However, the performance of the ant colony with better cooperative model is better (i.e., transporting more foods in shorter time). Indeed, it is the objective of this research.

The ant colony world consists of a 32 by 32 toroidal grid. A 3 by 3 nest is centered at grid location (20, 11) (an

example, the location of the nest can be assigned optionally). An ant carrying a food will put the food down automatically when moving into the nest. There are a total of 144 food pellets located in two 3 by 3 food areas and each of them have 72 food pellets (i.e., 8 food pellets in a grid). One food area is located at (5, 17) and the other is located at (15, 29) (the locations of the food areas can be also assigned optionally). The ant colony consists of 20 ants. At first, each ant is randomly located within the nest and face in a random direction. Each ant can only transport one food pellet at a time step. Because the ant colony world is toroidal, the ant wandering off the edge will reappear on the opposite edge. The pheromone decay linearly and disappears after 50 simulation time steps. The ant colony world is shown in Figure 4.

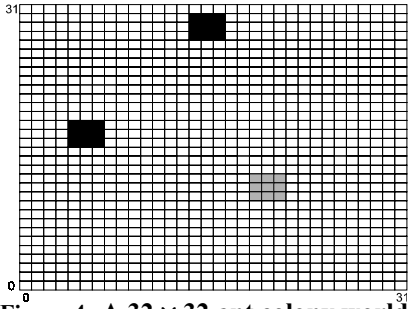

Figure 4. A 32 × 32 ant colony world.

In the ant colony world, every ant is moved under the same cooperative model, and it can only perform an action in every time step. The simulation (i.e., the evaluation for the certain cooperative model) is finished when all the food pellets are transported to the nest or the maximum number of time steps (4,000) is reached. Then, we can score for this cooperative model (i.e., the fitness of this cooperative model), and generate the next generation of the cooperative models by the GP's operators.

5. Evolving the Cooperative Model under the Single Program Architecture

For the single program architecture, a cooperative model is composed of a single program tree. At each time step of the simulation, each of the 20 ants in the colony determines the state of the surroundings and decides to do a certain appropriate action by evaluating the cooperative model for the ant colony. Moreover, NO-ACTION is excluded from the terminal set of cooperative model under the single program architecture. Table 1 lists the related information for the central-place foraging problem for an ant colony under the single program architecture. Use this configuration, we already evolved some effective cooperative model for the ant colony [4].

Table 1. related information for the central-place foraging problem for an ant colony under the single program architecture.

Function set	IF-FOOD-HERE， IF-FOOD-FORWARD， IF-SMELL-FOOD， IF-CARRYING-FOOD， IF-FACING-NEST， IF-PHEROMONE-HERE， IF-PHEROMONE-FORWARD， IF-SMELL-PHEROMONE， IF-NEST-HERE
Terminal set	MOVE-FORWARD， MOVE-RANDOM， TURN-LEFT，TURN-RIGHT， GRAB-FOOD， DROP-PHEROMONE-AND-MOVE
Fitness case	One fitness case
Fitness	$f(x) = (4000 - time)/10 + pellet + 1$ Where : time = the amount of time steps spent on the fitness evaluation 4000 is the maximum number of allotted time steps *pellet* = the amount of food pellets the ants transport to the nest
Hits	The number of food pellets that the ant colony has transported to the nest
Parameters	Population size M=100 Maximum number of generation to be run G=100 The percentages for the genetic operation are: 80% for crossover, 17% for reproduction, 3% for mutation

6. Evolving the Cooperative Model under the Multi-Agent Architecture

In addition to evolving the cooperative model under the single program architecture, the multi-agent architecture is also used to evolve the cooperative models for the ant colony. The cooperative model under the multi-agent architecture for the ant colony is composed of many program trees and each of them is viewed as a simple agent. So, in this research, it is hoped that the expectable emergent behavior of the whole agents appears — the ants can cooperatively and efficiently transport the food pellets to the nest.

The cooperative models under the multi-agent architecture are composed of many simple agents, and these agents are not in order or prioritized. Each agent is executed in parallel. An example is shown Figure 5.

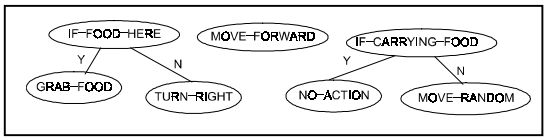

Figure 5. An example of multi-agent architecture with three agents.

For the multi-agent architecture, a cooperative model is executed when the evaluation begins or all actions suggested by the agents are finished. When the cooperative model is executed, each agent of the cooperative model proposes an action appropriate to the occasion. After all agents propose actions, the ant performs the suggested actions in the follow order: GRAB-FOOD, DROP-PHEROMONE-AND-MOVE, TURN-LEFT, TURN-RIGHT, MOVE-RANDOM, and MOVE-FORWARD. The ant spends one simulation time step for each action. Moreover, if there are more than one agent suggesting the same action, the action still be down once. As shown in Figure 5 , if this cooperative model is evaluated for an ant and this ant is not carrying food on a grid location with food, then the ant will performed the actions with the order: GRAB-FOOD, MOVE-RANDOM and MOVE-FORWARD. In this research, the cooperative model contains at most sex agents, and the size of an agent (i.e., amount of functions and terminals constructing the agent) is not limited. When evolving cooperative models under the multi-agent architecture, the operations of crossover and mutation are modified to operate properly. The operations are modified as fallows:

(1) crossover: This operation first choose two cooperative models in the population with a probability, and then randomly select an agent respectively from them to apply the operation of crossover as described in session 2.
(2) mutation: This operation first choose a cooperative model in the population with a probability, and then randomly select an agent to apply the operation of mutation as described in session 2.

In order to provide the genetic programming with the ability to modify the number of agents, when evolving cooperative models under the multi-agents architecture, two new operations are used to generate offspring. These two operations are as follow:

(1) Agent duplication operation: This operation randomly copies a agent from a selected cooperative model and inserts it into the same cooperative model. At this time, there are two identical agents in the selected cooperative model. However, the cooperative model has the chance to change the agents by using crossover and mutation operations. If the selected cooperative model contains the maximum number of agents, then do nothing.
(2) Agent deletion operation: This operation deletes an agent that is randomly selected from the cooperative model. If the selected cooperative model only has one agent, then do nothing.

The configuration of evolving cooperative models under the multi-agent architecture is the same as that of evolving cooperative model under the single program architecture except NO-ACTION terminal and some parameters. The parameters are as follow:

Population size, M = 100.
Maximum number of generation to be run, G = 100.
Maximum number of agents in an cooperative model is 6.
The percentages for the genetic operation are: 80% for crossover, 15% for reproduction, 1% for mutation, 3% for agent duplication, and 1% for agent deletion.

7. Simulations Results

The best cooperative model evolved under the single program architecture is shown in Figure 6. The fitness curves for evolving are shown in Figure 7, and the hit curves for evolving are shown in Figure 8. By using this cooperative model, the ant colony can cooperate efficiently. By observing the cooperation of the ant colony, it is found that the ants search for food pellets randomly, and if an ant finds food pellets, it will grab one food pellet and return to the nest along the shortest path. In addition, the ant will lay pheromones on the ground while returning to the nest. When other ants detect the pheromones on the ground, they can find food pellets fast by following the pheromone trail, and then join in transporting food pellets. So, the performance of the whole ant colony is efficient. However, if there is not any food pellet at the end of the pheromone trail, the ants following the pheromone trail will stay at the end of the pheromone trail until the pheromone decay entirely.

In addition, this cooperative model is capable of finding adjacent food pellets with the sense of smell. When the ant is adjacent to food pellets but not facing to the food pellets, it will change the direction it is facing and then move forward to the grid location of food pellets. It is also a reason of the high performance of this cooperative model.

The best cooperative model evolved under the multi-agent architecture is shown in Figure 9. The fitness curves for evolving are shown in Figure 10, and the hit curves for evolving are shown in Figure 11. All agents in Figure 9 are not complex, and when they operate alone, each of them can't lead the ant colony into an efficient cooperation. Even each of them can't lead the ant colony

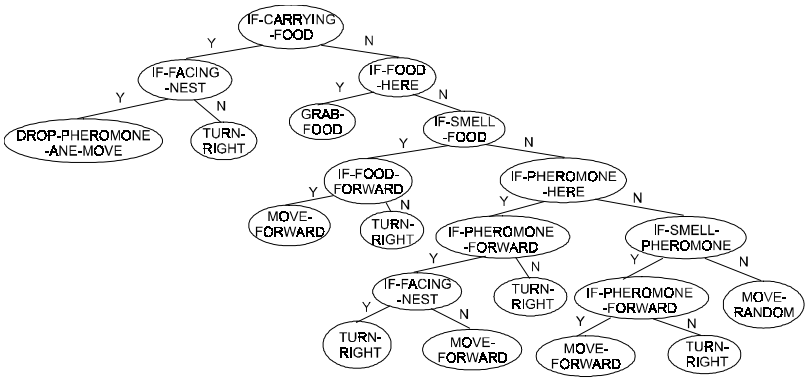

Figure 6. The best cooperative model evolved under the single program architecture.

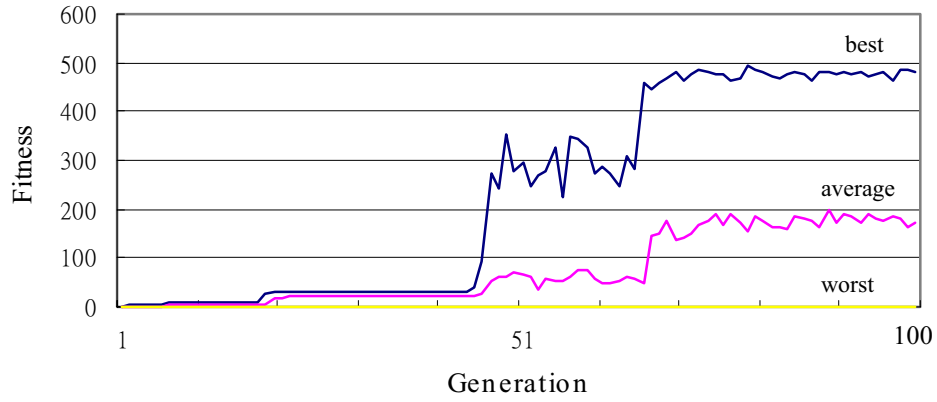

Figure 7. Fitness curves of the process of evolving cooperative model under the single program architecture.

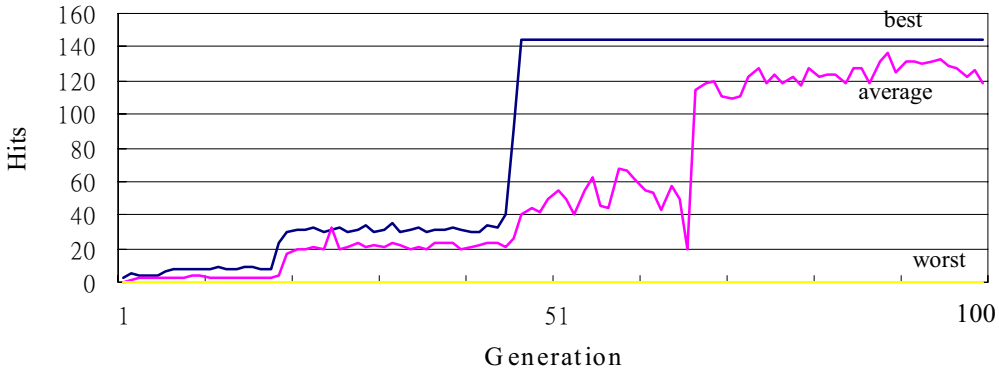

Figure 8. Hit curves of the process of evolving cooperative model under the single program architecture.

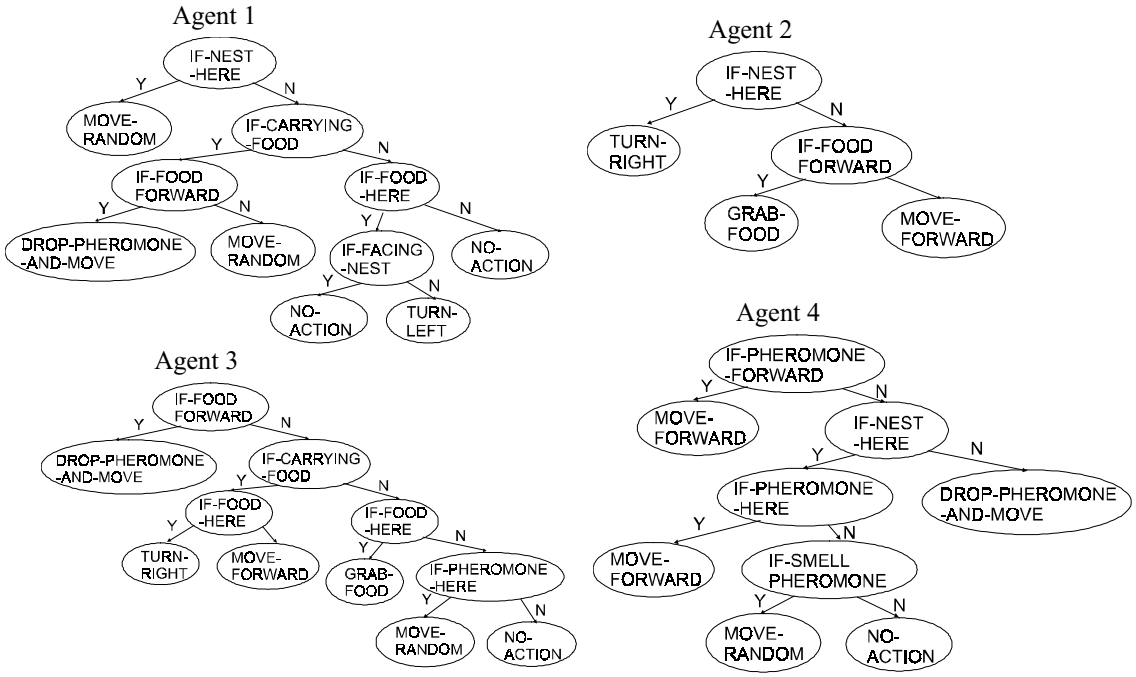

Figure 9. The best cooperative model evolved under the multi-agent architecture.

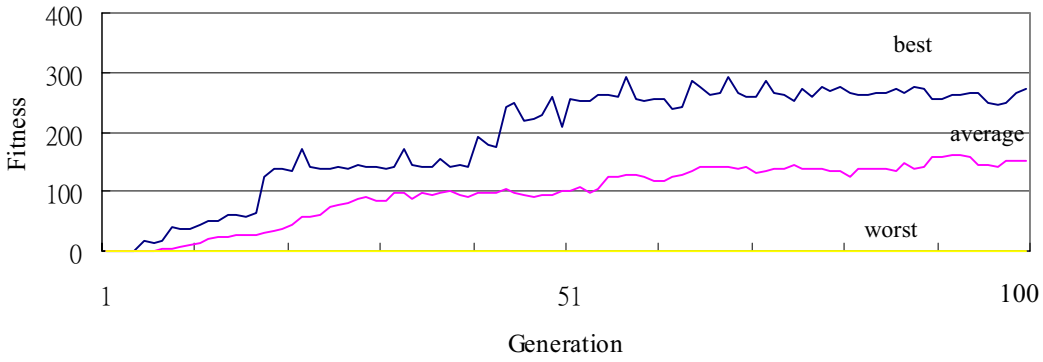

Figure 10. Fitness curves of the process of evolving cooperative model under the multi-agent architecture.

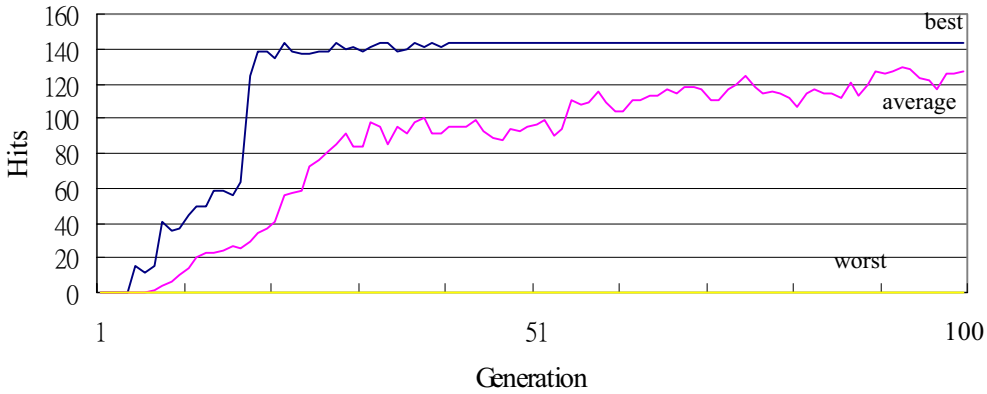

Figure 11. Hit curves of the process of evolving cooperative model under the multi-agent architecture.

into transporting any food pellets to the nest. But, when we organize them into a multi-agent system, the cooperative model can perform efficiently.

From the simulations results, it is found that the fitness value of the cooperative model evolved under the single program architecture is significantly better then that evolved under the multi-agent architecture. This result is involved with the architecture of the agents in this research. Under the multi-agent architecture, according to the states of the ants during executing the cooperative models, the agents propose actions to the occasion. However, other agents may change the states of the ants afterward. So, when the number of agents increases, the agents coordinate their results hard and it is harder by using GP to find a group of appropriate agents that can cooperate with others efficiently.

8. Conclusion

From the simulation results, an efficient cooperative model, which can let the ants transport 144 food pellets to the nest, is evolved under both the single program architecture and the multi-agent architecture. In this research, we also find that several local optimal solutions exist when cooperative models are evolved under both of the architectures. Moreover, the local optimal solutions are similar. All of them lead the ants into transporting the food pellets alone without taking advantage of pheromone. This problem may be improved by using related techniques of avoiding local optimal solutions, for example, pre-optimized initialization [14] and local search [2]. These techniques will be helpful when evolving cooperative models by using genetic programming.

In this research, cooperative models under the single program architecture can be view as a special case of cooperative models under the multi-agent architecture. In other words, a cooperative model under the single program architecture is a legal cooperative model under the multi-agent architecture. But in the process of evolving cooperative models under the multi-agent architecture, the population is evolved in the direction of multiple simple agents rather than in the direction of a single complex agent. When ants operate on cooperative model under the multi-agent architecture, all agents operate simultaneously. From the above properties, cooperative models under the multi-agent architecture are closer to real lives than these under the single program architecture. If an excellent architecture is used, it will be very useful to organize the agents for evolving more complex cooperative models.

Reference

[1] D. Andre, F. H Bennett Ⅲ, and J. R Koza, “Discovery by Genetic Programming of a Cellular Automata Rule that is Better than any Known Rule for the Majority Classification Problem,” *Proceedings of the First Annual Conference*, pp. 1-11, 1996.

[2] T. Bäck, D. B Fogel, and Z. Michalewicz, Handbook of Evolutionary Computation. Institute of Physics Publishing and Oxferd University press, 1997.

[3] F. H Bennett Ⅲ, “Automatic Creation of an Efficient Multi-Agent Architecture Using Genetic Programming with Architecture-Altering Operations,” *Proceedings of the First Annual Conference*, pp. 30-38, 1996.

[4] Y. H. Chen and K. T. Sun, “A Study of Genetic Programming on Foraging in an Ant Colony,” National Computer Symposium, vol. 2, pp. 229-236, 1999.

[5] J. Ferber, Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence, Addison Wesley Publishing Company, 1999.

[6] M. Gen and R. Cheng, Genetic Algorithms & Engineering Design, John Wiley & Sons, inc., 1997.

[7] J. H Holland, Adaptation in Natural and Artificial System, MIT Press, 1992.

[8] B. Holldobler and E. Wilson, The Ants, The Belknap Press of Harvard University Press, 1990

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

[10] J. R Koza, Genetic Programming Ⅱ: Automatic Discovery of Reusable Programs, MIT Press, 1994.

[11] J. R Koza, F. H Bennett Ⅲ, D. Andre, and M. A. Keane, “Automated WYWIWYG design of both the topology and component values of analog electrical circuits using genetic programming,” Genetic Programming: Proceedings of the First Annual Conference, pp. 123-131, 1996.

[12] K. F. Man, K. S. Tang, and S. Kwong, Genetic Algorithms, Springer-Verlag, 1999.

[13] M. Mitchell, An Introduction to Genetic Algorithms. MIT Press, 1998.

[14] G. R Raidl, “An Improved Genetic Algorithm for the Multiconstrained 0-1 Knapsack Problem,” *Proceedings of International Conference of Evolutionary Computation*, pp. 207-211, 1998.