

プログラム自動生成手法へのカオスニューロンの応用と効率化

Application of a Chaos Neuron for Automatic Programming and Improve of Efficiency

今福啓*

Kei Imafuku

Email: k03082@dokkyo.ac.jp

本研究では、コンピュータが目的となるプログラムを自動生成する手法の効率化を扱う。その際、ニューラルネットワークの局所解への収束を抑える、カオスニューロンを応用してプログラムを作成する手法を提案する。カオスニューロンは軌道が不安定性を持つことから、局所解から脱することが可能となる。また、提案手法では従来手法のようにプログラムを表現する多数の個体を用いるのではなく、1 個体のみでプログラムを表現し、作成の効率化を図る。そして、提案手法を用いたシミュレーション結果から提案手法と従来手法の結果を比較し、その有効性を検討する。

In this paper, we consider the efficiency of an automatic programming. For such problem, we propose a method by applying chaos neuron. Chaos neuron has an orbital instability, and it is used for suppressing the convergence to a local minimum of the neural network. Moreover, we express the program by only one individual instead of using a large number of individuals that is required in the preceding papers. Through the simulation results, we compare the performance between the proposed method and the previous work. Then we consider effectiveness of the method.

*: 獨協大学経済学部

1. Introduction

To construct an objective computer program automatically, many methods have been proposed. Genetic programming, grammatical evolution, genetic network programming and graph structured program evolution (GRAPE)⁽¹⁾ are known as the method for automatic programming. In these methods, GRAPE can represent computer programs with high flexibility. Therefore, in this paper we focus on GRAPE and propose a method that improves the efficiency.

In general, GRAPE requires a large number of individuals that express the computer program. Hence, to get the target program effectively, the diversity of the individual is required. In the previous paper⁽²⁾, we proposed a new method to create a desired program by maintaining high diversity of individuals.

To obtain the intended program more effectively, we propose a method which utilizes chaos neuron⁽³⁾ that has a characteristic of an orbital instability. In this method, there is an advantage that we can find an objective program by using only one individual without using a lot of individuals as the previous methods⁽¹⁾⁽²⁾.

2. Graph structured program evolution

Automatic programming method which we focus in this paper is called “GRAPE” (graph structured program evolution)⁽¹⁾. This method contains a plural number of “node” that represent one instruction (see Figure 1) and “data set” as a variable. Each node contains several elements that describe the number of the instruction, the number of the next node which will be transferred in the next, and the number of the data sets to be used. Instruction sets contain some operations and some branches. The elements which will be used depends on the kind of the instruction.

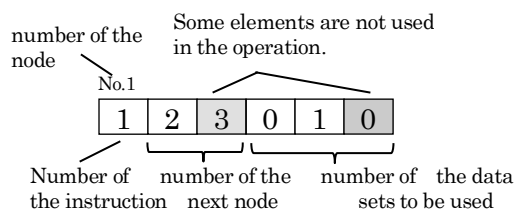


Figure 1 Example of a node

A computer program is constructed by connecting the nodes each other. Applying

evolutional methods to each node that can be seen in genetic algorithm (GA) such as “selection”, “crossover” and “mutation” repetitively, we can obtain the intended program automatically. Then, we obtain a graph structured program containing the loop. Example of the program structure is shown in Figure 2.

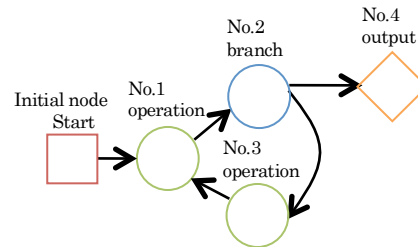


Figure 2 Example of the program structure of GRAPE

3. Applying chaos neuron to GRAPE

3.1 Previous work

To create a desired program efficiently by GRAPE, it is known that the diversity of individuals is required. Usually, minimal gap generation (MGG)⁽⁴⁾, which is called “generation alternation model”, is applied for the purpose of realizing the diversity. MGG uses two selection methods known as “elite selection” and “roulette selection” which are used generally in GA. However, it is clarified that the roulette selection delays convergence to the optimal solution and the elite selection tend to make its solution into local. In the previous report⁽²⁾, we mentioned that MGG has an importance in increasing the fitness value instead of maintaining the diversity of individuals.

To improve the performance of GRAPE, we proposed a new generation alternation model in the previous work⁽²⁾. The method does not contain the crossover, so child individuals are created only by the mutation. The parent individuals are replaced only with children of direct line. Summary of the method is illustrated in Figure 2. In the figure, P_i ($i = 1, \dots, np$) is the parent individual, $C_{i,j}$ ($i = 1, \dots, np, j = 1, \dots, nc$) is the child individuals which are created from the i th parent, and $C_{i,*}$ ($i = 1, \dots, np$) is the next child which has a best evaluation value in each $C_{i,j}$ ($j = 1, \dots, nc$). After obtaining $C_{i,*}$, it is substituted for P_i .

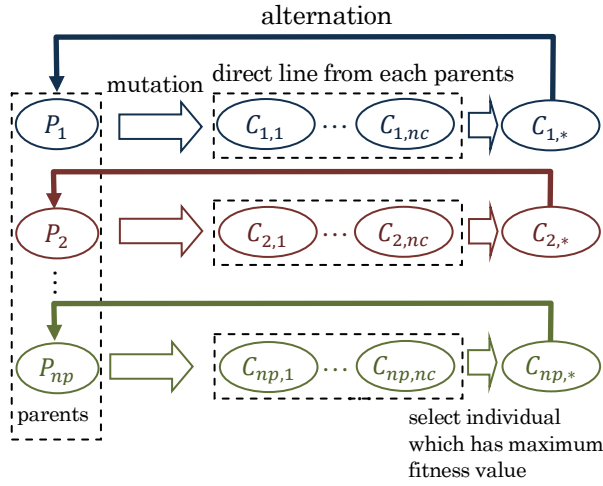


Figure 3 Generation alternation model in the previous work

3.2 Chaos neuron and its application to GRAPE

Neural network is known as a method to approximate input/output pairs by the nonlinear function, which is constructed by connecting a numerous number of neurons. However, as a result of learning, sometimes the correct function cannot be obtained so that the network becomes local solution. On the contrary, the output of chaos neuron has an orbital instability and it is possible to escape from the local solution. By constructing chaos neural network⁽⁴⁾ with chaos neurons instead of the conventional neurons, if the solution becomes local, the solution escapes from the local point and we can obtain the accurate function.

In this paper, we propose a new evolutionary approach by applying chaos neuron. In the previous methods⁽¹⁾⁽²⁾, as well as genetic algorithm, a large number of individuals are prepared to create the program. However, in this work we use only one individual, and apply chaos neuron to find the exact program effectively by evolving the individual.

The dynamics of chaos neuron which we used is described in the following equations. These equations are utilized to solve traveling salesman problem by using chaos neuron⁽⁴⁾. Each element in all nodes has the chaos neuron.

$$\xi_{i,j}(t+1) = \beta(\text{fit}_{\max i,j(t)} - \text{fit}_i(t)) \quad (1)$$

$$\zeta_{i,j}(t+1) = k_r \zeta_{i,j}(t) - \alpha x_{i,j}(t) + (1 - k_r)\theta \quad (2)$$

$$x_{i,j}(t+1) = s(\xi_{i,j}(t) + \zeta_{i,j}(t)) \quad (3)$$

In Eqs. (1)-(3), i ($i = 1, \dots, N_n$) and j ($j = 1, \dots, N_e$) represents the number of nodes and the number of elements, respectively. The variable $\xi_{i,j}$ reveals the exhaustion of chaos neuron according to the improvement of the value of fitness function. fit_i is the value of the fitness function for the i th node. The numbers of the j th node which we can use is defined according to the contents of the element such as “instruction”, “next node” and “data set” described in Figure 1. $\text{fit}_{\max i,j}$ is the maximum value of the fitness function which is obtained by changing the j th node to other numbers. Acquired number in this step becomes the candidate of the new value for the element. $\zeta_{i,j}$ is called refractoriness, and it restrain the exhaustion if the neuron had exhausted in the past time. The output of the chaos neuron is calculated by Eq.(3). In Eq.(3), $s(x)$ denotes the sigmoid function which is described in the following equation:

$$s(x) = \frac{1}{1 + \exp\left(-\frac{x}{\epsilon}\right)} \quad (4)$$

where ϵ determines the gradient around $x = 0$. Figure 4 shows the sigmoid function obtained by changing ϵ .

In the following simulations, we set $\beta = 1.0, k_r = 0.7, \alpha = 1.0$ and $\theta = 1.0$ for Eqs.(1)-(3).

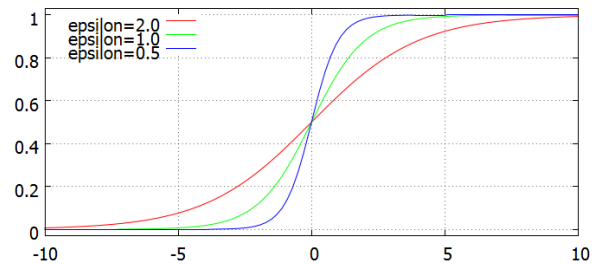


Figure 4 Sigmoid function for various ϵ

In spite of using the chaos neuron, sometimes the constructed program becomes into the local solution. Once the local solution is obtained, the program is hardly to evolve. Therefore, if the node is not updated continuously 150,000 steps¹, we consider that the individual became local solution and reset the program. This situation is thought to be avoided if two or more elements are updated simultaneously, we will propose this method in

¹ This value is selected through the preliminary experiment. However, we have not examined whether it produces the best performance.

future work.

Similar to chaos neural network, it is assumed that we can acquire the objective program effectively by applying chaos neuron to GRAPE. Process of the proposed method is summarized as follows.

1. Prepare chaos neurons for all elements of the nodes. Prepare a variable which counts the times of inactivation.
2. Select one node, and then select one element in the node.
3. For the selected element, calculate Eq.(1) by changing the value of the element² and find the new element that maximizes ξ_i .
4. Calculate Eq.(2) and Eq.(3). If the variable x in Eq.(3) becomes larger than 0.5, we consider that this node is excited, and change the selected element to a new value obtained in step 3.
5. If the new value is same as the old value, we increase the variable which counts the times of inactivation. If this variable exceeds 150,000, discard the obtained program and return to step 1.

4. Computer simulations and results

We show some results which are obtained through the computer simulations by using the proposed method. To compare the performance with the previous research, we used same problems to the proposed method. Problems which we constructed automatically are factorial, exponentiation, fibonacci sequence, and greatest common divisor (GCD). In order to create the program of factorial, exponentiation and fibonacci sequence automatically, same input/output pairs are used as previous works⁽¹⁾⁽²⁾. To create the program of GCD that was not executed in the preceding paper⁽¹⁾, we prepared the input/output pairs (a, b) as $a = 1, \dots, 60, b = 60^{(2)}$. When creating GCD, the initial values of the data sets are input value a for data[0] to data[4], value b for data[5] to data[9], and constant 1.0 for data[10] to data[14]. We used floating point data type for GRAPE in all problems.

The fitness function used in the simulations is as follows.

$$1.0 - \frac{\sum_{i=1}^n \frac{|c_i - e_i|}{|c_i| + |c_i - e_i|}}{n} \quad (5)$$

In eq.(5), n is the number of input/output pairs, c_i, e_i are the correct answer of the problem and the output of the created program for the i th input, respectively. If $c_i - e_i$ becomes 0 for all $i = 1, \dots, n$, the value of the function becomes 1, which is the maximum.

Table 1 is the node instructions which we used in the following simulations. In Table 1, “Idx” is the number of the instructions, “Transition” is the number of the nodes with which each node can branch after executing the instruction, “Parameters” is the data sets used in each instruction, and “Function” is the action which is executed in the instruction.

The size of the nodes is set as 500 for both the previous method and the proposed method. For each problem, simulations were continued until the correct program is obtained for 10 times. In the previous method, if the objective program cannot be obtained until 50,000 steps, we restarted the simulation by initializing all nodes.

Table 1 Node operations

Idx	Transition	Parameters	Function
1	1	x, y, z	$z := x + y$
2	1	x, y, z	$z := x - y$
3	1	x, y, z	$z := x * y$
4	1	x, y, z	$z := x / y$
5	1	x, y, z	$z := x \bmod y$
6	1	x, y	swap x and y
7	2	x, y	if $x > y$ goto node 1, else goto node 2
8	2	x, y	if $x < y$ goto node 1, else goto node 2
9	2	x, y	if $x = y$ goto node 1, else goto node 2
10	2	x, y, z	if $x \leq y$ goto node 1, else goto node 2 & $x := x - z$
11	2	x, y, z	if $x \geq y$ goto node 1, else goto node 2 & $x := x + z$
12	0	x	output x

4.1 Simulation results

Table 2 shows the alternation times which

² If the value of the element changes, the value of the fitness function which contains its element will also change.

we obtained in the simulation results. On exponentiation and factorial, the average value by the previous method is superior to the proposed method. However, in all the results, the proposed method has a smaller minimum value than the previous method. Especially in the result of fibonacci sequence and GCD, minimum value and maximum value becomes small sharply in the proposed method. From the results, we expect that the proposed method can create the desired program by the smaller repetition times. This is an advantage of the method, and we insist that the proposed method can create the intended program effectively.

Table 2 Repetition times required to obtain the correct program

Problem	Proposed method	Previous method
Factorial	Ave. 1553.3 Min. 109 Max. 4111	Ave. 1413.3 Min. 127 Max. 3575
Exponentiation	Ave. 4523.7 Min. 92 Max. 8975	Ave. 3545.3 Min. 416 Max. 17237
Fibonacci sequence	Ave. 4454.4 Min. 704 Max. 9316	Ave. 24463.8 Min. 5707 Max. 46094
GCD	Ave. 2231.3 Min. 139 Max. 8182	Ave. 8918.0 Min. 568 Max. 14792

5. Conclusion

To create the objective computer program efficiently by using the automatic programming, we proposed a method that uses the chaos neuron. Chaos neuron has an orbital instability, so that it brings the efficiency for the construction of the program. To consider the efficiency of the proposed method, we compared the result of some simulations with the previous method. Then we clarified that the proposed method can obtain the desired program effectively.

References

- (1) S. Shirakawa, S. Ogino and T. Nagao: "Graph structured program evolution",
- (2) K. Imafuku : "Proposition of a new method considering high diversity for automatic program generation", IPSJ SIG Technical Report, vol.2012-CE-115, no. 5 (2012)
- (3) K. Aihara: "Chaotic neural networks", Bifurcation Phenomena in Nonlinear System and Theory of Dynamic Systems, pp.143-161,

world Scientific, Singapore (1986)

- (4) H. Sato, I. Ono and S. Kobayashi : "A new generation alternation model of genetic algorithms and its assessment", Japanese Society for Artificial Intelligence, vol. 12, no 5, pp.734-744 (1997)

- (5) M. Hasegawa, T. Ikeguchi, and K. Aihara : "Solving large scale traveling salesman problems by chaotic neurodynamics", Neural Networks, Vol. 15, No. 2, pp.271-283 (2002)

(2013 年 9 月 30 日受付)

(2013 年 12 月 18 日採録)