Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics
San Antonio, TX, USA - October 2009

Evolving Plural Programs by Genetic Network Programming with Multi-Start Nodes

Shingo Mabu
Graduate School of Information, Production and Systems
Waseda University
Hibikino 2-7, Wakamatsu-ku, Kitakyushu,
Fukuoka 808-0135, Japan
mabu@aoni.waseda.jp

Kotaro Hirasawa
Graduate School of Information, Production and Systems
Waseda University
Hibikino 2-7, Wakamatsu-ku, Kitakyushu,
Fukuoka 808-0135, Japan
hirasawa@waseda.jp

***Abstract*—Automatic program generation is one of the applicable fields of evolutionary computation, and Genetic Programming (GP) is the typical method for this field. On the other hand, Genetic Network Programming (GNP) has been proposed as an extended algorithm of GP in terms of gene structures. GNP is a graph-based evolutionary algorithm and applied to automatic program generation in this paper. GNP has directed graph structures which have some features inherently such as re-usability of nodes and the fixed number of nodes. These features contribute to creating complicated programs with compact program structures. In this paper, the extended algorithm of GNP is proposed, which can create plural programs simultaneously in one individual by using multi-start nodes. In addition, GNP can evolve the programs in one individual considering the fitness and also its standard deviation in order to evolve the plural programs efficiently. In the simulations, Even-n-Parity problem and Mirror Symmetry problem are used for the performance evaluation, and the results show that the proposed method outperforms the original GNP.**

***Index Terms*—Genetic Programming, program generation, parity problem, mirror symmetry problem**

I. Introduction

Genetic Algorithm (GA) [1] and Genetic Programming (GP) [2] are the typical evolutionary computation and have been widely studied to solve complex problems. One of the research fields of evolutionary computation is automatic program generation such as generating boolean functions. GP and other learning techniques such as neural networks have been successfully applied to this field [2]–[5].

Genetic Network Programming (GNP) [6] has been proposed as one of the evolutionary computations and applied to many problems [7]–[9]. GNP has directed graph structures which have some inherent features such as re-usability of nodes and the fixed number of nodes. For example, the re-usability of nodes works as Automatically Defined Functions (ADFs) in GP.

GNP has been applied to mainly dynamic problems such as multi-agent systems, stock trading models and elevator group supervisory control systems because the graph structure of GNP has an implicit memory function [6] and can effectively use the past action sequence for decision making. Some comparisons between GNP and other graph-based GP are also described in [6]. On the other hand, GNP was applied to static problems [10] by using explicit memory mechanism, where the expressions generated by processing nodes are stored in the memory. In this paper, to enhance the expression ability of the solutions of the original GNP and generate programs efficiently by making use of the inherently equipped features of the graph structures, multi-start nodes are introduced and plural programs are evolved simultaneously in one individual. Basically, GP creates one program (tree structure) by one individual, but GNP can creates plural programs efficiently even if the number of nodes is small. In addition, GNP can evolve the programs considering the fitness and its standard deviation in the selection phase. Therefore, the aim of this paper is to make better use of each individual to find solutions with small number of generations, and create good programs with smaller number of individuals than standard GNP.

In the simulations, Even-n-Parity problem and Mirror Symmetry problem are used for the performance evaluation, and the results show that the proposed method can find the optimal solutions efficiently.

This paper is organized as follows. In the next section, the proposed method is explained in detail. Section 3 describes the simulations and their results, and some conclusions are given in section 4.

II. Genetic Network Programming with Multi-Start Nodes

In this section, a basic structure of GNP with multi-start nodes and its memory structure are introduced, then how to construct programs and evolve them is explained in detail considering the fitness and its standard deviation.

A. Directed graph structure with memory and multi-start nodes

GNP with multi-start nodes has directed graph structures shown in Fig. 1. The graph structure consists of three kinds of nodes, i.e., start nodes, judgment nodes and processing nodes, and a memory. The start nodes indicate the first nodes to be executed after starting the program. In Fig. 1, there are three start nodes, so three node transitions are independently carried out, and as a result, three sets of programs/expressions are created by three node transitions. This is the most important point in this paper, because many expressions can be evolved by each individual with a quite compact program structure.

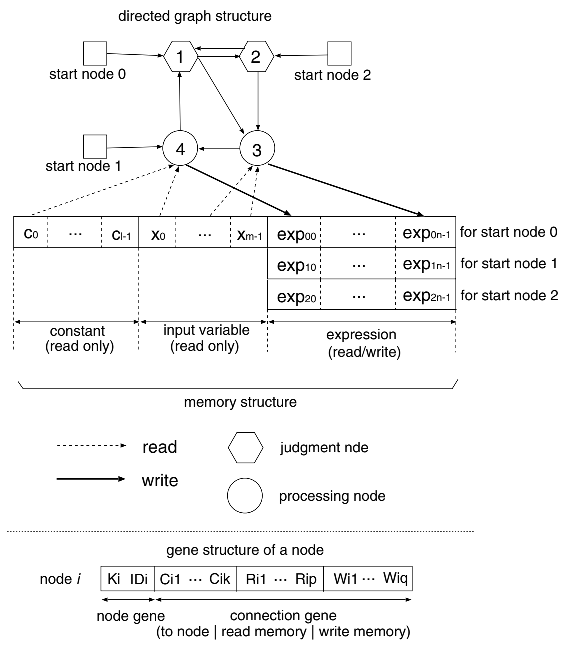
directed graph structure
start node 0
start node 2
start node 1
1
2
3
4
c0 … cl-1 x0 … xm-1 exp00 … exp0n-1 for start node 0
exp10 … exp1n-1 for start node 1
exp20 … exp2n-1 for start node 2
constant (read only)
input variable (read only)
expression (read/write)
memory structure
read
write
judgment nde
processing node
gene structure of a node
node i
Ki IDi Ci1 … Cik Ri1 … Rip Wi1 … Wiq
node gene
connection gene (to node | read memory | write memory)

Fig. 1. A structure of GNP with multi-start nodes

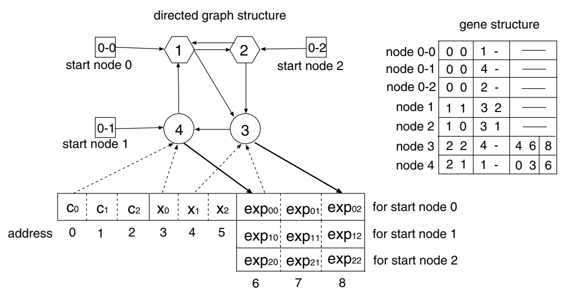

Fig. 2. An example of the structure of GNP with multi-start nodes

TABLE I
NODE FUNCTIONS

node type	ID	node function
judgment node	0	select one from two branches
($K_i = 1$)	1	select one from three branches
	2	select one from four branches
processing node	0	AND
($K_i = 2$)	1	OR
	2	NAND
	3	NOR

After the start node, the node transition is carried out according to the connections between nodes and branch selections. The role of a judgment node is to check the current situation and select a branch (connection) to the next node. In this paper, a judgment node selects the first branch when the node is visited for the first time, selects the second branch when the second time, and so on. After the last branch is selected, the first branch is selected next time. A processing node reads constants, variables or expressions from the memory, then implements an operation assigned to the node and puts the answer into the expression memory again. In this paper, a processing node has two connections to the memory to read data and write the answer, and one connection to the next node.

Next, the memory structure is explained. The constant part has constant values such as 0 and 1, but this part is not used in this paper. The input variable part has input values. In the case of Even-3-Parity problem, 0 or 1 is stored in x_0, x_1 and x_2. The expression part saves the generated programs/expressions. In this paper, the node transition started from start node 0 saves the generated expressions in $(exp_{00}, \ldots, exp_{0n-1})$, then all the expressions are regarded as the candidate solutions of the problem, and the fittest one is selected as a solution of the corresponding start node. When the node transition is started from start node 1, $exp_{10}, \ldots, exp_{1n-1}$ are used to save the generated expressions. Therefore, if the number of start nodes is set at three, three solutions are generated by one individual.

B. Relation between the graph structure and the gene structure

A graph structure and the connections to the memory is represented by the gene structure. The general expression of the gene structure is shown in Fig. 1 and an example of the graph structure with its gene structure is shown in Fig. 2.

In Fig. 1, K_i shows the node type (0: start node, 1: judgment node, 2: processing node). ID_i shows the ID number of the node function, and the contents of the functions used in this paper are shown in TableI. For example, $K_i = 2$ and $ID_i = 0$ shows the processing node having AND operator. $C_{i1}, \ldots, C_{ik}$ show the next node number connected from node i. In the case of judgment nodes, $k \geq 2$ (plural connections), and in the case of start nodes and processing nodes, $k = 1$ (single connection). $R_{i1}, \ldots, R_{ip}$ show the address of the memory which node i reads. If $p = 2$, a processing node reads two values/expressions in the memory. $W_{i1}, \ldots, W_{iq}$ show the address of the memory in which node i puts the generated expression. If $q = 1$, a processing node puts one generated expression into the memory whose address is W_{i1}.

C. Program generation by the node transition

The program generation by GNP with multi-start nodes is explained in this subsection. The proposed method has plural start nodes, so the node transition is carried out three times independently from each start node. Here, suppose the node transition is started from start node 0. In Fig. 2, the next node is node 1 (judgment node) which is visited for the first time, so the first branch is selected and the next node number becomes $C_{11}(= 3)$. Node 3 is a processing node, its function is $NAND$, the addresses of the data node 3 reads are 4 and 6, and the address of the memory into which the node puts the generated expression is 8. Therefore, node 3 reads variables x_1 and exp_{00}, and puts the generated expression (x_1 $NAND$ exp_{00}) into exp_{02}. In addition, the next node number becomes $C_{31}(= 4)$ because the current node number is 3 and processing nodes have only one branch. GNP repeats the above process

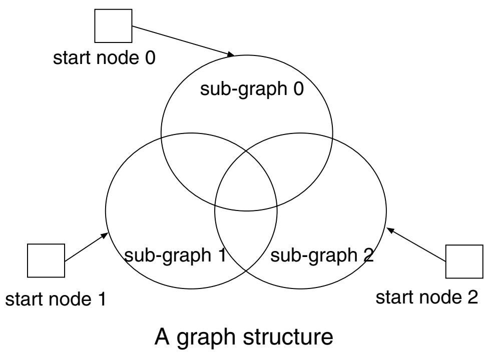

Fig. 3. The concept of a function localized structure

until the predefined number of nodes are executed. Finally, GNP can create $n(=3)$ expressions $(exp_{00}, exp_{01}, exp_{02})$ by one start node and the fittest one is selected. In this example, three expressions are created by three start nodes, and the fittest one among them becomes the solution of the individual. Note that although only two judgment nodes and two processing nodes are used in this example and the structure is quite simple, various kinds of expressions can be created if the number of nodes becomes larger in the actual use. In addition, Fig. 3 shows the concept of a function localized structure, where a certain part of the graph structure is used by the node transition from start node 0, another part is used by that from start node 1, and the other part is used by that from start node 2. Moreover, some parts are shared by the transitions from some start nodes. Therefore, the function localized networks could be created by making good use of the features of the graph structure.

D. Fitness considering standard deviation

First, how the best expression is selected from each individual is summarized.

1) In each individual, n expressions are created by the node transition started from each start node.
2) The fittest expression[1] among these n expressions is selected.
3) If the number of start nodes is s, we can obtain s expressions with the fitness of $f_1, f_2, \ldots, f_s$, respectively.
4) Finally, the fittest expression among s expressions is selected as a solution of the individual.

Next, consider the following extended fitness functions which are used to select individuals for the crossover and the mutation.

$$\text{fitness}_{ind} = \min\{f_1, f_2, \ldots, f_s\} \quad (1)$$
$$\text{fitness}_{ind} = \min\{f_1, f_2, \ldots, f_s\} + \alpha\sigma_{ind} \quad (2)$$
$$\text{fitness}_{ind} = \min\{f_1, f_2, \ldots, f_s\} - \alpha\sigma_{ind} \quad (3)$$

Eq. (1) is the basic fitness function which shows that the fitness of the best expression in each individual becomes the fitness of the individual. However, in the evolution phase described in the next subsection, not only Eq. (1), but also Eq. (2) is used to select parents. Eq. (2) considers the standard deviation σ of the fitness of all the expressions $f_1, f_2, \ldots, f_s$ in an individual. α is a coefficient which determines the weight of σ. If σ is smaller, the fitness is better, which means that an intensified search is more important. On the other hand, Eq. (3) focuses on a diversified search because the large fitness variation is regarded as important. In this paper, the combination of Eq. (1) and (2) is used because the simulation results show that the intensified search, i.e., small σ, is more important than the diversified search, i.e., large σ. The comparison of the performance between the case of using Eq. (2) and (3) is shown in the simulation section.

[1]The fitness in this paper is the error; thus the smaller, the better.

E. Evolution

This subsection describes the procedures of crossover and mutation.

1) Crossover: The crossover is carried out as follows.

i) select two individuals as parents using tournament selection[2] twice.
ii) each node number is selected as a crossover node number with the crossover rate P_c.
iii) exchange all the genes of the nodes with the crossover node number between two parents.

2) Mutation: The mutation is carried out as follows.

i) select one individual as a parent using tournament selection once.
ii) select each connection ($C_{i1}, \ldots, C_{ik}, R_{i1}, \ldots, R_{ip}, W_{i1}, \ldots, W_{iq}$) of the parent with the mutation rate P_m.
iii) the selected connections are randomly changed/reconnected to another node or data in the memory.

In the crossover and the mutation procedures, note that 80% of the individuals are selected based on Eq. (1), and the remaining 20% are based on Eq. (2), in order to consider the balance between the individuals including at least one excellent expression and those including many relatively good expressions.

III. Simulation

In this section, the performance of the proposed method is evaluated comparing with the standard GNP with single start node (SGNP), where two benchmark problems are used; one is Even-n-Parity problem and the other is Mirror Symmetry problem. Even-n-Parity function is a boolean function which returns 1 when the number of 1 in the n input values is odd, otherwise 0. Mirror Symmetry function returns 1 when the input pattern is symmetric, e.g., 101, 1001,..., otherwise 0.

A. Simulation conditions

Table II shows the parameters used in the simulations. In this paper, the number of individuals produced by crossover and mutation is fixed. In Even-3-Parity problem, for example,

[2]The reason why tournament selection is used is that the selection pressure is easily determined by the tournament size and the calculation cost is relatively small.

TABLE II
SIMULATION CONDITIONS

population size (Even-3-Parity, Mirror-Symmetry)	301 (crossover: 120, mutation: 180, elite: 1)
population size (Even-4-Parity)	601 (crossover: 240, mutation: 360, elite: 1)
crossover rate P_c	0.1
mutation rate P_m	0.02
α	0.1
tournament size	5
the number of expressions n	12
maximum expression length	200 bytes (even-3-parity) 500 bytes (even-4-parity, mirror symmetry)
the number of nodes	105 (15 nodes per each kind of function/ID)
the number of steps	50

the number of individuals (population size) is 301 including 120 individuals produced by crossover, 180 individuals produced by mutation and one elite individual. The crossover and the mutation are separately carried out in order to avoid too much changes of the genes and to determine the crossover rate and the mutation rate independently through the simulations.

The maximum expression length means that each expression consists of no more than 200 bytes or 500 bytes data, where each input variable and each operator use one byte. The total number of nodes is 105 because the number of functions is seven (three judgment functions and four processing functions) and the number of nodes of each function is 15. The number of steps means that the node transition is repeated 50 times after the start node, i.e., GNP uses 50 nodes to create expressions.

Finally, how to initialize the population is explained. In each individual, 15 nodes per each kind of function/ID are prepared, thus the total number of nodes is 105 ($= 7 \times 15$). The connections between nodes are randomly determined, those for reading memory are connected to randomly selected input variable or expression, and those for writing expressions are connected to randomly selected expression memory.

B. Simulation results

1) Even-3-Parity problem: Fig. 4 shows the fitness curves[3] of GNP with multi-start nodes (four start nodes) and SGNP averaged over 100 independent simulations, where the fitness of the proposed method decreases faster than SGNP. Table III shows the results of the t-test on the fitness obtained at the last generation. From the table, it can be seen that there is a significant difference between the proposed method and SGNP. Fig. 5 shows the success ratio, i.e., the ratio of obtaining the optimal solutions, at each generation, and the proposed method obtains the optimal solutions in 99% of the simulations. SGNP obtains the optimal solutions in 92% of the simulations, therefore the proposed method searches for solutions faster and accurately.

[3]Note that y-axis (fitness) in the figure shows the error of the best program at each generation. That is, the standard deviation is not considered. The standard deviation is considered in the selection phase only.

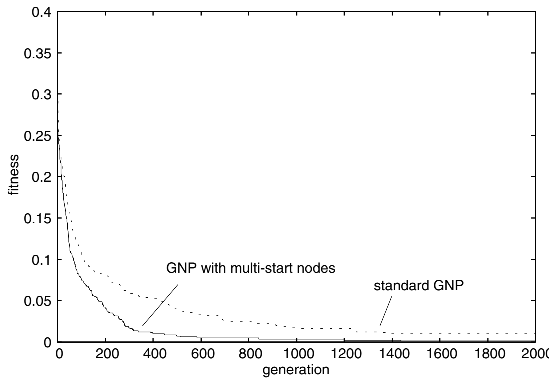

Fig. 4. Fitness curves in Even-3-Parity problem

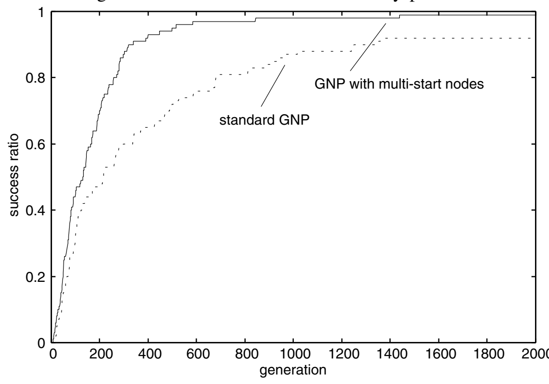

Fig. 5. The ratio of obtaining the optimal solutions in Even-3-Parity problem

TABLE III
T-TEST ON THE FITNESS OBTAINED AT THE LAST GENERATION

	proposed method		SGNP		
	mean	STD	mean	STD	p-value[%]
Even-3-Parity	0.00125	0.0125	0.0115	0.0363	0.723
Even-4-Parity	0.0319	0.0529	0.0594	0.0631	0.0999
Symmetry	0.0594	0.0578	0.104	0.0642	0.0001

STD: standard deviation

2) Even-4-Parity problem: Fig. 6 shows the fitness curves of GNP with multi-start nodes (five start nodes) and SGNP. The tendency is the same as the case of Even-3-Parity problem, where the fitness decreases faster than standard GNP, and Table III shows that there is a significant difference between the proposed method and SGNP. Fig. 7 shows the success ratio, where the proposed method can obtain the optimal solutions in 68% of the independent simulations, while the standard GNP can obtain them in 45% of the simulations.

Next, the effects of the fitness function considering standard deviation are examined. We compare the proposed method using the fitness functions of Eq. (1) and (2) with that using Eq. (1) only and that using Eq. (1) and (3). Fig. 8 shows the fitness curves and Fig. 9 shows the success ratio under the above conditions. From the figures, the combination of Eq. (1) and (2) shows the best results among the three. The GNP based on Eq. (1) shows the better result than that based on Eq.

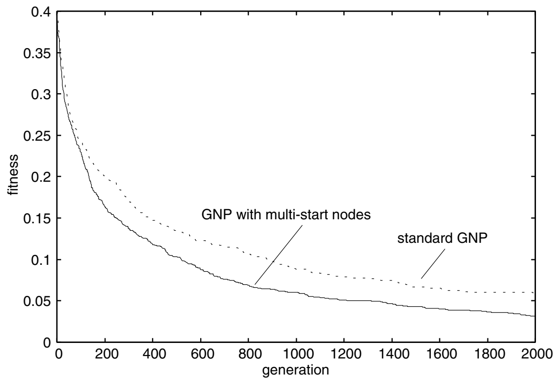

Fig. 6. Fitness curves in Even-4-Parity problem

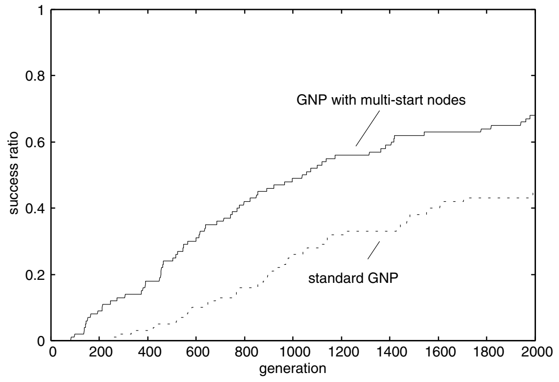

Fig. 7. The ratio of obtaining the optimal solutions in Even-4-Parity problem

(1) and (3), therefore, it can be said that the intensified search is more important than the diversified search in this problem, and the combination of multi-start nodes and the new fitness function contributes to the better performance than standard GNP.

Fig. 10 and 11 shows the tracks of the node transitions of the best individual at the last generation, which are started from start node 1 and 2, respectively. The x-axis shows the node functions, which includes the processing functions $\{AND, OR, NAND, NOR\}$, and the judgment functions $\{J0, J1, J2\}$. $J0$, for example, shows a judgment node with $ID = 0$ (Table I). The y-axis distinguishes the same kind of nodes, i.e., 15 nodes per each kind of node function are in the graph structure and they have the number $0, 1, 2, \ldots, 14$, respectively. For example, $(x, y) = (NOR, 1)$ shows the second NOR node.

From the figures, it is shown that some nodes used in the node transitions are different depending on the start nodes although a number of nodes are shared by both node transitions. As the generation goes on, the useful nodes for constructing boolean functions are shared by several node transitions. However, each transition still keeps variation to search for better solutions by using different kinds of nodes.

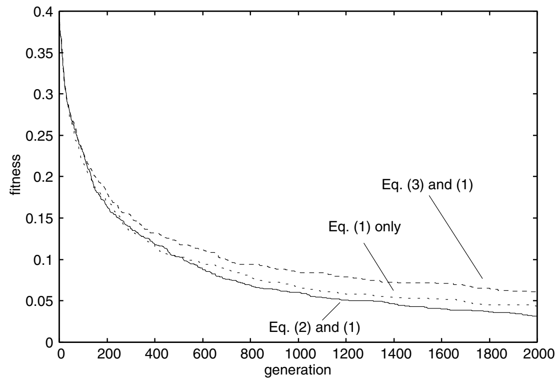

Fig. 8. Fitness curves of the different fitness functions in Even-4-Parity problem

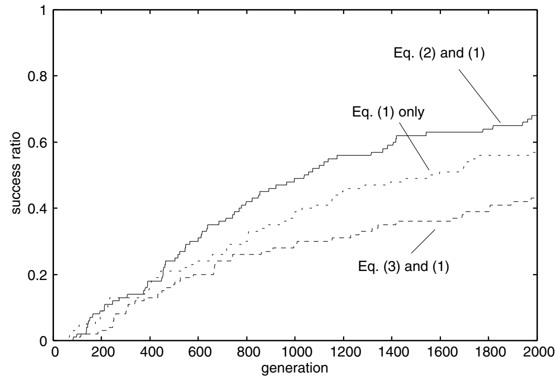

Fig. 9. The ratio of obtaining the optimal solutions of the different fitness functions in Even-4-Parity problem

In addition, each node transition uses specific nodes repeatedly like sub-routine.

3) Mirror Symmetry problem: In this subsection, another benchmark problem, Mirror Symmetry problem, is used to confirm the performance of the proposed method. In this problem, the number of input variables is set at four. Fig. 12 shows the fitness curves of GNP with multi-start nodes (five start nodes) and SGNP, and Fig. 13 shows the success ratio. From the results, it can be seen that the proposed method also shows better results than SGNP, where the proposed method can obtain the optimal solutions in 45% of the independent simulations, while the standard GNP can obtain them in 21% of the simulations. In addition, Table III shows that there is a significant difference between the proposed method and SGNP in terms of the fitness.

IV. Conclusions

In this paper, GNP with multi-start nodes is proposed and its search ability for solutions are evaluated. The multi-start nodes contribute to creating many expressions even if the number of nodes is small (in this paper, 105 nodes), because the graph

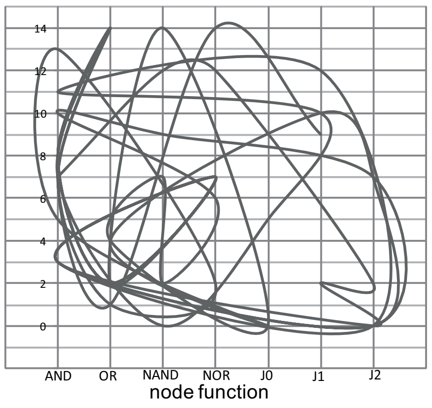

Fig. 10. Track of the node transition from start node 1 at the last generation

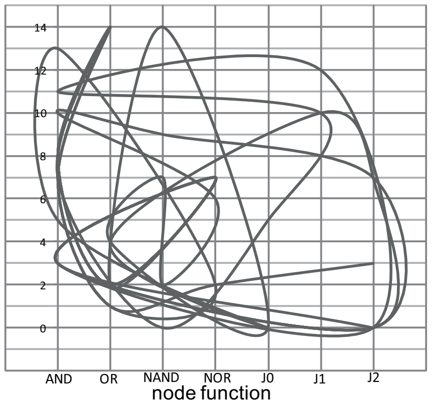

Fig. 11. Track of the node transition from start node 2 at the last generation

structure inherently has an ability to reuse nodes, and each start node implements its own node transition and creates the programs. In addition, the standard deviation of the fitness values is considered to evolve programs efficiently.

From the simulation results using Even-n-Parity problem and Mirror Symmetry problem, it is clarified that the proposed method shows higher fitness and success ratio comparing to standard GNP with single start node.

In the future, the proposed method is enhanced and applied to complicated function approximation, time series prediction, and finally a stock market analysis as an application.

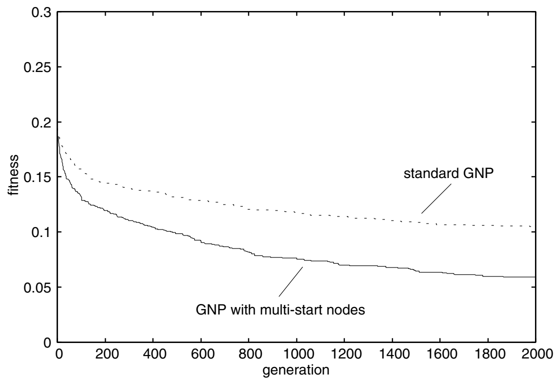

Fig. 12. Fitness curves in Mirror Symmetry problem

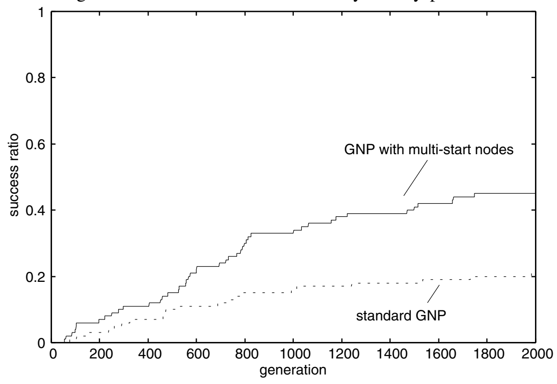

Fig. 13. The ratio of obtaining the optimal solutions in Mirror Symmetry problem

References

[1] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, 1975.

[2] J. R. Koza. *Genetic Programming, on the programming of computers by means of natural selection*. MIT Press, Cambridge, Mass., 1992.

[3] J. R. Koza. Genetic Programming II, Automatic Discovery of Reusable Programs. MIT Press, Cambridge, Mass., 1994.

[4] R. Setiono. On the solution of the parity problem by a single hidden layer feedforward neural network. Neurocomputing, Vol. 16, pp. 225–235, 1997.

[5] D. G. Stock and J. O. Allen. How to solve the N-bit-parity problem with two hidden units. Neural Networks, Vol. 5, pp. 923–926, 1992.

[6] S. Mabu, K. Hirasawa and J. Hu. A Graph-Based Evolutionary Algorithm: Genetic Network Programming and Its Extension Using Reinforcement Learning. *Evolutionary Computation*, MIT Press, Vol. 15, No. 3, pp. 369–398, 2007.

[7] K. Hirasawa, T. Eguchi, J. Zhou, L. Yu and S. Markon, A Double-Deck Elevator Group Supervisory Control System Using Genetic Network Programming, IEEE Trans. on Systems, Man and Cybernetics, Part C, Vol. 38, No. 4, pp. 535–550, 2008.

[8] S. Mabu, Y. Chen, K. Hirasawa and J. Hu. Stock Trading Rules Using Genetic Network Programming with Actor-Critic. 2007 IEEE Congress on Evolutionary Computation (CEC2007), pp. 508–515, 2007.

[9] S. Mabu, H. Hatakeyama, Moe Thu Thu, K. Hirasawa and J. Hu, Genetic Network Programming with Reinforcement Learning and Its Application to Making Mobile Robot Behavior, IEEJ Trans. EIS, Vol. 126, No. 8, pp. 1009–1015, 2006.

[10] S. Mabu, K. Hirasawa, Y. Matsuya and J. Hu. Genetic Network Programming for Automatic Program Generation. Journal of Advanced Computational Intelligence and Intelligent Informatics, Vol. 9, No. 4, pp. 430–436, 2005.