Available online at www.sciencedirect.com
ScienceDirect
Procedia Computer Science 96 (2016) 120 – 128
Procedia Computer Science

20th International Conference on Knowledge Based and Intelligent Information and Engineering Systems

Obtaining repetitive actions for genetic programming with multiple trees

Takashi Ito[a,]*, Kenichi Takahashi[a], and Michimasa Inaba[a]

[a]*Graduate School of Information Sciences, Hiroshima City University, 3-4-1 Ozukahigashi, Asaminami-ku, Hiroshima, Japan*

Abstract

This paper proposes a method to improve genetic programming with multiple trees (GP_{CN}). An individual in GP_{CN} comprises multiple trees, and each tree has a number *P* that indicates the number of repetitive actions based on the tree. In previous work, a method for updating the number *P* has been proposed to obtain *P* suitable to the tree in evolution. However, in the method efficiency becomes worse as the range of *P* becomes wider. In order to solve the problem, in this study, two methods are proposed: inheriting the number *P* of a tree from an excellent individual and using mutation for preventing the number *P* from being into a local optimum. Additionally, a method to eliminate trees consisting of a single terminal node is proposed.

Keywords: autonomous agent; garbage collection problem; genetic programming; evolutionary learning; multiple trees.

1. Introduction

In the field of artificial intelligence, which aims at modeling human intelligence, search algorithms for obtaining agent decisions and action rules to reach a goal have been studied by many researchers. Reinforcement learning and evolutionary learning are representative means to learn agent behavior. Evolutionary learning, which imitates the mechanism of biological evolution, is known to be a way in which we can obtain optimum rules for agent action from a broad search space. Among evolutionary methods, the genetic algorithm (GA)[1], the genetic programming (GP)[2,3] and the genetic network programming (GNP)[4] have been investigated eagerly and widely.

In GP, the population in the next generation is produced by generating child individuals from parent individuals with high fitness values in the current generation. Each individual is comprised of a single tree structure in GP. The leaf nodes correspond to agent actions, and the other nodes correspond to branches depending on perceptual information. Individuals in the next generation are generated using genetic operations, namely crossover, mutation, and inversion. Some of individuals with high fitness values are called elites and are inherited to the next generation. In order to improve the performance of GP, various methods have been proposed: the methods generating individuals in the next generation by joining fragments of the tree structure that randomly been sampled from several parent individuals[5], extracting useful tree structures from individuals called frequent trees[6,7] that are subtrees that frequently appear in the population, using the island model that combines those frequent trees[8], using the Semantic Aware Crossover (SAC)[9] that uses the similarity of subtrees to avoid destructive of tree structures, and using the select operation with semantics for keeping diversity [10].

In GP, the depth of generated trees tends to be deep to obtain complex action rules because one individual has only one tree. Deep trees with complex action rules have shortcomings: the first is that the readability of rules obtained by the individual becomes lower, and the second is that there is a possibility that excellent action rules might be destroyed by a single genetic

* Corresponding author. Tel.: +81-80-4555-0426.
E-mail address: ito@cm.info.hiroshima-cu.ac.jp

doi:10.1016/j.procs.2016.08.111

operation. In our study, genetic programming with control nodes (GP_{CN})[11] has been proposed to improve the readability. In addition, its some kinds of modification have been proposed[12,13,14]. An individual in GP_{CN} comprises multiple trees. The tree has the following two numbers: the identification number that indicates the order in which an agent refers to a tree, and the number *P* that indicates the number of repetition by which an agent carries out the action designated by the leaf node in a tree. In $GP_{CN,}$ an individual has multiple trees, and each tree represents different an action rule. Therefore, GP_{CN} can solve the problem that GP produces deep trees, because in GP_{CN}, complex action rules represented by a single tree in GP can be divided into multiple simple trees. However, we have to preset the identification number and the number *P* for GP_{CN}, and these optimal values might be different depending on benchmark problems. Thus, in previous work[12], a method for updating the number *P* to obtain *P* suitable to the tree in evolution has been proposed. In the method for updating the number *P*, one tree is selected at random from the individual, and then the number *P* of a selected tree is changed by adding or subtracting a randomly selected value. With the method, we can let the value of *P* of each tree in an individual gradually converges to a suitable value in evolution.

However, there is a problem that efficiency becomes worse as the range of *P* becomes wider. In order to solve the problem, we propose a method to inherit the number *P*. When crossover is applied to two parent individuals, value of *P* of the parent individual with the higher fitness value in the two parent individuals is set to *P* of both two child individuals. The authors consider that with the method, the number *P* can be converged to an optimal value, because the number *P* of excellent individuals can be inherited to descendants.

However, a value of *P* might quickly spread out among individuals, which might cause to lose the diversity of *P* and to hider *P* from converging a value because the number *P* in a parent individual is inherited to two child individuals. Therefore, we propose a method using mutation for the number *P* to avoid the problem. The method randomizes the value of *P* when mutation for a tree in genetic operations is carried out. With the method, when the population whose values of *P* are being lead into local optima, the method can prevent it by setting a random value to *P*.

Additionally, in our study, we discovered a problem that trees obtained by GP_{CN} tend to be trees with a single terminal node, because individuals that are comprised of these trees obtain higher the fitness value than other individuals at early generations. In order to solve the problem, we propose a new fitness function to eliminate trees with a single terminal node.

We apply the proposed methods to a garbage collection problem[15] to compare the performance with that of the previous methods. We adopt the garbage collection problem because this problem has been used to show the ability of GP in previous work. Although the symbolic regression problem exists as another type of benchmark problem for GP, we chose the garbage collection problem, because the objective of this paper is to obtain rules for agent actions.

2. Genetic Programming with Control Nodes (GP_{CN})

2.1. Outline of GP_{CN}

An example of an individual of GP with control nodes (GP_{CN})[11] that has been extended to have multiple trees is depicted in Fig. 1. Individuals of GP_{CN} comprise multiple trees which correspond to action rules for an agent. The tree has the following two numbers: the identification number that indicates the order in which an agent refers to a tree, and the number *P* that indicates the number of repetition by which an agent carries out the action designated by the leaf node in a tree. The number of trees in one individual, i.e. the number of control nodes is denoted by *M* and is supposed to be determined in advance. Each tree has its own number *P*.

The algorithm of initially setting the number *P* to each tree is the following.

```
BEGIN
    FOR $i$ = 1 to $M$
        $i$th tree–> $P$ = Random.uniform( 1, $Total\ Steps/M$ )
        Generate Tree( $i$th tree )
    ENDFOR
END
```

Here, *i*th tree is the tree with the identification number *i*, *i*th tree–> *P* is the number *P* of *i*th tree, *Total Steps* is the maximum number of actions of an agent set for each problem.

A non-terminal node corresponds to a branch by the perceptual information, and a terminal node corresponds to an action that an agent can execute. An agent refers to a tree with the smallest number and carries out an action according to the tree. When the number of actions that an agent carries out using the tree exceeds a designated number *P*, the agent refers to a tree with the next number. After the tree with the largest number is processed, the agent refers to the tree with the smallest number. At that time, the number of actions that an agent carries out using the tree in each tree is initialized to 0. Until the accumulated number of actions of trees which an agent refers to becomes *Total Steps*, the agent repeats receiving perceptual information from the environment and then carrying out an action. When the total number of actions in trees that the agent carries out so far reaches *Total Steps*, agent simulation for the individual stops, and then its fitness value is evaluated.

The algorithm of GP_{CN} is the same as that of the previous GP. First, GP_{CN} generates the initial population of individuals. Then, it evaluates the fitness of each individual that has been generated. If the condition to terminate processing is not met, then

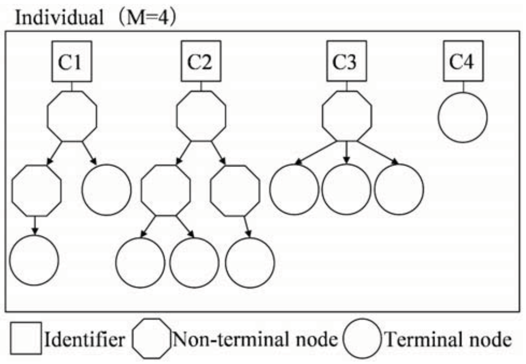

Fig. 1. An example of an individual in GP_{CN}.

it performs reproduction of the population of individuals and genetic operations. It then generates a population of individuals in the next generation. Here, the condition to terminate processing is that the number of generations becomes the designated number of generations. Although the GP_{CN} individuals have multiple trees, the fitness is evaluated for each individual, not for each tree. Details of the genetic operations for GP_{CN} are described in the next subsection.

2.2. Genetic Operations

Because one individual in GP_{CN} has multiple trees unlike normal GP, for each genetic operation an individual is selected by tournament selection. Then one tree is selected at random from the selected individual. Each genetic operation is applied to the selected tree.

- Crossover

Crossover is the operation that exchanges subtrees in trees of two parent individuals. First, two trees are selected from two parent individuals respectively, and nodes are selected at random for crossover from all nodes of each tree. Second, subtrees whose root nodes are the selected nodes are exchanged. However, no crossover is executed when a tree consists of only a root node.

- Mutation

We use two kinds of mutation: a mutation-tree and a mutation-node. The mutation-tree is an operation that randomly selects one node from all nodes in a tree of a parent individual and then replaces the subtree subsequent to the selected node with a randomly generated subtree. The mutation-node is an operation that changes the content of the selected node after selecting a node in a tree of a parent individual. In mutation-node, if the selected node is a non-terminal (terminal) node, then the node content is replaced with another content of a non-terminal (terminal) node. When any content of a non-terminal node is changed, the edge number might change. If the number of edges of a new content becomes smaller, then the extra edges and the succeeding subtrees are removed. However, if the number of edges becomes larger, then randomly generated subtrees are connected to the increased edges.

- Inversion

The inversion operation selects only a non-terminal node at random from all nodes of a tree in a selected individual and selects at random two child nodes of the node. Then it exchanges the subtrees that have the two child nodes as the root nodes. However, if the root node of the selected tree is terminal node, the inversion operation is not applied to the selected tree.

In mutation and inversion that generate one child individual from one parent individual, the generated child individual inherits the value of P from the parent individual, because we think that the tree structure of its child individual resembles the parent individual. In crossover that generates two child individuals from two parent individuals, each child individual inherits the value of P from the parent individual, which resembles the generated child individual in tree structure.

3. The method for updating the number *P*

We think that optimal values of the number M that indicates the number of trees in one individual and the number P that indicates the number of repetition by which an agent carries out the action designated by the leaf node in a tree differ with benchmark problems. Additionally, we think that unnecessary actions of an agent are inhibited by setting a suitable value of P for each tree.

Therefore, we have proposed a method for updating the number P to obtain P suitable to the tree in evolution[12]. GP_{CN} updating the number P is represented as $GP_{CN}(e)$. The method selects a tree in individual at random, and then changes the value of P for the selected tree. Let i denotes the identification number, and $P_C(i)$ denotes the number P of $i$he tree in current generation. Then, $P(i)$ which is the changed number P of $i$he tree is changed as follows.

$$P(i) = P_C(i) + \beta \tag{1}$$

In (1), the value of β is the range to change the value of P, and chosen randomly between $[-\gamma, +\gamma]$. The initial value of γ is 16. The value of γ is decreased gradually to 1 at every [*Maximum generation*/10] generation. The method is applied to a tree in an individual.

4. Proposed Methods

4.1. The method for inheriting the number P

In previous work[12], the method for updating the number P to obtain a value suitable to the tree in evolution has been proposed. The method changes gradually the value of P of a tree in an individual by increasing or decreasing. However, there is problem that efficiency becomes worse as the range of P becomes wider.

In order to solve the problem, we propose a method to inherit the number P. GP_{CN} with inheriting the value of P is represented as GP_{CN_I}. The algorithm of GP_{CN_I} is the following.

1) Select a tree A from a parent individual at random.
2) Select a tree B from another parent individual at random.
3) Generate the child individual X and the child individual Y by crossover of the tree A and the tree B.
4) Compare the fitness values of the two parent individuals, and then execute the following.
   If the fitness value of the parent individual with tree A (tree B) is higher than the parent individual with tree B (tree A), the value of P of the parent individual with tree A (tree B) is inherited to trees X and Y.

With the method, child individuals in the population can inherit the value of P for trees from excellent individuals, and we can expect suitable values of P to spread among individuals.

4.2. The method using mutation of the number P

In the method to inherit the number P, the number P for trees in individuals can converge into a suitable value of P by inheriting the value of P for trees in excellent individuals. However, in the method, the value of P might fall into a local optimum.

Thus, we propose a method using mutation of the number P to prevent P from being lead into a local optimum. Let GP_{CN_M} denote GP_{CN} using mutation of the number P. In addition to the two kinds of mutation, we use mutation for the number P. As having shown in Section 2.1.2, there are two kinds of mutation: the mutation-tree and the mutation-node. The mutation for the number P is executed just after the mutation-tree that largely changes action rules, because we think that the method largely affects the structure of individuals.

First, a value is chosen randomly between [1, *Total Steps*/*M*], which is a range of the number P, and then the value of P for the tree which is changed by mutation-tree is changed to the randomly chosen value.

4.3. The fitness function to eliminate trees with a single terminal node

In GP, an individual is comprised of a single tree, and the individuals with a tree comprised of a single terminal node (i.e. the tree that is comprised of a single agent action) is eliminated in evolution. For example, in a garbage collection problem that an agent picks up all pieces of trash scattered in the field and carries them to a garbage dump site, the agent that can only move forward or turn right can't carry all trash to a garbage dump site. Those individuals are eliminated in evolution, because the fitness values of them become low.

However, in GP_{CN}, an individual is comprised of multiple trees. Individuals with trees comprised of a single terminal node are functionally same as individuals comprised of a single tree with multiple agent actions. Thus, in the garbage collection problem, these individuals with trees consisting only of one terminal node can survive, because the agent who refers to multiple trees can move and carry some trash to a garbage dump site. On the other hand, individuals with complex action rules comprised of non-terminal nodes and terminal nodes tend to be eliminated in evolution, because these individuals include many unnecessary actions of the agent at early generations, which makes their fitness values low.

In this study, we propose a new fitness function to eliminate trees with a single terminal node and use the proposed fitness function. In the proposed fitness function, a penalty value for having trees with a single node is included in the fitness function.

We can expect trees with a single node to die out from population.

5. Experiments

5.1. Garbage collection problem

The objective of a garbage collection problem is that an agent picks up all pieces of trash scattered in the field and carries them to a garbage dump site. An example of the field of the garbage collection problem is depicted in Fig. 2. The field comprises a two-dimensional lattice plane of the size 11×11 cells, and the outermost cells are walls. The garbage collection problem has one agent, ten pieces of trash, and one dump site on the field. The agent can move forward, turn left or right, or stay at each step. The agent can also pick up a piece of trash by reaching the cell where it exists and then can carry it to the dump site. The maximum number of pieces of trash that the agent can carry is assumed as two. We prepare 10 environments generated by placing the agent, trash, and the dump site in advance. Here, the agent and trash are placed at randomly selected cells.

In previous work[11,12], we had defined a fitness function as a total of the number of trash carried to the dump sites in the 10 environments in 250 steps per environment. Let $N_i(j)$ denote the number of trash collected by individual j in environment i and $T(j)$ denote the total number of trash collected by individual j. Then, $T(j)$ is calculated as shown below.

$$T(j) = \sum_{i=1}^{10} N_i(j) \tag{2}$$

In this study, we use the proposed fitness function that is given a penalty value for having trees with a single node. Let $M(j)$ denote the number of trees in individual j, $M_{a\ single\ node}(j)$ denote the number of trees with a single node in individual j, α denote a penalty value for having trees with a single node, and $f_{new}(j)$ denote the fitness value proposed to eliminate trees with a single terminal node. Then, the fitness value of individual i of the proposed fitness function is calculated as shown below. In addition, if calculation result is a minus, we change the fitness to 0.

$$f_{new}(j) = T(j) - \alpha \times \frac{M_{a\,single\,node}(j)}{M(j)} \tag{3}$$

The maximum value of the proposed fitness function is 100. In experiments, we measure the highest fitness value obtained in a simulation run at each generation and calculate the average of those fitness values obtained through 30 simulation runs.

Table 1 presents the functions of non-terminal nodes and terminal nodes in the garbage collection problem. We have two kinds of nodes: 0 denotes non-terminal nodes (branch nodes), and 1 denotes terminal nodes (action nodes).

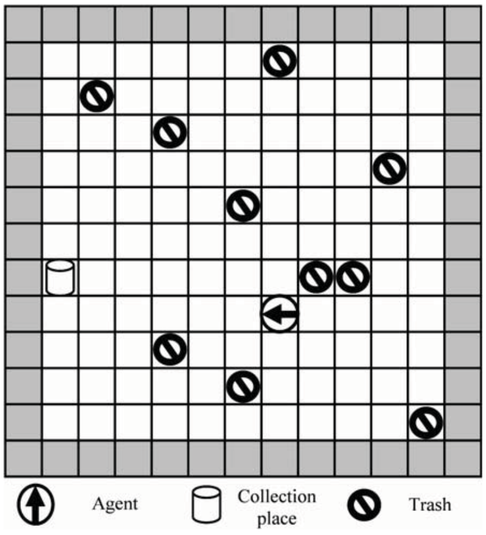

Fig. 2. An example of the field of the garbage collection problem.

Table 1. Functions of non-terminal nodes and terminal nodes for the garbage collection problem.

kind	function (number of edges)
0	check the distance from the agent to the dump site (3)
0	how many pieces of trash the agent has (3)
0	check the direction of the agent to the dump site (8)
0	check the direction of the agent to the nearest trash (9)
0	check the direction of the agent to the second nearest trash (9)
1	move forward (1)
1	turn right (1)
1	turn left (1)
1	stay (1)

5.2. Parameters

In the garbage collection problem, the population size is 300, the maximum number of generations is 1,000, and the number of trees in an individual *M* is 10. In addition, α is 10 because its value showed best performance in the parameter tuning. We employ the grow method as the generating method of the initial population, and set the maximum depth of trees to 6. Other parameter values used in the experiment are listed in Table 2.

Table 2. Parameters for the garbage collection problem.

	GP, GP_{CN}	GP_{CN}(e)	GP_{CN_I}	GP_{CN_M}	GP_{CN_IM}
Probability of crossover	0.8	0.8	0.95	0.8	0.95
Probability of mutation	0.05	0.05	0.05	0.05	0.05
Probability of mutation tree	0.1	0.1	0.1	0.05	0.05
Probability of inversion	0.2	0.2	0.2	0.2	0.2
Tournament size	2	2	2	2	2
Elite number	1	1	1	1	1
Probability of updating the number P	-	0.05	-	-	-

5.3. Performance Evaluation

First, to examine an effect of trees with a single node in GP_{CN}, we compare GP_{CN} using a proposed fitness function in this study with GP_{CN} using a fitness function in previous work[11,12]. The change of the number of trees with a single node in individuals obtained in 1,000 generations is depicted in Fig. 3 (a). We plot the average of the number of total trees in population obtained through 30 simulation runs. The maximum number of trees in population is 3,000 because 300 individuals with 10 trees are generated in this study. We see from Fig. 3 (a), that 1500 trees with a single node have been generated in the initial population in both fitness functions. With the fitness function in previous work, the number of trees with a single node temporarily increases until 50 generations because individuals with trees of a single node obtain high fitness values at early generations. Afterward, the number of trees has slowly decreased. However, 700 trees with a single node have still survived at generation 1,000 because they slowly decrease. With the proposed fitness function, the number of trees with a single node doesn't increase in contrast to the fitness function in previous work because a penalty value is given the fitness function. In addition, the number of trees quickly decreases to 250 at generation 200 and the number of trees is 50 at generation 1,000.

Let *T* denote the total number of trash carried to the dump sites, and we depict the change of *T* obtained in 1,000 generations in Fig. 3 (b). We plot the average of maximum *T* in population obtained through 30 simulation runs. With the fitness function in previous work, the increase of collected trash slow after the number of total pieces of trash is 200 because of increase of trees with a single node, as shown in Fig. 3 (b). With the proposed fitness function, the number of total pieces of trash largely increases to 60 at generation 1,000. Thus, these results suggested that the proposed fitness function is effective.

Next, we compare the proposed methods, namely GP_{CN} with inheriting the value of P (GP_{CN_I}) and GP_{CN} using mutation of the number *P* (GP_{CN_M}), with the previous methods, namely genetic programming (GP), GP with control node (GP_{CN}), and GP_{CN} updating the number *P* (GP_{CN}(e)). We evaluate these methods using *T* used in Fig. 3 (b). The change of *T* obtained in 1,000 generations is depicted in Fig. 4. We plot the average of maximum *T* obtained by proposed and previous methods through 30 simulation runs. In Fig. 4, GP_{CN}(e) proposed in the previous work shows better performance than GP and GP_{CN}. Thus, we

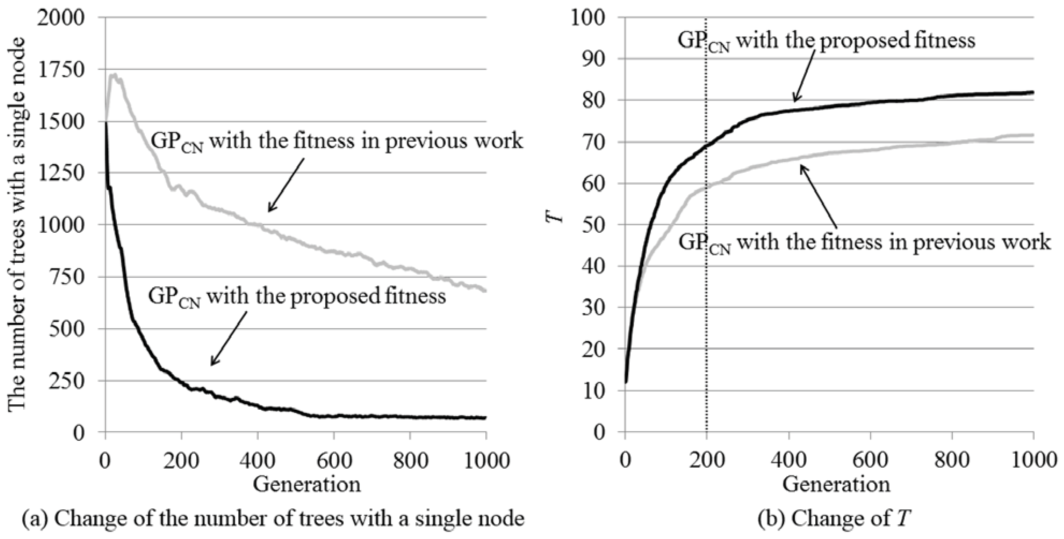

Fig. 3. Change of each number in the garbage collection problem obtained for 1,000 generations.

confirmed that the method to obtain *P* suitable to the tree in evolution is effective to improve the performance of GP$_{CN}$. GP$_{CN\_I}$ shows better performance than previous methods, namely GP, GP$_{CN}$ and GP$_{CN}$(e), and shows the best performance in the garbage collection problem. GP$_{CN\_M}$ shows better performance than previous methods.

In addition, with the standard deviation of maximum *T* obtained by the proposed and previous methods through 30 simulation runs, we evaluate these methods. The change of the standard deviation in 1,000 generations is depicted in Fig. 5. In Fig. 5, the standard deviation of GP$_{CN\_I}$, GP$_{CN\_M}$, and GP$_{CN\_IM}$ is 18 at generation 150, and the even of GP is 13 at generation 200. Thus, the proposed methods searched solutions in a way space than GP did, since the fitness values of them were dispersed. The fitness values of GP$_{CN\_I}$, GP$_{CN\_M}$, and GP$_{CN\_IM}$ converge by evolution because the standard deviation decreases as the generation increases after generation 150. The standard deviation of GP$_{CN\_I}$ decreases to 5.The standard deviation of GP$_{CN}$ and GP$_{CN}$(e) increases until generation 200. After generation 200, the standard deviation of these methods decreases. However, the decrease of their standard deviation is small. Thus, their fitness values are still dispersed in 1000 generations.

Therefore, we conclude that the proposed method for inheriting the number *P* converges the number *P* to the suitable value in

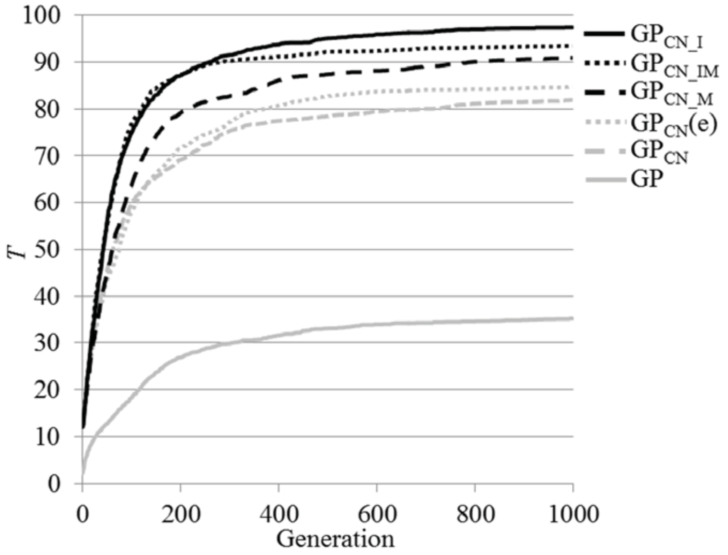

Fig. 4. Change of *T* obtained for 1,000 generations.

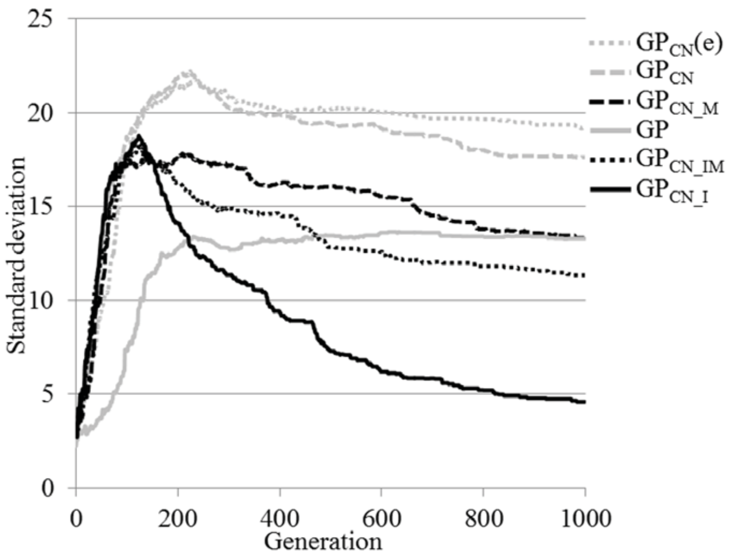

Fig. 5. Change of the standard deviation obtained for 1,000 generations.

GP$_{CN}$, the proposed method using mutation of the number P is better than the previous method for updating the number P, and the method that combines inheriting the number P with mutation of the number P is better than only using mutation of the number P, but the method shows lower performance than only inheriting the number P. The authors consider that the performance of the method is low because using mutation of the number P can not maintain the diversity of P.

6. Conclusion

In this paper, we proposed a new fitness function to solve the problem that trees obtained by GP$_{CN}$ tend to be trees with a single terminal node. In the garbage collection problem, trees with a single node are eliminated using the new fitness function, and GP$_{CN}$ with the proposed fitness function shows better performance than GP$_{CN}$ with the fitness function in previous work. Thus, the effectiveness of the proposed fitness function was shown.

In addition, we proposed two methods: GP$_{CN}$ with inheriting the value of P (GP$_{CN\_I}$) and GP$_{CN}$ using mutation of the number P (GP$_{CN\_M}$). They were compared with previous methods. GP$_{CN\_I}$ and GP$_{CN\_M}$ showed better performance than previous methods. Therefore, the methods for inheriting the number P and using mutation of the number P are effective to converge the number P in GP$_{CN}$. However, GP$_{CN\_IM}$ was lower performance than GP$_{CN\_I}$ by combination of inheriting the value of P and the mutation of the number P.

As our future works, we will propose the new method for maintaining the diversity, and apply each method to other problems.

Acknowledgements

This research was in part supported by a Hiroshima City University Grant for Special Academic Research (General).

References

1. Iba H. *Genetic algorithm*. Japan: Igaku Shuppan; 2002.
2. Koza JR. *Genetic programming: on the programming of computers by means of natural selection*. MA: MIT Press. Cambridge; 1992.
3. Iba H. *A primer of genetic programming*. Japan: Tokyo University Press; 2002.
4. Hirasawa K, Okubo M, Katagiri H, Hu J, Murata J. Comparison between genetic network programming and genetic programming using evolution of ant's behaviors. *IEEJ Transactions on Electronics, Information and System*. 2001. vol.121 (6). p. 1001-1009.
5. Tanji M, Iba H. A new gp recombination method using random tree sampling. *IEEJ Transactions on Electronics, Information and Systems*. 2010. vol.130 (5). p. 775-781.
6. Ono K, Hanada Y, Shirakawa K, Kumano M, Kimura M. Depth-dependent crossover in genetic programming with frequent trees. *2012 IEEE International Conference on Systems, Man, and Cybernetics*. 2012. p. 359-363.
7. Ono K, Hanada Y, Kumano M, Kimura M. Genetic programming for lighting control using frequent trees and depth information. *IEEJ Transactions on Electronics, Information and Systems*. 2013. vol.133 (11). p. 2044-2052.
8. Ono K, Hanada Y, Kumano M, Kimura M. Island model genetic programming based on frequent trees. *2013 IEEE Congress on Evolutionary Computation*. 2013. p. 2988-2995.

9. Nguyen QU, Nguyen TH, Nguyen XH, O'Neill M. Improving the generalisation ability of genetic programming with semantic similarity based crossover. *Genetic Programming 13th European Conference*. 2010. p. 184-195.
10. Galv´an-L´opez E, Cody-Kenny B, Trujillo L, Kattan A. Using semantics in the selection mechanism in genetic programming: a simple method for promoting semantic diversity. *2013 IEEE Congress on Evolutionary Computation*. 2013. p. 2972-2979.
11. Minesaki T, Ueda H, & Takahashi K. Island model genetic programming based on frequent trees. *The Conference Program of the 2009 (60th) Chugoku-branch Joint Convention of Institutes of Electrical and Information Engineers*. 2009. p. 546.
12. Ito T, Takahashi K, Inaba M. Experiments assessing learning of agent behavior using genetic programming with multiple trees. *The 6th International Conference on Agents and Artificial Intelligence*. 2014. p. 264-271.
13. Ito T, Takahashi K, Inaba M. Extension of Genetic Programming for Agent Learning. *The IEICE Transactions on Information and Systems*. 2015. vol.J98-D (6). p.905-915.
14. Ito T, Takahashi K, Inaba M. Extension of genetic programming with multiple trees for agent learning. *JOURNAL OF COMPUTERS*. 2016. vol.11 (4). p. 329-340.
15. Eto S, Mabu S, Hirasawa K, Huruzuki T. Genetic network programming with control nodes. *2007 IEEE Congress on Evolutionary Computation*. 2007. p. 1023-1028.