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.



Peer-review under responsibility of KES International

*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


Peer-review under responsibility of KES International
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
        ith tree–> P = Random.uniform( 1, Total Steps/M )
        Generate Tree( ith 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