Ph.D. Thesis
Representation Schemes for Evolutionary Automatic Programming

YABUKI Taro
Department of Frontier Informatics,
Graduate School of Frontier Sciences,
The University of Tokyo

Abstract

We propose a new representation scheme for Genetic Programming (GP). It is a recurrent network consisting of functions (Recurrent Network consisting of Trees, RTN). GP is a type of evolutionary computing (EC). EC is a framework of automatic optimization or design.

Features of EC are:

- Representation scheme for solution candidates and variation operators for them.
- Fitness function that shows a quality of the solution candidate.
- Selection method that selects prospective solution candidates from a set of them.

Usually, these features are defined independently. Although it is interesting to rethink this model completely, we reconsider only the representation scheme for GP.

We use GP to generate a program automatically. The program is represented by RTN. RTN can represent any algorithms, in other words, Turing-complete. Thus, a user of RTN need not worry about whether a solution of a given problem can be described by RTN. On the other hand, the expressiveness of solution candidates of standard GP, which is the most popular GP, is strongly restricted. A solution candidate of standard GP is represented by a single parse tree. The parse tree consists of terminals and non-terminals. If all non-terminals are pure functions and we treat an evaluated value of the parse tree as an output (or behavior) of the solution candidate, the repertoire of this representation is smaller than the one of finite state machine.

This does not matter if we know that the restricted expressiveness is sufficient to describe the solution of a given problem in advance of evolutionary computation. However, in case that we do not know that and the search should fail, it will be impossible to find out whether it is attributable to evolutionary computing or the representation scheme. For example, suppose we try to generate a classifier for the language $\{ww|w \in \{0,1\}*\}$. If we use a representation whose repertoire is the same as one of the pushdown automaton, then we will never succeed, because it is proved that any pushdown automaton cannot decide this language.

One conceivable approach is to introduce an ideally infinite indexed memory and non-terminals to access to it. It is proved that if the solution candidate is represented by a parse tree consisting of these non-terminals and we can repeat the evaluation of the parse tree until the data stored in the memory meets a halting condition, then the expressiveness is equivalent to the one of a Turing machine, i.e. Turing-complete.

However, the representation scheme for GP must satisfy other conditions except Turing-completeness. These are as follows:

- User can specify the input-output scheme.
- Useful components can be introduced easily.

- The program is modularized and hierarchised.

We will discuss necessity of them in detail in this paper.

We propose another representation scheme, RTN. It satisfies above conditions. It is a natural extension of standard GP. Standard GP uses a single parse tree to represent a solution candidate. On the other hand, RTN is a recurrent network consisting of plural nodes. Each node consists of a value and a pure function represented by a parse tree. The parse tree consists of non-terminals and terminals. Special non-terminals are not needed. Terminal set consists of four variables and constants. In case of standard GP, the input data bind variables. On the other hand, they bind the values of the RTN nodes.

This is an example of RTN consisting of two nodes. The functions of each node are

$$\#1 : (c - P[c])/2,$$
$$\#2 : P[a]d,$$

where P is a procedure which returns a remainder of its argument divided by 2. The function has at most four parameters, i.e. a, b, c, and d. These parameters are bound to the value of nodes. In this case, the binding rule is expressed as follows: Links of #1 and #2 are $\{*, *, 1, *\}$ and $\{1, *, *, 2\}$, respectively. The third parameter of #1, i.e. c and the first parameter of #2, i.e. a are bound to the value of #1, because both the third link of #1 and the first link of #2 are 1. The fourth parameter of #2, i.e. d is bound to the value of #2, because the fourth link of #2 is 2.

The program is executed according to the discrete time steps. Define the function and the value of the $\#n$ at time t as f_n and $v[[n, t]]$, respectively, the number of parameters as k_n, and let i-th parameter be bound to the value of $\#l_{n,i}$. The value at $t + 1$ will be

$$v[[n, t + 1]] = f_n[v[[l_{n,1}, t]], \cdots, v[[l_{n,k_n}, t]]].$$

Suppose the value of #1 is bound to the input data and the value of #2 is 1 at $t = 0$. For example, when the input data is a binary digit 1011, the transition of RTN will be as follows:

Value of #1	1011	101	10	1	0
Value of #2	1	1	1	0	0

When the value of #1 becomes 0, the value of #2 is 0 if and only if the inputted binary digit contains 0.

It is straightforward to prove that RTN can simulate any Turing machine, in other words, RTN can represent any algorithms. We give the proof in this paper.

We use RTN to generate language classifiers. These are the tasks that GP has failed to solve. GP using RTN succeeds to solve them. We also apply the representation scheme using indexed memory to these tasks. However, it does not succeed. This comparison implies that our approach is effective and promising.

Various representations for GP have been proposed so far. We discuss differences between RTN and other representation scheme. We also discuss the criteria used in comparing various approaches. For example, No Free Lunch Theorem does not matter.