IDEA GROUP PUBLISHING

701 E. Chocolate Avenue, Suite 200, Hershey PA 17033-1240, USA ITB10415
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com

Chapter IV

Genetic Programming Using a Turing-Complete Representation: Recurrent Network Consisting of Trees

Taro Yabuki, The University of Tokyo, Japan

Hitoshi Iba, The University of Tokyo, Japan

ABSTRACT

In this chapter, a new representation scheme for Genetic Programming (GP) is proposed. We need a Turing-complete representation for a general method of generating programs automatically; that is, the representation must be able to express any algorithms. Our representation is a recurrent network consisting of trees (RTN), which is proved to be Turing-complete. In addition, it is applied to the tasks of generating language classifiers and a bit reverser. As a result, RTN is shown to be usable in evolutionary computing.

INTRODUCTION

Genetic Programming (GP) is a technique for generating programs or functions automatically (Koza et al., 1999). GP is a type of evolutionary computing that aims to solve problems through the repetition of modification and selection of prospective solution candidates. Various representations for programs or functions have been used. The most

popular GP (standard GP) uses a single parse tree (S-expression) as a program representation. S-expressions (e.g., plus (times *x* 5) (minus (*y x*))) are made by combining nonterminals (e.g., plus, minus, times, and divide) and terminals (e.g., *x*, *y*, and integers).

We have proposed a substitution for the single S-expression. It is a recurrent network consisting of trees (RTN). In the following paragraphs, we will explain why a new representation is required for GP.

When using GP, we must set various configurations. For example: the representation of individuals, the components of representation, the way to use the representation and evolutionary operators, and so forth. The strategy to set these configurations depends on the objective tasks.

The objective tasks of GP can be classified as follows:

(1) Programs that can be easily written by humans.
(2) Programs that are simpler or more efficient than those written by humans.
(3) Programs that solve unsolved problems.

If the task belongs to either the first or second class, then the configurations can be decided with reference to the previously known solution. However, if the task belongs to the third class, then it is not easy to set the configurations. Choosing a smaller nonterminal set and restricting the expressiveness of individuals may make the search easier. However, should the search fail, it will be impossible to find out whether it is attributable to GP or the configurations. 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.

One conceivable approach is to start with simple settings and gradually introduce complex ones. One method proposed composes the S-expression of basic arithmetic functions in the early stage, and then introduces a loop or recursion as the search progresses (Koza et al., 1999). A strategy like this is adoptable for a task belonging to the first or second classes mentioned above, but not for the unsolved problems, because it is not clear how a loop or recursion affects the expressiveness of individuals. For unsolved problems, a strategy that can confirm the increase of expressiveness is desirable. In the ideal case, the expressiveness of an individual finally becomes Turing-complete. In other words, an individual will be able to express any algorithms.

Additionally, there are other requirements for the new representation for GP.

(1) *Simplicity:* For example, representations that use too many terminals are not easy to use.
(2) *Extensibility:* If it is known that some functions, for example, trigonometric functions, are essential for the problem, then it must be easy to add these functions.
(3) *Similarity to standard GP:* A natural extension of the standard GP is desirable, because of its widespread use.

The RTN proposed in this chapter meets those requirements.

This chapter is organized as follows. In the second section, the expressiveness of the standard GP is summarized. In the third section, RTN is proposed with an example. In the fourth section, the proof that RTN is Turing-complete is given. In the fifth section, evolutionary operators for RTN are reviewed. In the sixth section, RTN is applied to