論文内容の要旨

論文題目　進化計算による自動プログラミングのための表現の研究

氏名　矢吹太朗

Representation Schemes for Evolutionary Automatic Programming

We propose a new representation scheme for Genetic Programming (GP). It is a recurrent network consisting of functions (recurrent tree network, 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.

We propose another representation scheme, RTN. 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

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.

遺伝的プログラミング (Genetic programming, GP) のための新しい表現形式を提案する。関数の回帰的なネットワーク (recurrent tree network, RTN) である。GP は進化計算 (evolutionary computing, EC) の一種で、プログラムの自動生成に用いられる。EC は自動的な最適化やデザインのフレームワークである。

EC は次の要素で特徴付けることができる。

- 解候補の表現形式と解候補を改変するためのオペレータ
- 解候補の質を定義する適合度関数
- 解候補の集合の中から有望なものを選び出す選択方法

通常これらの特徴は独立に定義される。このようなモデル自体を考え直すのも興味深いことではあるが、ここでは GP のための表現形式のみを研究対象とする。

GP を利用する際に、われわれは RTN を用いてプログラムを表現する。RTN は任意のアルゴリズムを表現できる、つまりチューリング完全な表現形式である。そのため、RTN の利用者は問題の解が RTN で表現可能かどうかを心配する必要はない。その一方で、最も普及している標準的な GP においては、プログラムは単一の構文木で表現され、その表現力は強く制限されたものになっている。例えば、構文木を構成する非終端ノードが純関数で、構文木を評価した結果をプログラムの出力（または振舞い）と見なすならば、この表現のレパートリーは有限状態機械のものよりも小さくなる。

制限された表現力でも与えられた問題の解を記述するには十分だということを、計算に先立って知っているならば、このことは問題にはならない。しかしながら、もしこの十分性を知らないで、しかも探索に失敗した場合、失敗の原因が進化計算にあるのか表現形式にあるのかを知ることはできない。例えば、$\{ww|w \in \{0,1\}*\}$ という言語を判定するプログラムを生成したいとしよう。もしレパートリーがプッシュダウン・オートマトンのそれと同じ表現を用いたならば、探索は決して成功しない。プッシュダウン・オートマトンではこの言語を判定できないことが示されているからである。

考えられる解決法として、理想的には無限の番地付きメモリとそれにアクセスするための非終端ノードを導入するというものがある。そのような非終端ノードからなる構文木で解の候補を表現し、特定のメモリの値が停止条件を満たすまで構文木の評価を繰り返すならば、表現力はチューリング・マシンと同等つまりチューリング完全になることが示されている。

ここでは別の表現形式、RTN を提案する。RTN は標準的な GP の自然な拡張である。標準的な GP は解の表現として単一の構文木を用いる。一方、RTN は回帰的なネットワークである。ネットワークは複数のノードで構成されるが、個々のノードは値と構文木で表現される純関数を持つ。構文木を構成するのに特別な非終端ノードは必要ない。終端ノードは 4 つの変数と定数のいずれかである。標準的な GP においては、プログラムへの入力は変数に代入されるが、RTN においてはノードの値に代入される。

RTN の例を示す。2 つのノードからなる RTN である。各ノードが持つ関数は、

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

である。ここで P は 2 で割った余りを返すような手続きである。関数は最大で 4 つの引数 a, b, c, d を持つ。これらの引数にはノードの値が代入される。代入の仕方は、#1 については {*,*,1,*}、#2 については {1,*,*,2} のように表現される。#1 の 3 番目の要素と #2 の 1 番目の要素はともに 1 であるから、#1 の第 3 変数つまり c と #2 の第 1 変数つまり a には #1 の値が代入される。#2 の 4 番目の要素は 2 であるから、#2 の 4 番目の変数つまり d には #2 の値が代入される。

プログラムは離散的なタイム・ステップに従って実行される。ノード#nが持つ関数をf_n、時刻tでの値を$v(n,t)$とする。関数f_nが持つ引数の数をk_nとし、i番目の引数には#$l_{n,i}$の値が代入されるものとする。時刻$t+1$における#nの値は、

$$v(n,t+1) = f_n(v(l_{n,1},t), \cdots, v(l_{n,k_n},t))$$

となる。プログラムへの入力は#1の値に代入され、#2の値は$t=0$において1とする。例として、プログラムへの入力が2進数1011だったとする。RTNの遷移は次のようになる。

#1の値	1011	101	10	1	0,
#2の値	1	1	1	0	0.

#1の値が0になったとき、プログラムへの入力が0を含んでいた場合に限って#2の値が0になる。

RTNが任意のチューリング・マシンをシミュレートできること、つまりRTNが任意のアルゴリズムを表現できることは簡単に示すことができる。証明は本文中で与える。

応用例として、RTNを用いたGPによる言語判定装置の自動生成を試みる。これは従来のGPが失敗していた課題である。RTNを用いたGPはこの課題に成功する。比較実験として、番地付きメモリを用いたGPでもこの課題を試みるが、これは成功しない。このことから、一般的な自動プログラミングにおけるRTNの有効性が期待される。

GPのための表現形式は他にもさまざまなものが提案されている。それらとRTNの違いについても議論する。また、違いを議論する際に用いる基準についても考察する。例えば、No Free Lunch Theoremは重要ではないことがわかる。