オブジェクト指向に基づいた遺伝的プログラミングシステム

田中正造† 中道義之† 有田隆也‡

名古屋大学大学院人間情報学研究科† 名古屋大学大学院情報科学研究科‡

1 はじめに

遺伝的プログラミング(GP)[1,2]は，遺伝的アルゴリズムにおける個体表現を構造化したものであり，広範に応用されるようになってきた．本稿では，オブジェクト指向パラダイムを導入することによる進化性能，進化速度，及び，システム構築効率の向上を目的とした遺伝的プログラミングシステム OOGP（Object Oriented Genetic Programming）に関して述べる．

2 OOGP の概要

2.1 設計方針

本研究において，オブジェクト指向パラダイムを遺伝的プログラミングシステム OOGP の構築に利用した理由は，1) GP における進化性能の向上，2) GP における進化速度の向上，3) OOGP の構築効率の向上の３点である．第１の点は，オブジェクト指向の概念や処理を GP のアルゴリズムの改良に活用することを意味するものであり，本稿では，従来のエリート選択とは別に，普遍的に優良な個体を持続的に保持する「エリートプール」を導入した遺伝的操作をオブジェクト指向に基づく処理で実現することを検討する．優れたスキーマを活用する同時に進化の方向性を安定させることを狙う．第２の点は，動的コンパイルの技術を利用することにより個体の評価を高速化することに基づくものである．適応度評価に時間がかかるような場合，あるいは，多くのデータを与えて何回も評価を繰返す場合に性能が向上することが期待できる．第３の点は，オブジェクト指向に基づくプログラム開発効率の一般的な向上を期待することである．

2.2 基本アルゴリズム

OOGP のアルゴリズムの特徴はエリートプールの利用にある．優良な個体(木)が生成された場合にエリートプールに保存しておき，各個体がエリートプールに保存された木もしくはその部分木を呼び出して利用することができる．エリートプールに保存された木は，自身とその部分木の評価値(エリート評価値と呼ぶ)を保持している．

以下に OOGP の処理の流れを示す．

[1] 初期設定：初期個体を生成する．

[2] 進化的操作：[2.1]から[2.4]までの処理を定められた世代まで繰り返し行う（図 1）．

[2.1] 適応度の評価：集団の個体を評価し適応度を求める．

[2.2] エリート評価値の計算：個体がエリートプールの木を利用している場合は，その個体の評価値をエリート評価値とする．

[2.3] エリートプールへの複製:

次式で示す $f_d(t)$ が閾値 T_f 以上であった場合，その世代の最良の個体をエリートプールに複製する．

$$f_d(t) = \log_{f_{gbest}} f_{best}(t) - 1 \qquad (1)$$

f_{gbest} はそれまでに得られた最良の個体の適応度，$f_{best}(t)$ はその世代 t で得られた最良の個体の適応度である．

[2.4] 次世代個体生成：[2.4.1]から[2.4.4]までの処理を繰り返し行い次世代の個体を生成する．また，この世代における最良の個体は次世代の個体とする(従来のエリート選択)

[2.4.1] 選択：[2.1]で得られた適応度に基づいたルーレット選択により 2 個体選択する．

[2.4.2] 交叉：選択された個体同士を交叉して新たな個体を 2 個体生成する．

[2.4.3] 突然変異(1)：[2.4.2]で生成された個体に，確率 m1 で突然変異を起こす．ここで起こる突然変異は通常の GP における突然変異と同様の処理であり，ランダムに選ばれた部分木をランダムに生成された木で置換するという操作である．

[2.4.4] 突然変異(2)：確率 m2 で突然変異を起こす．ここで起こる突然変異はエリートプールに保存されている木の利用に関するものであり，ランダムに選ばれた部分木を，エリートプールから選ばれた木または部分木で置換するという操作である．加える木はエリート評価値に応じて選択される．なお，エリート評価値が定まっていない木がエリートプールにある場合には優先的に選択される．

[3] 解の出力：これまでに得られた最良の個体を出力する．

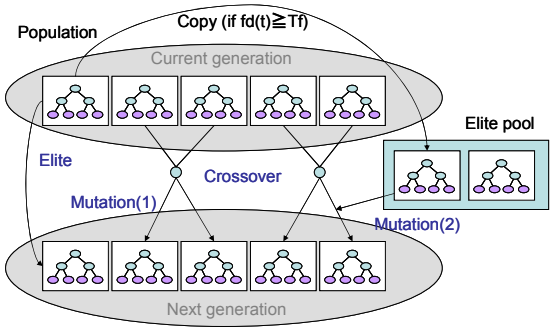

図 1：OOGP の遺伝的操作

2.3 オブジェクト指向による実現

オブジェクト指向パラダイムに基づき，2.2 節で述

A Genetic Programming System based on Object Oriented Paradigm

†Shozo Tanaka, Yoshiyuki Nakamichi, Graduate School of Human informatics, Nagoya University

‡Takaya Arita, Graduate School of Information Science, Nagoya University

べたアルゴリズムを基本とする OOGP を C#言語（ECMA-334）を用いてインプリメントした．進化させる基盤となる集団の個体をサブクラス，エリートプールの個体をスーパークラスとしてクラス化する．クラスのメソッドは個体の木から生成する．エリートプール個体の部分木からは仮想メソッドを生成し，部分木を評価するためにサブクラスのメソッドから仮想関数呼び出しを追加するソースコードを生成する．図 2 に生成されるソースコードの例を示す．このようにして生成されたソースコードを動的にコンパイルし，リフレクションにより生成されたモジュールを実行する．

```
public class gpSuper0
{
  public virtual float gp0(float x) {
    return ( 0.5 * 1.5 ) - ( 4.0 + x );
  }
  public virtual float gp1(float x) {
    return ( 0.5 * 1.5 );
  }
}
```

(a)エリートプールの木

```
public class gpSub0 : gpSuper0
{
  public override float gp0(float x)
  {
    return base.gp0( x );
  }
}
```

(b)集団の個体

図 2: 生成されるソースコード

3 評価実験

3.1 進化性能の評価

OOGP の基本的な性能を評価するために，Sin 関数と 4 ビットパリティ関数（入力 4 ビット中の 1 のビット数が奇数なら 1，偶数なら 0 を出力）の近似関数を合成する実験をおこなった．特に，エリートプール利用の効果を確認するために，T_f を変化させて実験を行った．また，比較のため T_f が∞（エリートプールにエリート個体を全く複製しない）の場合の実験を行った．両者とも個体数 200 で世代数は 50 世代を 1 試行とし，10 試行の結果（適応度）の平均を表 1 に示す．

Sin 関数の実験では，非終端記号には+,-,*,/，終端記号には変数 x と[-9.0,9.0]の定数を使用した．y=sin(x)に対し 64 組(x と y)のテストデータを用意し，T_f は 0,0.01,0.02,∞，m1 は 0.5，m2 は 0.3 と設定し，適応度（誤差の合計の逆数)を計測した．最もよい結果を出したのは T_f を 0.02 とした場合であった．

4 ビットパリティ関数に関する実験では，非終端記号には AND,OR,NAND,NOR,?:，終端記号には d0～d3 を使用した．T_f は 0，0.01,∞，m1 と m2 は 0.3 と設定し，適応度（エラー率の逆数）を計測した．T_f を 0.01 とした場合が最もよい結果となった．

どちらの実験においても，T_f が 0.02 のときに，エリートプールを使わない場合(T_f =∞)よりも良い結果を得た．しかし，T_f が 0 の場合はエリートプールを使わない場合と同等の性能である．これは，前世代との適応度差がある程度以上ある場合のみエリートとしてエリートプールに格納するという本方式の有効性を示している．

表 1: 実験結果(Sin 関数, 4 ビットパリティ)

T_f	Sin 関数 （誤差の合計）	4 ビットパリティ （エラー率）
0	11.9	0.037
0.01	11.6	0.012
0.02	7.89	0.012
∞	9.36	0.042

3.2 進化速度の評価

動的コンパイルの効果を確かめるために従来の手法と今回の提案するモデルの速度を比較する実験をおこなった．Sin 関数の近似問題（テストデータ数 512）で実験をおこない． 1 世代の処理にかかる時間を計測した．実験結果を図 3 に示す．130 世代までは従来方式（インタプリタ型）方が速いが，それ以降は本方式（コンパイラ型）の方が速い．世代を重ねることに木が複雑になり，従来方式ではそれを処理する時間がかかるが，本方式ではコンパイル済みのモジュールを実行するために処理が高速になると考えられるからである．

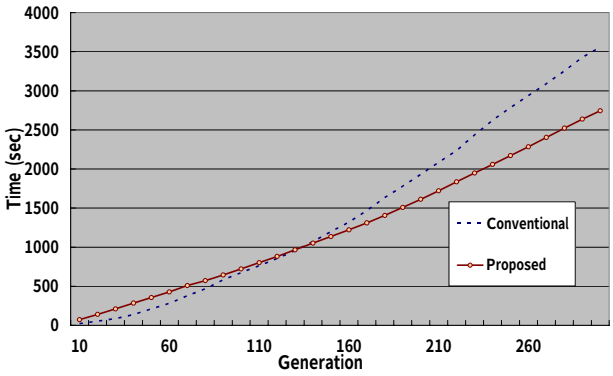

図 3: １世代あたりの処理時間

4 終わりに

オブジェクト指向パラダイムを利用した遺伝的プログラミングシステム OOGP に関して述べた．前世代と現世代の優良個体の適応度の差が一定値を超えたときだけ，エリート個体をエリートプールに格納し，エリートプール個体の優れた部分木との進化操作をオブジェクト指向フレームワークに適用することで，進化性能が向上することが示された．適応度評価に時間がかかるように解を求める場合には，動的コンパイルにより処理の時間が減少することが示された．

参考文献

[1] John Koza, “Genetic Programming: On the Programming of Computers by Means of Natural Selection”, MIT Press, 1992.

[2] John Koza, “Genetic Programming II: Automatic Discovery of Reusable Subprograms”, MIT Press, 1996.