2F1-05

Robocode Tank Fighters プログラムの進化論的計算手法による学習に関する研究

Research on study by the evolutionary calculation technique of a Robocode Tank Fighters program

村上 幸一[*1]
Yukikazu Murakami

徳永 秀和[*2]
Hidekazu Tokunaga

[*1] 高松工業高等専門学校 電気情報工学科
Takamatsu National College of Technology
Dept of Electrical & Computer Engineering

[*2] 高松工業高等専門学校 制御情報工学科
Takamatsu National College of Technology
Dept of Electro-Mechanical Systems Engineering

It this paper, we focus on the automatic programming by Genetic Programming(GP). GP is what extended the genotype of Genetic Algorithm. GP can use the Symbolic expressions. Therefore GP is suitable for automatic programming. We evolve the action of the robot in the virtual space called Robocode. The validity of GP is verified by comparing with the program before applying Genetic Programming.

1. はじめに

本研究では，汎用性の高いプログラムの自動生成を目的として，進化論的計算手法を用いた実験を行なった．

進化論的計算手法とは，生物進化のメカニズムを真似た，データ構造を変形，合成，選択する手法であり，最適化問題の解法，人工知能の学習･推論，プログラムの自動生成などに広く応用が可能な手法である．それらは，遺伝的アルゴリズム（GA），遺伝的プログラミング（GP），進化論的戦略（ES），進化論的プログラミング（EC）の4つの手法に，大きく区別することができる．そこで本研究では，進化論的計算手法の中でも，特にプログラム生成への適用が可能な GP を，Java 学習ソフトとしても用いられている，“Robocode Tank Fighters プログラム” [可知 2003]へと，適用することにより，プログラムの自動生成についての実験を行なっていく．

遺伝的プログラミング（以下 GP）とは，遺伝的アルゴリズム（以下 GA）を拡張した手法であり，GA 同様，生物遺伝子の突然変異，交叉，および選択淘汰をモデルにしたデータ操作手法を用いて操作を行なうが，GA との大きな違いは，木構造を扱えるようにしたことにより，LISP 言語における表現木（S 式）が記述できるようになった点である．LISP 言語とは 1960 年代に MIT において開発されたプログラム言語であり，多くの人工知能における研究に用いられてきた経緯がある．本研究では，この LISP の S 式をもとにして定義したユーザー関数を使用することにより，GP の Javaプログラム生成への適用を行なう．

しかし，GPによるプログラムの自動生成においては，S 式およびアトム(木構造表現における末端ノード)の組み換えの際になんら制限を加えなければ，意味の成さないソースコードが生成されてしまう可能性が高く，実用的かつ汎用性の高いソースコードの生成は極めて難しいといった欠点がある．

そこで本研究では，関数およびアトムの組み換えの際に簡単な制約条件を加えることにより，実用性の高いプログラムの生成を目指した．予備知識がある場合の学習効果は，予備知識がない場合に比べて効果が高いことが予測されるが，本研究ではこれを，関数およびソース組み換えの際の制約条件という形で付帯することにより，汎用性の高いプログラムの生成のための原理についても合わせて解明していくことを狙いとしている．

連絡先：高松工業高等専門学校　〒761-8058 高松市勅使町 355　E-Mail: {murakami,togunaga} @takamatsu-nct.ac.jp

2. 実験システム

システムは，Robocode Tank Fighters プログラム，および，(1)Robocode インターフェース部，(2)ドライバー部，および(3)遺伝的プログラム部の，大きく 3 つにわけられる．（図1）

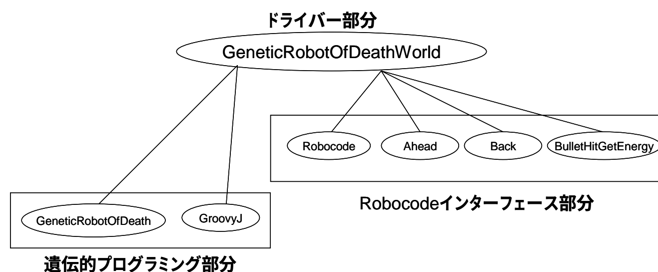

図1 システム構成

2.1 Robocode Tank Fighters プログラム

Robocode Tank Fighters (以下 Robocode) では，プレイヤーが自分でロボットをプログラミングする対戦ゲームの一種であり，プログラミング言語に Java を用いている．Robocode は，ロボット同士による対戦を行なうゲームプログラミングであるが，これはユーザーが操縦することによる対戦ではなく，あらかじめ作成されたプログラムに従ってロボット自身が自律的に対戦を行なうものであり，高度なプログラムを構築すれば，相手の動きに合わせて複雑な動きをすることも可能である．

ロボットは戦車を模したもので，車両，大砲，レーダーといった3つの構造に分かれており，作成されたロボットはバトルフィールドと呼ばれる戦闘区域内に配置され，その領域内で対戦する（図2）．そしてレーダーにより敵ロボットの位置を把握し，弾丸を撃ち，敵を撃破する．更に，高度なプログラミングを構築することにより，敵ロボットの動作を予測したり，敵ロボットの弾丸をかわしたりすることが可能となる．

Robocode のプログラムは Robocode API により基本的な動作プログラムを提供されている．そのため部品（メソッド）の組み合わせれば動作するといった簡単な仕様となっている．

図2　Robocode Tank Fighters プログラム

2.2 Robocode インターフェース部

Robocode インターフェース部では，Robocode とドライバー部との接続を行なう．Robocode プログラム実行用の battle ファイルをあらかじめ生成しておくことにより，Robocode インターフェース部を通じて，ドライバー部から，ロボットを対戦させることが可能となる．また，逆に Robocode の戦いによって生成された結果ファイルから，GP の評価値として使用するための情報を形成し，解析することもできる．

2.3 ドライバー部

ドライバー部では，システムパラメータの設定を行なうことができる． ドライバー部では，ユーザーが指定したパラメータをとり，GP をセットアップする．このパラメータは個体数や世代数であり，またユーザーがなにも指定しなければデフォルトの値をとる．本研究では，個体数を 10，世代数を 1-3 の間で変化させるものとした．また，使用する関数のセットとしては，①ahead，back などといった，ロボット自身を行動させる関数，②add,sbtract などといった四則演算および数値を定義する関数，③if，equal などといった条件文に関係する関数，の3種類を用いるものとした．

2.4 遺伝的プログラミング部

遺伝的プログラミング部では，子ロボット(現在のロボット GP に適用して新しく生成するロボット)のコードを任意に数回修正する．また子ロボットの中で成績のよかったものを選択する．そして，Robocode インターフェースを使用してプロセス(対戦)を繰り返していく．子ロボットでの対戦が終われば再度遺伝的プログラミングを行い，孫ロボットを生成し，対戦させる．

3. シミュレーション

3.1 学習目標

Robocode における強力なロボットをプログラミングするためには，

(1) 敵ロボットへ確実に弾丸を命中させることができる．

(2) もしくは敵の弾丸をかわすことのできること．

といった 2 条件が存在し[可知 2003]，少なくとも 1 条件を具備しておく必要がある．そこで，本シミュレーションにおいては，“ 敵のエネルギーの変化(弾丸の発射)を感知して，回避行動をとる(図3)．”ロボットを GP によりプログラミングすることを目標としてシミュレーションを行なっていく．Robocode API においては敵の弾丸発射を検知するメソッドは実装されておらず，意味のある関数の創出といった観点から，これを学習目標プログラムとするものとした．

```
public void onScannedRobot(ScannedRobotEvent e) {
    fire(1);
    turnRight(90 + e.getBearing() + moveDirection * 15);

    // 敵エネルギーの変化で、攻撃を察知
    double changeInEnergy = enemyEnergy - e.getEnergy();
    if (changeInEnergy != 0) {
            moveDirection *= -1;
            ahead(100 * moveDirection);
    }
    enemyEnergy = e.getEnergy();
}
```

図3　学習目標プログラム

3.2 シミュレーション方法

撃破すべき敵ロボットとして，バトルフィールドの中央部分を上下に移動しながら，弾丸を発射するロボット(以後 Center)を作成し，Center と 1 対1の対戦を3回行なわせて，3回の対戦の合計得点を各個体の評価値として用いるものとした．また，簡単な制約条件として，関数の型によるチェックを行なっていくとともに，表現木を増やすために，システムの用意する TwoThings 関数を使用するものとした．

4. シミュレーション結果

制約条件をつけない場合の初期生成プログラム(図5)および，制約条件をつけた場合の初期生成プログラム(図6)を以下に示す．2つのプログラムの比較から，制約条件がない場合には，意味のないプログラムが生成されているのに対して，制約条件をつけた場合に生成されるプログラムにおいては，fire,ahead,turnRight などといった，Robocode API の提供するメソッドが出現していることがわかる．また，2つを比較した場合，制約条件をつけた場合に生成されたプログラムが，より学習目標プログラム(図3)に近いソースコードの記述となっていることもわかった．

制約条件を付帯した場合に生成されたプログラム同士に，交叉を行わせた場合のプログラムの学習の一例を図4に示す．ここから，1 世代目の 9 番目の個体(1-9)のプログラムが，2世代目以降優勢となり，3 世代目にはすべてのプログラムが 1-9 と同じプログラムを持つ結果となることがわかった．

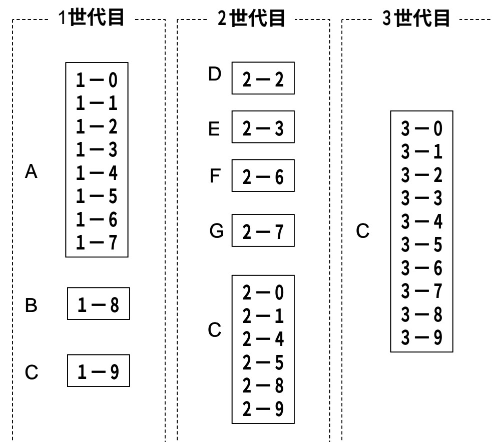

図4　各世代における生成プログラムの類似性

```
public void onScannedRobot(ScannedRobotEvent e){
    ((((((-1 + 1) * (0 + e.getEnergy())) - ((1 + -1) + (90 - 90))) -
    (((1 + 1) - (0 + 90)) * ((-1 * 100) - (15 * 1)))) + ((((100 * 0) -
    (90 - 0)) + ((0 * 100) * (1 + e.getBearing()))) - (((15 - 15) *
    (e.getEnergy() + e.getBearing())) - ((e.getBearing() + e.getBearing())
    + (1 + e.getBearing()))))));
}
```

図5 制約条件がない場合の初期生成プログラム

```
public void onScannedRobot(ScannedRobotEvent e){

fire((((1 * e.getEnergy()) * (0 - -1)) + ((e.getBearing() - e.getEnergy()) - (e.getBearing() + 100))));
if ((((e.getBearing() + 90) * (e.getEnergy() + 0)) == ((100 + -1) - (90 * 100)))) {
    ahead(((e.getEnergy() + e.getEnergy()) + (0 * 100)));
}
else {
    turnRight(((e.getBearing() * 90) + (e.getBearing() + e.getBearing())));
}
;

}
```

図６　制約条件がある場合の初期生成プログラム

5. 考察

シミュレーションから，制約条件の付与が GP によるプログラム生成の際に有用であることがわかった. また，図4のようにプログラムが 3 世代で収束してしまった原因としては，初期個体が10と少数であったことが，原因の一つとして考えられる.

参考文献

[可知 2003] 可知豊： Robocode＆ゲームプログラミング学習術, ソシム株式会社 ,2003 年.

[伊庭 2001] 伊庭済志： 遺伝的プログラミング入門, 東京大学出版会, 2001 年.

[筒井 2004] 筒井信一郎, 徳永秀和： Genetic Programming による Robocode Tank Fighters の自動プログラミング 高松高専専攻科特別研究論文集 2004 年