人工蟻の探索への GP（遺伝的プログラミング）の利用

T01K070F　濱本和志

1．GP（遺伝的プログラミング）とはなにか

GP とは GA（遺伝的アルゴリズム）を拡張したものである。
GA とは、進化論的な考え方に基づいてデータを操作し、最適化の問題や学習、推論を扱う手法。
GA で扱う情報は、PTYPE と GTYPE の二層構造からなる。

GTYPE 遺伝子型	低レベルの局所規則の集合で、細胞内の染色体に相当する。GA のオペレータの操作対象となる
PTYPA 表現型	環境内での発達に伴う行動や構造の発現を表す。環境に応じて PTYPE から適合度が決まる。

GP はこの GTYPE を拡張し、構造的な表現（木構造）を扱えるようにしたものである。
構造的表現を扱うことで GP は GA に比べ探索のための的確な部分構造の把握ができ、さらにより高次の知識の適応的な学習システムの構築を行うことが出来る。

2．人工蟻の探索問題

GP の利用として人工知能の問題を考える。
人工知能の問題では記号的な構造表現がしばしば登場する、これは知識表現として木構造が便利であるからである、複雑な数式や概念、関係などを木構造で表現するので GA より GP の方が適している問題である。

人工蟻の探索の概要

人工蟻とは与えられた空間を自由に動き回る人口生命である、蟻は一定のエネルギーをもっていて、空間には餌が配置されておりエネルギーを効率的に使いながら餌を探す問題である。

人工蟻の探索の空間

人工蟻の探索空間は16×16のマス目によって構成されている。ここには餌が32箇所配置されており、この空間で人工蟻（Ant）は限られたエネルギーの中で餌の探索が出来る。

人工蟻の動作

・Ant は向きを変えつつ前進する
・斜めには進めない
・壁を越えて進むことは出来ない
・Ant は動作を行うたびにエネルギーを1消費する
・Ant はセンサを持ち、自分の前方に餌があるかどうかが分かる

人工蟻の探索プログラムの関数、終端記号

関数（非終端記号）

If_Food_Ahead()・・二つの引数を持ち目前に餌があるとき第1引数、ないとき第 2 引数を実行する
PROG2()・・二つの引数を持ち第 1、2引数を順に実行
PROG3()・・三つの引数を持ち第 1、2、3引数を順に実行

終端記号

RIGHT・・右に90度向きを変える
LEFT・・左に90度向きを変える
FORWARD・・一歩前に進む

適合度の評価

人工蟻の探索の適合度は次のような流れで評価する 1. Energy＝90、food＝0 とする。さらに Ant のための環境の初期化をする
2. プログラムに従って Ant を動かす。FORRWARD、LEFT、RIGHT という終端記号を評価したら、Energy＝Energy－1とする
3. もし Ant の位置に餌があれば、Food＝Food+1とし餌を消す
4 .Energy が0なら、32－Food を適合度として返す
5. .2 に戻る
適合度は値が小さくなるほどよくなるとする。

交叉の親選択技法

今回のプログラムでは選択技法として以下のものを用いる。

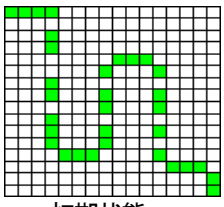
エリート戦略・・・成績の良い親を常にコピーして、次の世代に残す方式、これは一般に探索能力が優れているとされるが、親の成績をソートする手間がかかること、さらに探索が局所解に陥る場合があるなどの問題点も指摘されている。

基本パラメータ

集団数・・・500
世代数・・・100
交叉確率・・・80%
突然変異確率・・・10%
深さ制限・・・30

3．実行と結果

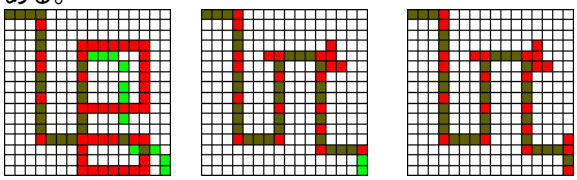
今回の実験では餌を次の図のように配置し行った。



- 初期状態

結果

次に示す結果は交叉にエリート戦略を用いたものである。

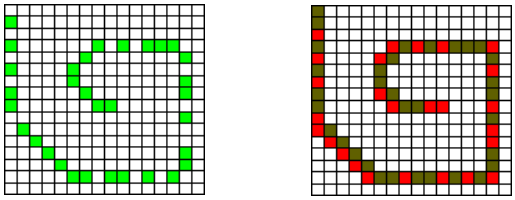


10 世代　50 世代　100 世代

適合度が0になる最適解が得られた。
しかし適合度が0になる個体が必ずしも得られるわけではなかった。
1000 回実行し最適解が何回得られたか数えた結果 386 回最適解が得られた。

頑強性の実験

100世代目に得られた個体の木構造が訓練された環境とは異なる環境で頑強性があるかどうかの実験を行った。
下は餌を配置し直した図とその結果の一部である。
実験には実験で得られた最適解を使った。



初期配置　実行結果

同じプログラムで餌の配置を100回変え実験した結果32回の最適解を得ることが出来た。

4. プログラムの改良

ADF(自動的関数定義)の利用

ADF とは関数を自分自身で定義し効率的に利用する手法である。これは探索の過程で木が膨大になり探索効率が劣化するのを防ぐことができる。
ADF を人工蟻の探索問題に適用することによりプログラムの改良をはかって行こうと考える。

ADF プログラムの例

ADF を次のように定義したとする。
ADF＝(OR ARG0 (AND ARG0 ARG1))
本体が評価されるときは ADF が呼ばれるたびに引数を代入して関数の値を求める、実際に引数に値が入った場合には
(ADF D0 D1)＝(OR D0 (AND D0 D1)
このように展開し実行される。

(ADF D1 (NOR D2 D0))
　このようなプログラムがあったとすると
(OR NOR D2 D0)(AND (NOR D2 D0)D1))
となる。

このことから ADF を用いることでコードをかなり節約出来るのが分かる、ADF に関しては通常の関数と同じように遺伝的オペレータ(交叉、突然変異など)が適用される。

ADF を用いたプログラムの実行結果

ADF を用いたプログラムでも最適解を得られた
それぞれ 1000 回実験した結果

	最適解の得られた回数	最適解が得られるまでの世代数の平均
ADFなし	384回	78．4世代
ADFあり	359回	61．2世代

5. 結果からの考察

頑強性の実験から今回出来るプログラムにはある程度の頑強性があることが分かった、頑強性のあるプログラムを人の手で書くのは困難なので GP を使うことへの利点がよく現われた実験結果であったと言える。
　また ADF を用いた場合、最適解の得られた回数にはほとんど変化が見られなかった、これは ADF を用いても実際に使う関数自体には変化はなく、局所解に陥るときは同じように解が得られないプログラムになってしまうことが原因だと考えられる、最適解の得られる回数を増やすには交叉確率や突然変異確率を見直すか、または問題に適した交叉の親選択技法への変更などの方が重要であると考えられる。
　ADF を用いることで最適解が得られるまでの世代数の平均は向上した、つまり最適解が得られるのが早くなったと言える、さらに出来上がったプログラムも短くなった、関数を効率的に組み合わせることで良い解への収束が早まったことが考えられる。

6. 参考文献

「ニューロ・ファジィ・遺伝的アルゴリズム」 荻原将文(著)
「遺伝的プログラミング入門」 伊庭斉志(著)