ゲーム戦略獲得に関する自動プログラミングへの進化的アプローチ

ゲーム戦略獲得に関する自動プログラミングへの進化的アプローチ

山城 正* 山田 孝治* 遠藤 聡志*

Evolutionary Approach to Automatic Programming for Game-Strategy Acquisition

Tadashi YAMASHIRO* Koji YAMADA* Satoshi ENDO*

Abstract

In this paper, we discuss an automatic programming for game strategy acquisition using genetic programming (GP). This paper shows an implementation of MOO game. MOO is a guessing game. In this game, the codemaker picks a number consisting of three distinct decimal digits. Then the codebreaker guesses three distinct digits being scored on each guess. We show and evaluate the codebreaker program that is created automatically by the GP.

Key Words: Genetic Programming, game strategy, MOO, Hit & Blow

1. まえがき

近年，自然界に内在するシステムをモデルとする情報処理の可能性に関しての関心が高まっている．特に，自然界の生体の個体・集団が持つメカニズムは，ゲーム戦略の獲得などの情報処理メカニズムを検討するうえで，参考となる部分が多い．

このような生物学的知見を情報処理の観点から捉え直した研究は，人工生命として新しいパラダイムを提唱しつつある．

この人工生命における進化パラダイムの根幹をなす手法が遺伝的アルゴリズムである．遺伝的アルゴリズム・遺伝的プログラミング（Genetic Algorithm/Programing）は遺伝子の交叉・突然変異のメカニズムを参考にして，対象物の構造・属性を順次改善していくような計算メカニズムの総称のことである．遺伝的アルゴリズムは，生物の進化過程をシミュレートする工学的モデルであり，最適探索法として，多様かつ頑健な側面をもち，しかも並列処理との親和性が高いことから，大きな期待が寄せられている．特に適応的な知識処理の枠組みを作る進化的計算の研究の中で，ゲーム戦略獲得に関する方法論の開発は，最良の結果に結びつける思考プロセスをどのようにプログラムするかという点で重要な課題の一つである．

本研究では，進化的プログラミング手法の一つである遺伝的プログラミング（GP）の適用により，ゲーム戦略の獲得プログラムの自動生成を検討する．GPは，論理式や数式などの構造的表現を扱えるように，遺伝子型を木構造として遺伝的アルゴリズム（GA）を拡張したものであり，その木構造に交叉や突然変異といった遺伝子操作を加えることで，評価関数を満たすプログラムを自動生成する手法である．GPは学習や推論，システム同定などの様々な分野に応用され，その有効性が示されている．

受理：1998年5月25日
*工学部情報工学科
(Dept. of Information Engineering, Fac. of Eng.)

GPの特徴をまとめると，次のようになる．

1. GPはプログラムを進化させるものである．
2. GPによって，ロボットのプログラムや人工知能の問題解決・推論・学習のためのプログラムが進化的に（創発的に）自動生成される．
3. GPはGAの考え方に基づいている．その違いは，GAが主に最適化を目指すのに対し，GPは記号処理的なプログラムの生成を目的としている．
4. ニューラルネットワークは数値的な処理を行なうのに対し，GPは記号的なプログラムによる処理を実現する．

GPを適用する際のゲーム対象として，3桁の数当てゲーム（MOO, HitBlow）を取り上げる．このゲーム戦略の学習において，GPを実行することにより，戦略を獲得するための解答プログラムの自動生成を行なう．

2. 数当てゲーム (MOO)

ゲームの基本的な分類法は，ゲームに参加するプレーヤーの人数によるもので，通常2人ゲームと3人以上のゲームとに大別される．

2人ゲームは，ゲーム理論全体からみて，基本的な役割を果たしているため，本研究では，2人ゲームの一種である数当てゲーム (MOO) を取り上げ，研究を進める．

MOOでは，2人のプレーヤーがゲームを行なう．出題者と解答者に分かれ，前者は正解となる任意の数列を用意して隠しておき，後者がそれを当てる．後者の出す質問に，前者はヒントを出す．いかに質問する回数を少なくするかが，このゲームのポイントである．

2.1 ゲーム手順

このゲーム手順は，以下の様に進められる．

手順 *1* 出題者は0から9までの互いに異なる数字（リピートなしと呼ぶ）を使った3桁（以下ではこのような数をMOO数と呼ぶ）を正解として用意し，解答者から隠しておく．MOO数は $10 \times 9 \times 8 = 720$ ある．

手順 *2* 解答者は質問としてMOO数を1つ提示する．出

題者は質問と正解の数を比較し，同じ数字が同じ桁に出現する回数（**Bull** と呼ぶ），同じ数字が別の桁に出現する回数（**Cow** と呼ぶ）をヒントとして与える．2つの MOO 数から Bull と Cow の対を求める計算を以下では **MOO 積**と呼び，(*Bull, Cow*) という対で記述する．

手順 *3* 質問に対する MOO 積が (3, 0) のとき，ゲームは終了する．そうでなければ，手順 2 に戻る．

図 1に，この様子を示す．

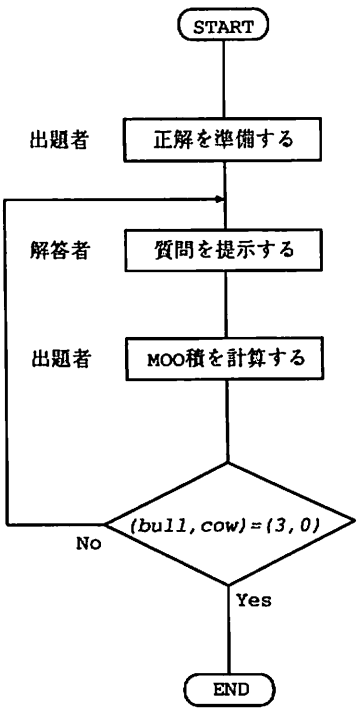

図 1. **MOO** ゲームのフローチャート

2.2 ゲームの進行例

ゲームの進行例を表 **1**に示す．上級者になると，例のように5回程度の質問で当てることが多いが，要領がわからないうちは何十回質問をしても当たらないこともある．それでいて，初心者でもビギナーズラックで1回目の質問で正解することもある点が，このゲームの面白さの1つとなっている．

3. 遺伝的プログラミング

遺伝的プログラミング（GP, Genetic Programming）は，**遺伝的アルゴリズム**（GA, Genetic Algorithm）の遺伝子（GTYPE）を拡張し，構造的な表現を扱えるようにしたもので，**John Koza** によって確立された．ここでの**構造的表現**とは，グラフ構造や木構造のことをいう．まずはじめに，なぜこのような表現を扱う必要があるのかを説明する．

例として，エキスパートシステムや学習などの人工知能の問題に，GA を応用することを考える．まず，人工知能では記号的な構造表現，特にグラフ構造や木構造がしばしば登場することに注目する．これは**知識表現**としてグラフ構造が便利であるからであろう．例えば，図 2と図 3に人工知能のシステムでしばしば用いられる知識表現を示す．図 2はフレーム（Frame）と呼ばれる概念を表現する言語によって，ジャックと変人の関係を記述したものである [Winston92, p.188]．ここで，ako は集合間の包含関係（a kind of），isa は要素が集合に帰属すること（is a）を述べている．これを用いて人工知能ではさまざまな推論，学習を試みる．図 3は**機械翻訳**で用いられる導出木である [Barr81, p.308]．" the boy ate the apples " という文章を句構造文法で解釈した結果が記述されている．さらに，図 4は数式の木構造である．

正解:723		
回数	質問	MOO 積
1	412	(0, 1)
2	526	(1, 0)
3	328	(1, 1)
4	923	(2, 0)
5	723	(3, 0)

表 1　ゲームの進行例

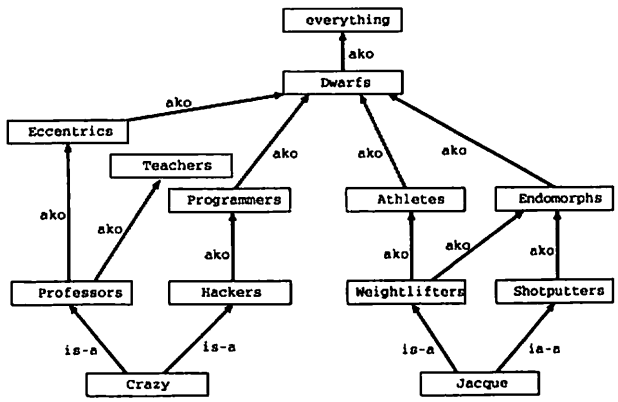

図 2. フレームによる概念木

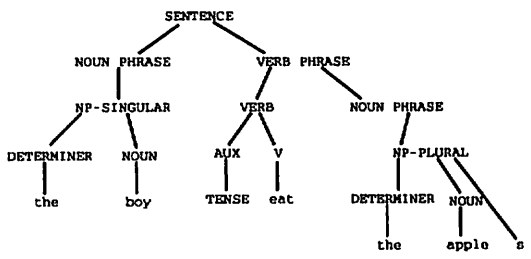

図 3. 構文解析の木

したがって，複雑な数式や概念，関係などを木構造で表現できることがわかる．このことから，グラフ構造（特に木構造）を扱えるように GA の手法を拡張することは意義のあることであり，GA の適用範囲の拡大につながると思

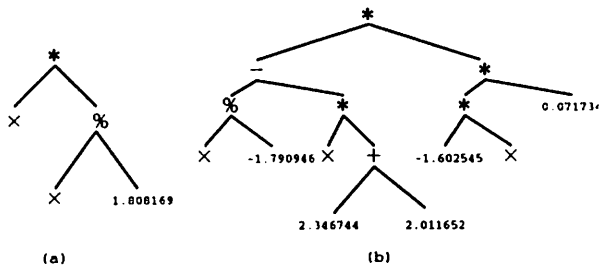

図 4. 数式の木構造

われる．

GA を構造的表現に拡張した枠組が GP であり，従来の GA の欠点を次のように補うことを試みるものである．

1. 探索のための的確な部分構造の把握
2. 問題の表現形式に基づいた効果的な探索の実現
3. より高次の知識の適応的な学習システムの構築

3.1 GP の仕組み

GP では木と呼ばれる構造表現を扱う．木はサイクルを持たないグラフのことであり，図5のような構造をいう．

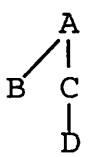

図 5. 木構造

木構造はかっこつきの表現で記述でき，例えば図5の木は，

(A (B)
(C (D)))

もしくは簡略化して，

(A B
(C D))

となる．この表記法を（LISP の）S 式表現という．以下では木構造と S 式を同一視する．なお，このような木構造に関して，以下の用語を用いる．

- ノード：記号 A，B，C，D のこと
- 根 (ルート)：A
- 終端ノード：B，D（終端記号，葉ともいう）
- 非終端ノード：A，C（非終端記号，S 式の関数記号ともいう）
- 子供：A にとっての子供は B，C（関数 A の引数ともいう）
- 親：C にとっての親は A

木に対する GA のオペレータとして，以下を導入する．これらはビット列を対象とする従来の GA オペレータの自然な拡張である．

Gmutation ノードのラベルの変更
Ginversion 兄弟の並べ換え
Gcrossover 部分木の取り換え

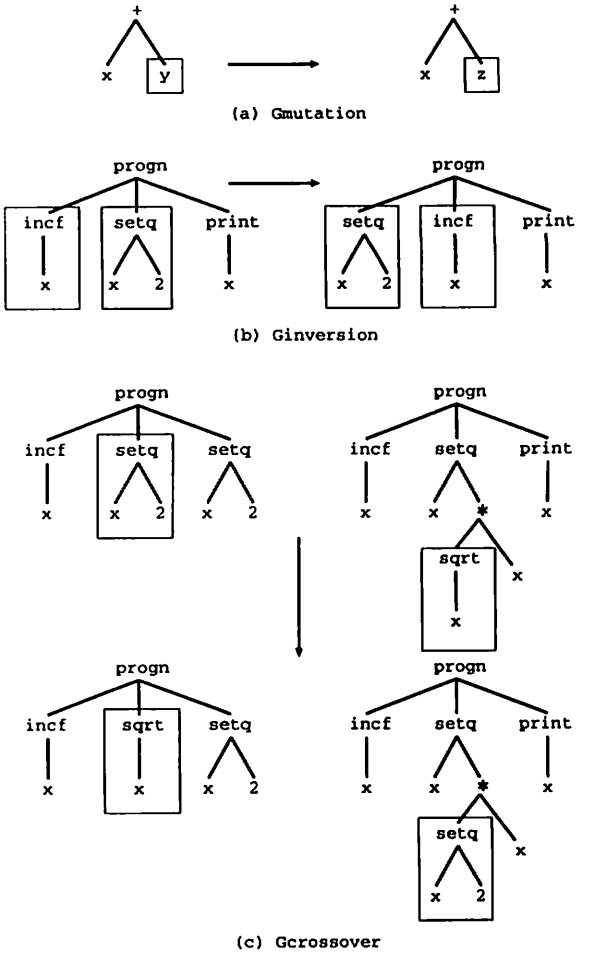

図 6. LISP の S 式への適用例

これらのオペレータを木構造に適用した例を図6に示す．この適用を S 式で記述すると次のようになる．ただし，オペレータの適用部分には下線を付した．

- **Gmutation**

  $(+\ x\ y)$
  ↓
  $(+\ x\ \underline{z})$

- **Ginversion**

  $(progn\ (incf\ x)\ (setq\ x\ 2)\ (print\ x))$
  ↓
  $(progn\ \underline{(setq\ x\ 2)\ (incf\ x)}\ (print\ x))$

- **Gcrossover**

  $(progn\ (incf\ x)\ (setq\ x\ 2)\ (setq\ y\ x))$
  $(progn\ (decf\ x)\ (setq\ x\ (*\ (sqrt\ x)\ x)\ (print\ x))$
  ↓
  $(progn\ (incf\ x)\ \underline{(sqrt\ x)}\ (setq\ y\ x))$
  $(progn\ (decf\ x)\ (setq\ x\ (*\ \underline{(setq\ x\ 2)}\ x)\ (print\ x))$

Gcrossover についてだが，GA の交叉との大きな違いは，同じ遺伝子を持った個体同士を交叉させても，同じ遺伝子を持った子供が必ずしも生まれないという点である．GA の場合，図7となり，遺伝子構造が変わらないのに対し，GP における交叉では，図8に示す通り，交叉点によっ

て生成される個体の遺伝子が変化してしまうのである．

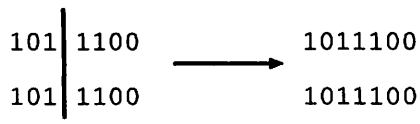

図 7. GA の交叉の例

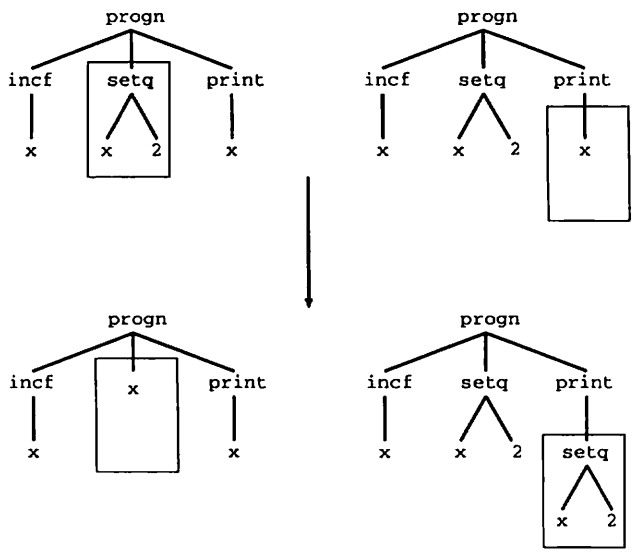

図 8. GP の交叉の例

また，**Gmutation** については次の種類がある（図 9）．

1. 終端記号から非終端記号への**突然変異**（図 9 (a)）
   新しい部分木の生成を伴う
2. 終端ノードから終端ノードへの突然変異（図 9 (b)）
   ノードラベルの付け換えのみ
3. 非終端ノードから終端ノードへの突然変異（図 9 (c)）
   部分木の削除を伴う
4. 非終端ノードから非終端ノードへの突然変異
   Case1 新しい非終端ノードと，古い非終端ノードの子の数が同じ場合（図 9 (d)）
   ノードラベルの付け換えのみ
   Case2 新しい非終端ノードと，古い非終端ノードの子の数が異なる場合（図 9 (e)）
   部分木の生成・削除を伴う

オペレータの適用の割合は確率的に制御される．

以上の準備のもとに GP のアルゴリズムは次の様になる．

Step1 ランダムに木構造 **GTYPE** $g_t(i)$ を構成する．

Step2 各 GTYPE $g_t(i)$ の**表現型 PTYPE** $p_t(i)$ に対して**適合度** $f(i)$ を求める．

Step3 適合度の大きな GTYPE に対して一定数のペアを取り出す．

Step4 取り出したペアに対して **Gcrossover** を適用し，適合度の小さな GTYPE と置き換える．

Step5 各 GTYPE に関して，ランダムに **Ginversion**, **Gmutation** を適用する．

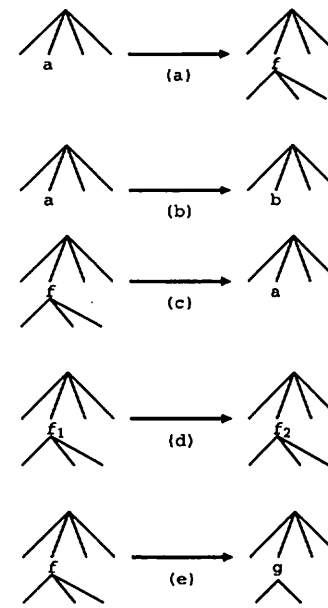

図 9. Gmutation オペレータ

Step6 以上によって求められた新しい GTYPE を，次の世代の $g_{t+1}(i)$ として，**Step2** へ戻る．

ただし，適合度は大きいものほど良いとしている．このアルゴリズムは，オペレータの違いを除いて，GA のアルゴリズムと同一である．したがって，GP では GA の知見の多くをそのまま用いることができる．

GP では次の 5 つの基本要素を設計することで，様々な応用問題への適用が可能になる．

1. **非終端記号**（以下 F で表す）
   非終端ノードで使う記号．LISP の S 式での関数．
2. **終端記号**（以下 T で表す）
   終端ノード（葉）で使う記号．LISP の S 式でのアトム．
3. **適合度**
4. **パラメータ**
   交叉，突然変異の起こる確率，集団サイズなど．
5. **終了条件**

Step1 のランダムな**木構造の生成**は，T と F が与えられたとき，次の手続き SUBTREE を呼ぶことでなされる．

手続き SUBTREE :

1. $T \cup F$ から 1 つのノード x をランダムに取り出す．
2. $x \in T$ なら x を返して終わり．
3. $x \in F$ なら x の引数の数を n とする．そして，
   SUBTREE を call して，その結果を a_1 とする．
   SUBTREE を call して，その結果を a_2 とする．
   ……
   SUBTREE を call して，その結果を a_n とする．
   最後に $(x\ a_1\ a_2 \cdots a_n)$ という部分木を返して終わり．

したがって，木構造は再帰的に生成されることがわかる．ただし，このままでは非常に深い木を得ることがあるため，木の生成を適切に制御する必要がある．

3.2 GP の応用例と問題点

GP は様々な分野に応用され，その有効性が確かめられている．GP の適用範囲は，AI の問題解決から，ロボット，分子生物学などの実際的な問題まで多岐に渡っている．

GP では木構造に交叉と突然変異を適用することで木を変形し，LISP（S 式の）プログラムや概念木などを探索する．この方式の効果や有効性についてはまだ不明な点も多い．GP 研究の現状での最大の問題点は計算量である．例えば，Koza らの実験は集団数が 4,000～16,000 となっている．各世代で LISP の S 式の評価が集団数分必要なことを考えると，これは通常の計算機パワーの限界に近い．GP の実験には実行が数日かかるというのも珍しくない．Koza によれば，集団数の多さはグラフ構造の多様性（population diversity）を保持するためであり，その結果，実行に要する世代数は比較的少なくなっている（10～20 程度）．これは通常の GA での実行方法（集団数 50～100 前後で世代を多く重ねる方式）とは対照的である．言い替えると，GP の実行では各世代での木構造の評価の計算量が莫大になり，実行が非常に遅くなる．

また，GP を適用する際に最も重要なのはノードの表現の設計（つまり終端記号と非終端記号を何にするか）である．これによって，交叉や突然変異によって木の意味が劇的に変化する．この現象を意味破壊（semantic disruption）と呼ぶが，GP オペレータにより意味破壊が頻繁におこると探索が安定しないことがある．

4. GP の設定

本研究の目的は，GP を MOO ゲームに適用させ，進化学習を行なうことで戦略獲得を導く解答プログラムの自動生成である．

この章では，GP を MOO ゲームに適用する際に必要な設定について説明して，実験を行なう．はじめに，MOO ゲームの平均質問回数を求める解答プログラムを設定する．そして，進化学習対象プログラムを抽出して，このプログラムを表現できるノード（非終端記号，終端記号）を設計することが本研究では最も重要である．これらの準備を基に 3.1 節で説明した 5 つの基本要素を設計することで，GP の適用が可能となる．

4.1 進化学習対象プログラム

MOO の解答プログラムとして，図 10のようなアルゴリズムを設定する．

このアルゴリズムは，正解を求める一般的な解答プログラムである．つまり，

1. すべての MOO 数の中から，それまでの質問の応答に当てはまる正解の候補の集合（以下では解のグループと呼ぶ）を求める．
2. 解のグループの中から，次の質問をランダムに選ぶ．

という手順でゲームを進行している．

ここで，質問と正解との MOO 積を $P1$，質問と正解の候補との MOO 積を $P2$ と置くと，$STEP4$～$STEP5$ の部分は，$P1$ と $P2$ から正解の候補を選択することによって，解のグループを絞り込むということができる．つまり，この部分は解答プログラムの中でもっとも重要だといえる．言い替えれば，この処理が戦略獲得につながっており，戦略の評価は平均質問回数によって行なわれる．よって GP

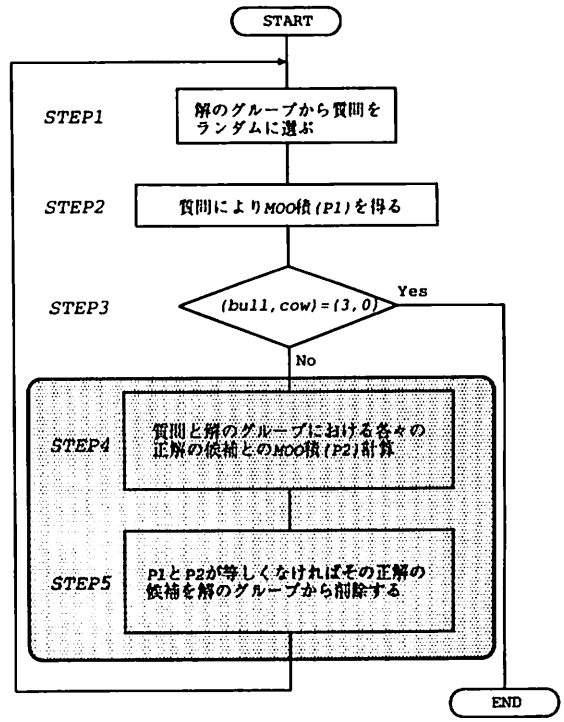

図 10. 解答プログラム

により進化学習させる対象は，解のグループの絞り込みを行なうプログラムとする．

GP の設定として，はじめに単純な処理を行なうノード（関数）をいくつか設計する．これらは非終端記号と終端記号であり，この組み合わせによってプログラム処理が行なわれる．この場合，質問と $P1$ のみの情報から，$P2$ を計算して正解の候補の選択を行なうということを学習しなければならない（図 11）．

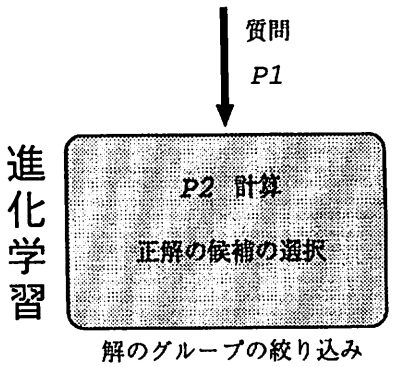

図 11. 進化学習対象プログラム

そこで，戦略を獲得するために，このプログラムの自動生成を目的とする．

4.2 基本要素設定

以下に，進化学習対象プログラムを表現するノード設計（非終端記号，終端記号）を含めた基本要素設定を示す．

1. 非終端記号（F）

- $F = \{IFMOO11, IFMOO12, IFMOO13, IFMOO23, IFMOO23, IFMOO22, IFMOO33\}$
  これらは2つの引数をとる関数である．質問と正解の候補の各桁どうしを比較して，等しければ第1引数を実行する．それ以外は第2引数を実行する．例として，$IFMOO13$ を挙げる（図12）．

IFMOO13（引数1、引数2）

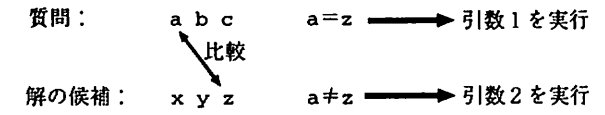

図12. *IFMOO* 13

- $F = \{IFBULL, IFCOW\}$
  これらは3つの引数をもつ関数である．$P1$ と $P2$ における *bull* を，$P1$ と $P2$ における *cow* を各々比較する．前者が大きければ第1引数を，等しければ第2引数を，そうでないときは第3引数を実行する．例として，$IFBULL$ を挙げる（図13）．

IFBULL（引数1、引数2、引数3）

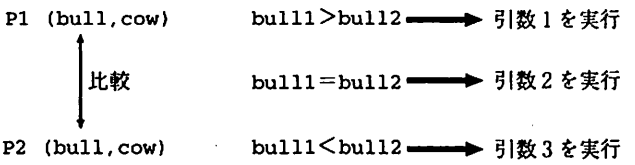

図13. *IFBULL*

- $F = \{PROG2, PROG3\}$
  $PROG2$ は2引数，$PROG3$ は3引数をとる関数である．前者は第1, 2引数を順に，後者は第1, 2, 3引数を順に実行していき，最後の引数を実行した値を返す．例として，$PROG3$ を挙げる（図14）．

PROG3（引数1、引数2、引数3）

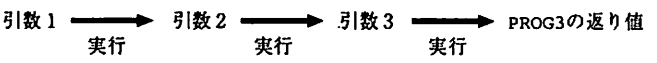

図14. *PROG3*

また，$PROG$ などの関数を用いて適切に非終端記号を設計することでGPの生成するプログラムに冗長性が導入できる．ここでの冗長性とは，ある状況で有効でなくても，別の状況において有効であるような部分構造の存在を意味する．冗長性は生成されたプログラムが未知のデータに対しても，うまく振る舞うことを保証する．

2. 終端記号（T）
   - $T = \{BULL\_pp, COW\_pp\}$
     これらは引数をもたない関数である．$P2$ における *bull* と *cow* を各々インクリメントする．
   - $T = REMOVE\_LIST$
     これは引数をとらない関数であり，正解の候補を解のグループから削除する．
   - $T = NOP$
     これは引数をとらない関数でなにも実行しない．
3. 適合度
   まず，平均質問回数を求め，これに，個体によって計算された $P2$ の誤差を加算した値を適合度とする．式を以下に示す．

$$fitness = question\_avg + (|e\_b| + |e\_c|)/10.0 \quad (1)$$

ここで，$question\_avg$ は平均質問回数，e_b , e_c は各々，$P2$ における *bull*, *cow* の誤差である．計算される $P2$ の誤差を，評価関数に含めることで，$P2$ 計算の学習を行なわせる．これより，解のグループの絞り込み操作が可能となる．
   なお，$fitness$ の値が小さいものほど適合度は良いものとする．
4. パラメータ
   - seed = 11287
     （乱数のシード）
   - population_size = 100
     （集団数）
   - max_depth_for_new_trees = 6
     （初期に生成される木の最大深さ）
   - max_depth_after_crossover = 17
     （交叉で生成される木の最大深さ）
   - max_mutant_depth = 4
     （突然変異で生成される木の最大深さ）
   - grow_method = RAMPED
     （木の生成規則：集団内の個体ごとに木の成長方式を変える）
   - selection_method = FITNESSPROP
     （選択方式：ルーレット方式）
   - crossover_func_pt_fraction = 0.2
     （非終端ノードでの交叉の確率）
   - crossover_any_pt_fraction = 0.2
     （非終端および終端ノードでの交叉の確率）
   - fitness_prop_repro_fraction = 0.1
     （コピーのみの確率）
   - parsimony_factor = 0.00000
     （適合度の変換係数）
   - 世代数は200，平均質問回数を求めるためのゲーム回数は100回とする．
5. 終了条件
   設定した世代数を終了したときとする．

以上の基本要素設定を基に，GPを実行する．

4.3 実験結果

実験結果を図15に示す．このグラフは，集団における適合度の最良値，平均値の推移を示している．縦軸を適合度，横軸を世代数としている．世代が進むに連れ，適合度が減少，つまり良くなってきているのが確認できる．

最良個体の平均質問回数の推移を，図16に示す．縦軸を平均質問回数，横軸を世代数としている．平均質問回数は，0世代目の約96回から，200世代目の約26回へとかなり減少していることが確認できる．

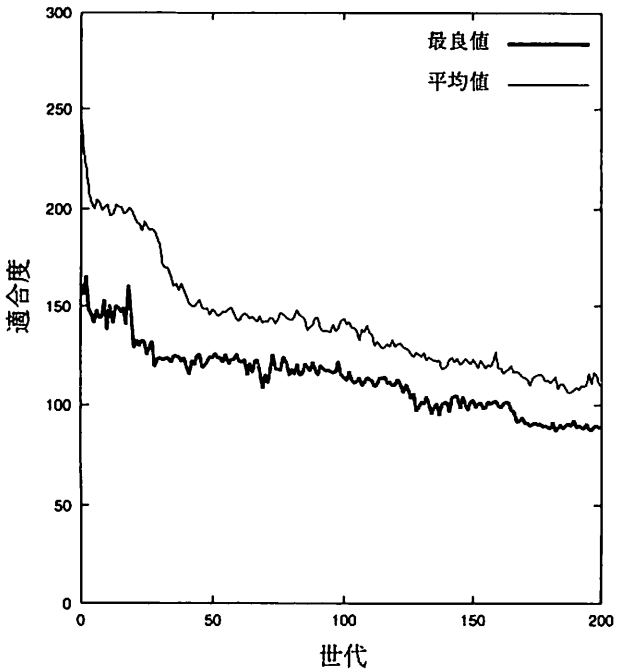

図 15. 適合度の推移

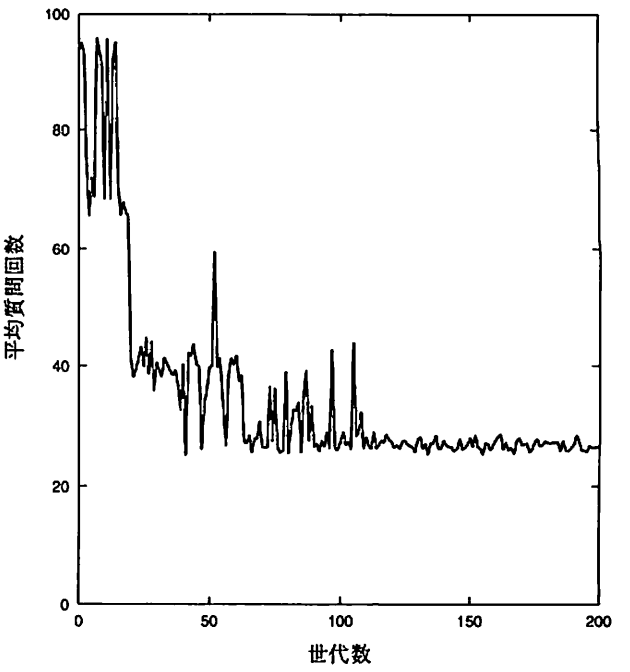

図 16. 最良個体の平均質問回数

182 世代目で得られた最良個体のプログラムを，図 17に示す．このプログラムは LISP 形式で記述されており，適合度は約 87.13，平均質問回数は 27.18 回である．

4.4 考察

実験結果より，次のことが確認できる．

1. 適合度は世代が進むに連れ向上していることから，集団の段階的な進化学習．
2. 平均質問回数の減少により，解のグループの絞り込み操作の実行．

よって，自動生成プログラムにより戦略を獲得するという段階まで進化することができたといえる．

しかし，本実験における基本要素設定では，ここまでの進化が限界であった．原因は，解のグループの絞り込みにおける $P2$ の計算が正確に行なわれていないことである．181 世代目で得られた最良個体の適合度と平均質問回数を比較すると，約 65 の開きがある．つまり，適合度は平均質問回数と個体によって計算された $P2$ の誤差を加算したものであるから，この 65 という値は $P2$ 計算の誤差ということになる．よって，この誤差をなくすことにより，的確な解のグループの絞り込みが行なわれ，最終的には平均質問回数を約 5 回まで減少させることが可能である．

これを実現するには，ノード設計（非終端記号，終端記号）の改善，プログラムの長さを考慮した評価関数の設定を考える必要がある．ノード設計改善策の一つとしては，*PROG* などの関数を用いて適切に非終端記号を設計することで GP の生成するプログラムに冗長性を導入することが挙げられる．

また本実験においては GP 実行の際，計算量の問題から，パラメータを変更して比較実験を行なうことができなかった．例えば，木の最大深さのパラメータを変更することで，

```
Generation 182 Population 0   Avg Std Fitness: 107.736982
Best-of-gen fitness: 87.129189
Best-of-gen tree:

(IFBULL (IFMOO23 (IFCOW NOP (PROG3 (IFMOO11 (IFMOO12 (IFMOO33 REMOVE_LIST BUL
(IFMOO13 (PROG3 NOP NOP (IFMOO23 (IFMOO11 NOP BULL_pp) REMOVE_LIST)) BULL_pp)
(IFBULL (IFCOW BULL_pp COW_pp (IFMOO13 (IFMOO12 NOP NOP) (IFCOW REMOVE_LIST
REMOVE_LIST NOP))(IFMOO33 (IFMOO33 BULL_pp (IFMOO23 COW_pp COW_pp)) COW_pp) B
(IFMOO12 (IFMOO11 REMOVE_LIST (IFMOO12 NOP NOP)) REMOVE_LIST)) NOP) (IFMOO11
(IFMOO13 REMOVE_LIST (IFCOW REMOVE_LIST (PROG2 (PROG3 (IFMOO23 (PROG2 BULL_pp
(IFMOO12 BULL_pp (IFMOO33 BULL_pp COW_pp))(IFMOO33 COW_pp REMOVE_LIST)(IFMOO
REMOVE_LIST NOP)) NOP (IFMOO33 (IFMOO22 BULL_pp REMOVE_LIST)(IFMOO22 (IFBULL
(IFMOO23 BULL_pp REMOVE_LIST) BULL_pp (PROG2 COW_pp BULL_pp))
(IFCOW REMOVE_LIST(IFMOO12 COW_pp BULL_pp)))(IFMOO11 REMOVE_LIST BULL_pp) (IF
COW_pp COW_pp NOP))))) NOP)) (IFBULL (IFCOW BULL_pp COW_pp(IFMOO13 REMOVE_LIST
REMOVE_LIST REMOVE_LIST NOP))) (IFMOO33 (IFMOO33 BULL_pp(IFMOO23 COW_pp COW_p
COW_pp) BULL_pp)) (IFMOO22 (IFMOO13 (IFMOO12 (IFMOO12 (IFBULL(IFCOW NOP BULL_
COW_pp) (IFMOO13 NOP COW_pp) (IFMOO22 COW_pp (IFMOO12 NOP NOP)))(IFMOO22 (IFM
NOP (IFMOO13 REMOVE_LIST BULL_pp)) (PROG3 BULL_pp REMOVE_LIST BULL_pp))))(IFCC
(PROG3 (IFMOO13 (IFMOO33 (IFMOO22 BULL_pp BULL_pp) COW_pp) COW_pp) (PROG3 COW
(IFMOO23 (IFMOO13 (IFMOO33 COW_pp (IFMOO13 (IFCOW (IFMOO12 COW_pp BULL_pp) BU
(IFMOO13 NOP(IFMOO12 NOP REMOVE_LIST))) COW_pp)) NOP)(PROG2 REMOVE_LIST REMOV
 REMOVE_LIST) REMOVE_LIST) COW_pp)) REMOVE_LIST)
(IFMOO12 NOP (IFMOO13 REMOVE_LIST BULL_pp))))

AVERAGE COUNT2 = 27.180000
```

図 17. 最良個体のプログラム

良い結果が得られるということが考えられる．

5. むすび

本論文では，ゲームの戦略獲得に関して，3 桁の数当てゲーム（MOO）を対象に遺伝的プログラミング（GP）を適用をすることにより，戦略獲得のためのプログラムの自動生成を行なった．

このことにより，自動プログラム生成の応用例として，知識工学の分野における知識抽出プログラムの生成が可能と思われる．

謝辞

本研究の一部は，財団法人テレコム先端技術研究支援セン

ターの支援により実施した.

文 献

[1] 吉永 良正："「複雑系」とは何か", 講談社現代新書 (1997).
[2] 田中哲朗：数当てゲーム, 松原・竹内 編「bit 別冊 ゲームプログラミング第3章」, 共立出版 (1997).
[3] 伊庭 斉志："遺伝的アルゴリズムの基礎 -GA の謎を解く-", オーム社 (1994).
[4] 伊庭 斉志："遺伝的プログラミング", 東京電機大学出版局 (1996).
[5] 根路銘 もえ子："競合共進化に基づくゲーム戦略の学習に関する研究", 琉球大学卒業論文 (1996).
[6] 山城正："進化的プログラミング手法に基づく戦略獲得法", 98 年度琉球大学情報工学科卒業論文.