論　文

Genetic Network Programming とそのマルチエージェントシステムへの応用

学生員　片桐　広伸*　正　員　平澤　宏太郎**
正　員　胡　　敬炉*　正　員　村田　純一*

Genetic Network Programming and Its Application to the Multiagent System

Katagiri Hironobu*, Student Member, Hirasawa Kotaro**, Member, Hu Jinglu*, Member, Murata Junichi*, Member

Recently many studies have been made on automatic design of complex systems by using the optimization technique such as Genetic Algorithms (GA), Evolution Strategy (ES), Evolutionary Programming (EP) and Genetic Programming (GP). In this paper, a new method named Genetic Network Programming (GNP) which describes the behavior sequences of agents as a network is proposed in order to acquire intelligent behavior sequences. GNP includes judgement nodes and processing nodes in the network. The proposed system can evolve itself by an evolutionary method using mutation and crossover.

キーワード：GA, GP, GNP, 進化論的計算, マルチエージェント, タイルワールド
Keywords: GA, GP, GNP, Evolutionary Computation, Multiagent, Tile World

1. はじめに

従来より，人工知能の制御への応用や移動ロボットのプラニングに関する様々な研究がなされている(1)(2)。このような研究は，ある環境下に存在するエージェントが，与えられた目的を達成するために，その環境に応じてどのような行動をとるべきかを決定する規則，即ちエージェントの行動系列を生成することに関する。またそれは，エージェントを生物とみなしたならば，いわゆるその生物の'脳'を設計することであるとも言える。

この'脳'の実現，即ちエージェントの行動系列の作成には設計者が関与しても良い。事実，大抵の機械は設計者により設計され，要求されるタスクを処理している。しかし，エージェントの目的や環境がより複雑になるにつれ，そのような行動系列の設計が困難を極めることは容易に想像できる。さらに，特定の設計者によって設計された行動規則に基づいた行動では，エージェントの能力が設計者のスキルに大きく依存するといった問題や，環境変化が起こると与えられた行動系列では対処できないケースが多いといった問題点が挙げられる。

このような事から，自律性と創造性に富んだエージェントの行動系列は，設計者によって与えられるものよりはむしろ進化論的手法により，設計者なしに自動的に獲得されることが望ましい。

こうした背景から，特に複雑系において，マルチエージェントを含むエージェントの行動系列の自動設計に関する多数の研究がなされてきた(2)。それらは，仮想世界において与えられたタスクを処理するためのエージェントの行動系列を設計するものから(3)，蟻や魚に類似した人工生命を作り出そうという試み(4)，さらには実環境において実機による単純な目的をもったエージェントのアクチュエータの設計(5)など多岐に渡る。そのようなエージェントの行動系列を実現するモデルが，Genetic Algorithm, Evolution Strategy, Evolutionary Programming, Genetic Programming といった進化的手法を用いて多数提案されている(3)(5)(6)。

これまでに報告されてきたGPやその他の手法では，エージェントが現在置かれている情報を全て用い，'今，なにをするべきか'という点を重視した行動系列の生成を試みている例が多い。これに対し，本論文では行動を一連の流れと

*九州大学
〒812-8581 福岡県福岡市東区箱崎 6-10-1 九州大学大学院システム情報科学研究院電気電子システム工学専攻
Kyushu University, Department of Electrical and Electronic Systems Engineering, Kyushu University.
6-10-1, Hakozaki, Higashi-ku, Fukuoka, 812-8581, Japan
**早稲田大学
〒808-0135 福岡県北九州市若松区ひびきの 2-2 早稲田大学大学院情報生産システム研究科
Waseda University, Graduate School of Information, Production, and Systems, Waseda University.
2-2, Hibikino, Wakamatsu-ku, Kitakyushu, Fukuoka, 808-0135, Japan

して捉え，これまでにどのような判断/行動を行ってきたかを重視し，環境に関する情報も現在必要と思われるもののみを必要に応じて用いるといった現実の生物の'脳'が行っている機能により近い手法でエージェントの行動系列のプラニングを試みる。換言すると，非マルコフプロセス環境下での行動系列のプラニングを考察する。

具体的には，自律的，創発的行動を生み出す行動系列の生成を目指して，簡単な機能を持った複数のノード素子をネットワーク状に接続し，ネットワーク内の制御の遷移規則により行動系列を表現するモデル "Genetic Network Programming (GNP)[8]~[11]" を提案する。GNPの設計手法のポイントは，ノード間の接続を進化論的に変更することによって，より優れた行動系列を生成し，意図する機能を具備するGNPプログラムを得ることにある。これまでのGNPの研究例として，GNPとGPとの比較研究[9][10]，GNPのオンライン学習[11]が挙げられるが，本論文ではより基本的なレベルでのGNPの検証を行っている。具体的には，GNPの要であるノード関数の種類が適合度曲線へ及ぼす影響，ノード数が進化能力へ及ぼす影響，さらには得られたGNPプログラムの汎化能力の検証を行う。

提案手法の上記の検証は，動的環境のテストベッドとして知られるタイルワールド環境を使用して行った。なお，GNPのオンライン学習[11]とは学習方式そのものが異なることに注意して頂きたい。

本論文の構成は次のとおりである。第2章で提案するGNPの基本構成，GNPの進化方法について述べた後，進化型GNPとオンライン学習型GNPの差異について説明を行う。第3章は本論文におけるGNPの適用例の説明，第4章はシミュレーション結果とその考察，第5章はまとめである。

2. Genetic Network Programming (GNP)

Genetic Network Programming (GNP) は知的エージェントの行動系列のモデル化，およびその自動獲得を目的とする。類似研究として，GPやEP等を利用した幾つかの報告があるが，本提案手法ではより柔軟な行動系列の表現が可能であると考える。以下に，従来手法との簡単な比較を交え，GNPの説明を行う。

基本的にGNPはGA及びGPの拡張である。GPには，獲得した知的情報を木構造によって保存し，終端ノードからの入力（環境情報）を予め用意された非終端ノード内の演算子（処理関数）により計算し，根ノードにおいて適切な値を得ることによって行動系列を得るものや[7]，根ノードから始まり，各非終端ノードにおいて，'if-then ルール'による判定を行い，最後に終端ノードに記載された処理を実行する，処理終了後は再び根ノードに戻るという形で行動系列を表現するもの[4]等がある。即ち，GPでは各ノードを単純な行動を処理する最小単位と考え，それらを木構造状に接続することによりエージェントの行動系列を表現する。一方GNPは，GPの後者の研究例と同様に，各ノードをエージェントの判断，行動を行う最小単位と考えるが，それらをネットワーク状に接続することによりエージェントの行動系列を表現する点に特徴がある。この構造の違いから，GNPでは各行動終了後にGPのように必ず根ノードに戻るといったことはせず，次の判断/行動は常に以前の判断/行動の影響を受けることになる。

従って，従来の手法では，行動する直前の環境情報によりエージェントの行動が決定されるが，GNPでは，行動する直前の環境情報に加えてエージェントの過去の行動にも依存して行動が決定される。なぜならばノードをネットワーク状に接続することにより，ネットワークがエージェントの過去の行動を記憶する一種のメモリ機能を果たすからである。

次に，EPを用いた研究例との比較を行う。EPは，近年になって最適化問題に適用された研究事例が多いが[12]，古くはオートマトンやグラフ構造の生成に適用されてきた[13]~[15]。これらは，反復囚人のジレンマゲームや物体認識システムなどに適用されており，基本的には状態，もしくは単純な処理を最小単位とするノードを用意し，それらの接続をEPを用いて変更することによりシステム（有限状態オートマトン:FSA）の構造の進化を試みるものである。一般にFSAでは入力の種類が増加するとそのネットワーク構造も非常に複雑かつ大規模になる。なぜなら，FSAはある状態における全ての入力に対する応答を用意しなければならないからである。本論文中で使用しているシミュレーション環境のように，種々の状況判断を要する環境においては，状況判断の分岐が爆発的に多くなりノード数や接続数もそれに比例して多くなる。また，環境内の全ての情報を得ることができない場合，マルコフ性を前提としているFSAでは適切な構造の獲得が困難になる場合がある。このため，従来型のFSAをベースとした構成は不向きである。一方，GNPでは問題特有の状況に依存した判定ノードと処理ノードを構造的に分離し，これらの構成と接続を自在に設定できるようにすることで，より問題に依存した適切なアーキテクチャの構成を可能にしている。換言すると非マルコフプロセス環境下でも適用できる進化論的計算手法を提案している。

〈2·1〉 GNPの基本構成　前節までは人の思考になぞらえて，判断·行動という言葉を用いてきたが，GNPにおいては判定·処理と置き換えて話を進めることとする。

GNPプログラムの基本構成を図1に示す。ここで，Sは初期起動ノード（start node）を示し，GNPプログラムはこのノードから起動する。初期起動ノードを複数個用意することにより，並列処理（'脳'でいうならば同時に別のことを思考する）の実現も可能であるが，本論文中のシミュレーションでは簡単化のため，初期起動ノードは1つに設定してある。GNPプログラムによる行動はすべてノードにおいて行われる。ノードには判定ノードと処理ノードの2種類が存在し，それぞれの各ノードに判定J_{n_j}と処理P_{n_p}が設定されている。J_i $(i=1,\cdots,n_j)$は判定ノード用のライブラリ

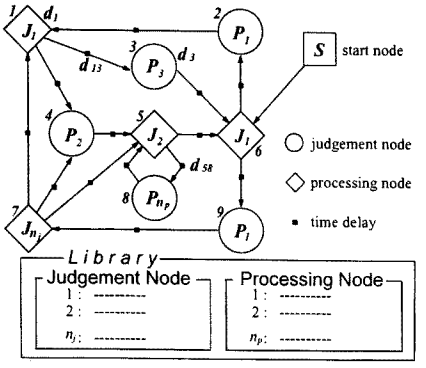


図1 GNPの構造
Fig. 1. Structure of GNP.

に記載されているi番目の判定を表し，P_i $(i=1,\cdots,n_p)$は処理ノード用のライブラリに記載されているi番目の処理を表す。ここで，ライブラリには設計者が用意したエージェントの最小の行動単位が記載されており，内容に重複はないものとする。d_iはノードiの処理に要する処理時間を，d_{ij}はノードiからノードjに移行する際の遅れ時間を示す。各処理に必要な時間を与えることによって，よりきめ細かなGNPの構造を構築することができる。また，ノード間の遷移に必要な遅れ時間を設定することによって，より現実味を帯びた判定/処理が可能となる。ここで，GNP内部閾値を設定し，ネットワーク上の制御遷移にともなう遅れ時間d_i, d_{ij}の累積値がGNP内部閾値を超えるまでを1ステップと定義する。1ステップの行動修了後，遅れ時間の累積値は0にリセットされる。複数のエージェントが存在する環境では，エージェントは1ステップ毎に行動権を獲得するものとしている。本論文のシミュレーションでは簡単化のため，ノード間の遅れ時間を0，判定時間を1に，処理時間を5，GNP内部閾値を5に設定した。すなわち，1つの処理を行った際，または5つの判定を行った際にエージェントは1ステップの行動を終えるものとしている。本論文ではこれらのパラメータの学習は行わないものとした。

〈2・2〉 **GNPの進化方法**　GNPプログラムの進化は，ノード間の接続のみを進化させることによって実現する。すなわち，ノード間の接続を進化的に変更することによって適切な構造を得ようというものである。本論文のシミュレーションでは，次のルールに基づき進化を行うこととした。

- ノード関数の種類毎に同数のノードを配置する。
- 初期世代におけるノード間接続をランダムに決定する。
- 進化対象はノード間の接続のみである。処理時間，遅れ時間に関して進化させることも可能であるが，本論文ではそれを行わない。

GNPプログラムの進化のためのオペレータとして，最も基本的な交叉と突然変異の2種を用いている。交叉は「2親個体内のノード間接続を交叉確率P_cで確率的に交換する」オペレータであり，一様交叉を採用している。また，突然変異は「ノード間接続の一部を突然変異率P_mで確率的に変更する」オペレータである。GNPプログラムの遺伝子表現，遺伝的オペレータの具体的説明に関しては文献[9]~[11]を参照されたい。

〈2・3〉 **進化型GNPとオンライン学習型GNP**　GNPはその名前の通り，進化論的手法をベースとしたプログラム合成を行う[8]~[10]。一方，オンライン学習型GNP[11]では強化学習でよく知られる利益共有法（Profit Sharing）による接続の更新を行っている。進化型GNPでは交叉，突然変異といった遺伝的オペレータによる部分的または大域的なプログラムの変更を行い，「全行動終了後」に得られる適合度を基準に集団内の個体の選択・淘汰を行う。交叉は高い適合度を持つ個体同士の部分プログラムを互いに交換することにより，さらに高い適合度を持つ個体を生成することを期待し，突然変異は高い適合度を持つ個体の一部をランダムに変更することにより高い適合度を持つ個体の生成を期待する。これに対しオンライン学習型GNPでは，基本的に1個の個体が学習を行い，ネットワーク上の制御の遷移を繰り返す過程において累積される環境からの報酬（負の報酬も含む）を基準に接続をランダムに変更する。接続の変更は，タスク実行中を通じて「ノード関数実行直後」に環境から受け取る累積報酬が閾値を下回った際に行われ，更新される箇所は累積報酬が閾値を下回った接続周辺で行われる。

上述をまとめると，両者の違いは（1）学習個体数，(2）接続変更のタイミング，(3）接続変更箇所の選択方法にあると言える。「全行動終了後」に評価が得られる環境においては進化型GNPを，「各行動終了後」に評価を得ることができる環境ではオンライン学習型GNPを用いるのが適切である。進化型GNPは，適合度による淘汰圧を受けながら試行錯誤的な進化を行う多点探索方式であり，考え方の基本は比較的長期間にわたる多様化と淘汰の結果による進化である。他方オンライン学習型GNPは1点探索方式であり，累積報酬を指標とした「悪い接続」をランダムに変更することにより，各行動直後に報酬をもらえるという利点を活かした漸近的な学習を行うことが可能である。したがって，進化型GNPは進化初期の大域探索に強い探索能力を発揮し，オンライン学習型GNPは環境変化に適宜適応しなければならないシステムに適している。

3. GNPのマルチエージェントシステムへの応用

これまで述べてきたGNPを実際の問題に適用する。GNPを用いた知的エージェントとそれを取り巻く環境は図2のように要約される。図に示すように，環境内のエージェントはその'脳'としてGNPプログラムを持つ。既に述べたように，GNPプログラム内の各ノードはそれぞれ簡単な判定/処理を行い，判定/処理後はノード処理と接続情報に依存して決定される次のノードへと制御を遷移する。このように，GNPプログラム上における制御の流れにしたがって，各ノードにおいて判定/処理が行われ，環境内でのエー

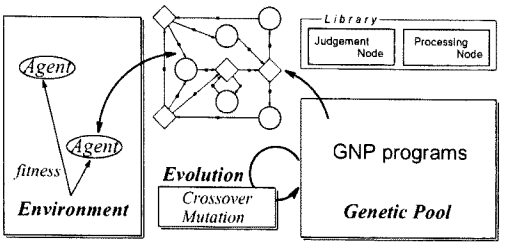


図2 GNPを用いたシステムの骨組み
Fig. 2. Skeleton of GNP system.

ジェントの行動系列が得られる。GNPの性能を評価する問題として，次節に示すタイルワールドを使用した。

〈3・1〉 タイルワールド　タイルワールドは，2次元格子状の環境内におかれた仮想的なエージェントのシミュレーション環境であり，動的環境におけるテストベッドとして知られている[16]。タイルワールドの世界はエージェント，タイル，障害物，穴からなり，それぞれ格子の1マスを占める。エージェントは1度に上下左右に1つのセルだけ移動できる。また，タイルに隣接しているエージェントはタイルの先に障害物がない場合，タイルを押すことができる。この問題におけるエージェントの目標はすべてのタイルをできるだけ早く穴に落とすことである。タイルワールドの世界の構成を図3に示す。

〈3・2〉 適合度関数　GNPプログラムは適合度によって評価され，選択・淘汰による進化を繰り返す。ここで，適合度はエージェントの目標達成度の指標となるように，すなわち，より多くのタイルをより早く落としたエージェントほどその適合度が高くなるよう設定した。具体的には式(1)により評価を行う。

$$\begin{aligned} fitness = & C_{tile} \times DropTile \\ & + C_{dist} \times \sum_{t \in TN} (InitialDist_t - LastDist_t) \\ & + C_{step} \times RestStep \quad \cdots\cdots\cdots\cdots\cdots (1) \end{aligned}$$

ここで，$DropTile$はエージェントが所定のステップ内で落としたタイルの総数を，$InitialDist_t$は初期状態におけるタイルtと穴の最小距離を，$LastDist_t$は最終状態におけるタイルtと穴の最小距離を示す。TNはタイルの添字の集合であり，本論文内では$|TN| = 3$である。また，$RestStep$は与えられたステップ内に全てのタイルを落とした場合にのみ計算され，(所定のステップ数−全てのタイルを落とすのに要したステップ数)で計算される。すなわち，第1項はエージェントが落としたタイルの数を，第2項はどれだけタイルを穴に近づけることができたかを，第3項はどれくらい速くタスクを達成できたかを評価する。C_{tile}，C_{dist}，C_{step}はそれぞれ各項にかける重み係数である。ここでは，各項の評価が偏らないように，$C_{tile} = 100$，$C_{dist} = 20$，$C_{step} = 1$としている。上記パラメータの下では，適合度が約500を越えた辺りから，環境内の全てのタイルを落とすことができたと判断して良い。また，初期状態よりもタイルを穴から遠ざけてしまった場合は適合度が負の値になることもありうる。

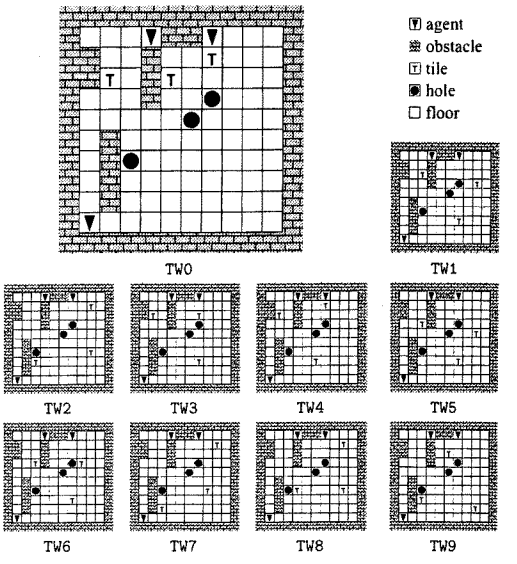


図3 学習用タイルワールド
Fig. 3. The tileworlds for training.

4. シミュレーション

前記タイルワールドにおいて，エージェントの行動系列をGNPにより構成することを試みる。すべてのエージェントは同一のGNPプログラムに基づいた行動系列に従い行動するものとした。すなわち均質的交配戦略[3]を採用している。

〈4・1〉 シミュレーション条件　GNP内の判定/処理ノード関数として，表1および2で示される2種類のノード関数集合を用意した。これらはエージェントの判定と処理の集合である。ここで前者を絶対処理，後者を相対処理と呼ぶことにする。絶対処理においては，'方向'はタイルワールドを天井から見下ろした際の上下左右情報によって与えられるものとする。一方，相対処理においては，'方向'はエージェントの向きに依存して与えられるものとする。すなわち，エージェントは自分から見た相対的な位置情報を用いて，環境に対して相対的な行動を行うものとしている。また，相対処理では，移動と方向変換の機能を分離した。

GNPプログラム内のノード数は，判定/処理ノードを各種類毎にN^{ini}個ずつ用意することとしており，以下の節では$N^{ini} = \{1, 5, 10\}$のケースについて比較を行う。このとき，GNPプログラム内のノード総数は$N^{ini}\times$(ノード関数の種類数)となる。

進化に関する規則および各種パラメータは次のように設定した。集団内の個体数は301，その内120個体を交叉(一様交叉)により，180個体を突然変異により毎世代生成する。残る1個体はエリート保存選択により再生される。親となる

表 1　判定/処理ノード関数（絶対処理）
Table 1. Judgement/Processing node functions (Absolute Rule).

判定	上のセル情報
	右のセル情報
	左のセル情報
	下方のセル情報
	最も近いタイルの方向
	最も近い穴の方向
	2 番目に近いタイルの方向
	最も近いタイルから見た最も近い穴の方向
処理	上へ進む
	右へ進む
	左へ進む
	下へ進む
	行動しない

表 2　判定/処理ノード関数（相対処理）
Table 2. Judgement/Processing node functions (Relative Rule).

判定	前方のセル情報
	右方のセル情報
	左方のセル情報
	後方のセル情報
	最も近いタイルの方向
	最も近い穴の方向
	2 番目に近いタイルの方向
	最も近いタイルから見た最も近い穴の方向
処理	前方へ進む
	右方へ 90°向きを変える
	右方へ 90°向きを変える
	行動しない

個体はトーナメント戦略により決定し，交叉率 $P_c = 0.1$[†]，突然変異率は $P_m = 0.01$[††]である。遺伝的オペレータによるノード間の接続規則としては，基本的に任意のノードに接続可能としている。世代数は問題に応じて適宜設定してあるが，進化に十分な世代数であると考えられる。また，エージェントに与えるステップ数は絶対処理の場合で 30，相対処理の場合は，移動と方向変換の処理を分離しているため 60 に設定した。

〈4・2〉 シミュレーション 1（絶対処理）　図 3の 1 種類のタイルワールド（TWO）において，絶対処理ノード関数を用い，$N^{ini} = \{1, 5, 10\}$ の条件で比較を行う。各条件ごとに 10 種の乱数系列を用いて独立したシミュレーションを行っており，以下の実行結果における「平均」とは，各世代における最良適合度を 10 回の実行結果で平均したものを意味する。

図 4に各世代における最良適合度の平均値を示す[†††]。図 4から，TWO において，$AbsN_5^{ini}1$, $AbsN_{10}^{ini}1$ では全ての実行においてタスクを達成する（全てのタイルを落とす）ことができる GNP プログラムを獲得できたことがわかる。一方，ノード数の少ない $AbsN_1^{ini}1$ の場合には 10 回中数回のタスク達成に留まった。

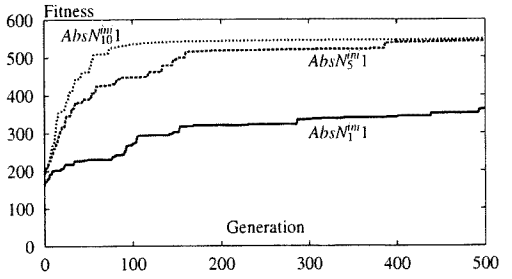


図 4　適合度曲線（絶対処理，TWO）
Fig. 4. Fitness curves (Absolute rule, TWO).

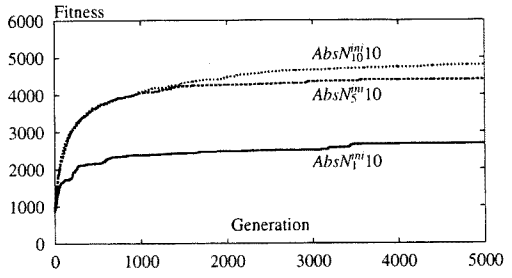


図 5　適合度曲線（絶対処理，TWO～TW9）
Fig. 5. Fitness curves(Absolute rule, TWO～TW9).

次に，図 3の全てのタイルワールド（TWO ～ TW9）において上記と同じ条件（$N^{ini} = \{1, 5, 10\}$）でシミュレーションを行った際の結果を図 5に示す[††††]。なお，複数のタイルワールドを同時に用いて進化を行う際は，適合度をそれぞれのタイルワールドの適合度の和で定義することにする。

図 5においてもノード数の違いにより大きな差が現れることが確認できる。$AbsN_1^{ini}10$ では十分な学習回数にも関わらず，全てのタイルを落とすことが可能な GNP プログラムを一度も獲得することはできなかったのに対し，$AbsN_5^{ini}10$, $AbsN_{10}^{ini}10$ では全てのタイルワールドにおいてタスクを達成することが可能な GNP プログラムを獲得することができた。N^{ini}が大きいほど，タスクを達成した回数も多く，適合度の平均値も高くなる傾向が見られる。これから，知識を蓄えるための十分なノード数を用意することにより，複数の異なる環境においてもタスクを達成可能なプログラムを合成することができることがわかる。プログラム内のノード数の増大につれて，表現能力が向上することは当然であり，GNP の場合ノード数が増大することは，より多くの判定/処理の組合せパターンを表現可能になることに相当する。$AbsN_1^{ini}10$ では 13 種のノード関数が各々1 個ずつし

[†]プログラム内の全ノードの内，P_cに相当する比率のノードを確率的に選択し，選択されたノードからの接続を 2 つの親プログラム間で交換する。
[††]プログラム内の全接続の内，P_mに相当する比率の接続を確率的に選択し，選択された接続をランダムに変更する。
[†††]図中の適合度曲線のラベルに見られる 'Abs' は絶対処理ノード関数を使用したことを，'N_k^{ini}' は $N^{ini} = k$であることを，最後の数字 '1' は 1 種のタイルワールドにおける実行であることをそれぞれ示す。
[††††]図中の適合度曲線のラベルに見られる最後の数字 '10' は 10 種のタイルワールドを使用したことを示す。

か与えられておらず,「現在の情報のみから一意に次の行動が求まる if～then 形式の木構造」よりも低い表現能力に留まっているといえる。一方，$AbsN_5^{ini}10$，$AbsN_{10}^{ini}10$ では過去の行動に依存した行動を表現することができる。このことから，GNP の主張点のひとつである「過去の行動に依存する行動系列」が表現能力を向上させるために非常に重要であることがわかる。

〈4・3〉 シミュレーション 2（相対処理） TWO において，相対処理ノード関数を用い，$N^{ini}=\{1,5,10\}$ の条件で比較を行った。図 6に各世代における最良適合度の平均を示す†。N^{ini}と適合度の関係は絶対処理の場合と同様の傾向を示した。しかし，いずれの場合もタスクを達成するプログラムを得ることができたものの，絶対処理おける結果と比べると，タスク達成率が減少している。絶対処理では $N^{ini}=5,10$ のどちらの場合でも 100%のタスク達成率であったのに対し，相対処理では $N^{ini}=5$ の場合では 80%の達成率に留まり，$N^{ini}=10$ の場合で 100%の達成率となった。したがって絶対処理に比べ，相対処理の進化は困難であるようだ。特に，$RelN_1^{ini}1$ の場合では初期世代において局所解に陥り，進化の停滞が多く見られた。相対処理の場合，局所解に陥るケースが頻発しており，そのため絶対処理の場合に比べて，やや進化に世代数がかかる傾向がある。ノード数を増やすことにより局所解に陥りにくくなる理由は，表現能力の向上に加えて，プログラムの冗長性の増加にある。後者は，適合度に影響を及ぼさない遺伝子が淘汰の危険をともなうことなく変化することが許され，それらの変化が蓄積し，ある時点で変化の蓄積したノードを使用するケースが発生したときに進化の効果として現れるという中立遺伝子の点から説明できる。

我々の用意した絶対処理と相対処理の大きな違いは次の 2つであるといえる。1）絶対処理はその性質上，エージェントはどの方向にも自由に移動することができるため，行動政策の変更が容易である。他方，相対処理では移動機能と方向変換機能を分離しているため，行動政策の変更の際に，方向変換処理と移動処理を同時に獲得する必要がある。そのため，絶対処理に比べ，突然変異によって偶然良い行動政策を発見する確率が低くなる。相対処理の進化が困難である理由は，エージェントが持つ方向感覚ではなく，機能を細分化したことに原因があると考えられる。2）絶対処理では環境を回転させるという単純な環境変更に対処できない。他方，相対処理では環境の回転にかかわらず高い適合度を獲得することが可能である。すなわち，相対処理ノード関数を用いたプログラムの方が，より高度な知識をプログラム内に含蓄することが可能といえる。

次に，TWO～TW9 において同様な条件の下でシミュレーションを行った際の結果を図 7に示す。図 7でも，これまでと同様に N^{ini}が大きいほど高い適合度を得ることができている。しかし，$N^{ini}=10$ よりも $N^{ini}=5$ の場合の方が良い結果を得ている。これは，探索空間の増大による弊害が，表現能力の向上および中立遺伝子の恩恵を上回ったためであると考えられる。前述したように，相対処理では移動機能と方向変換機能を分離しているため，進化が困難である。そのため絶対処理では現れなかったノード数増加の弊害が表れやすかったのであろう。探索空間と表現能力のトレードオフに関しては，別の論文で述べることにする。

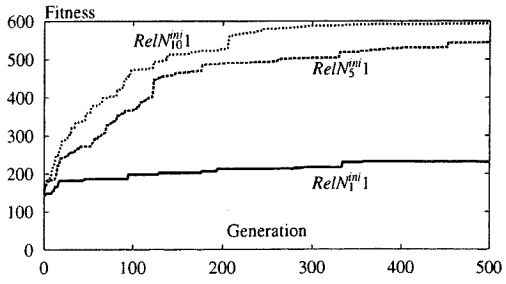


図 6 適合度曲線（相対処理，TWO）
Fig. 6. Fitness curve (Relative rule, TWO).

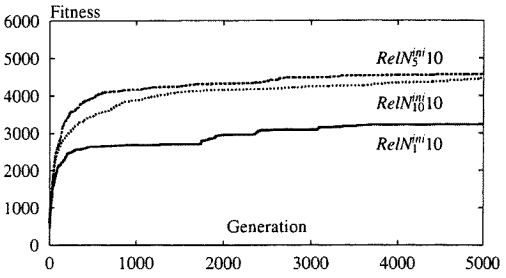


図 7 適合度曲線（相対処理，TWO～TW9）
Fig. 7. Fitness curve (Relative rule, TWO～TW9).

〈4・4〉 シミュレーション 3（汎化能力） シミュレーション 1および 2で獲得された GNP プログラムを，テスト用タイルワールド（図 8 TWa～TWh）で実行し，得られたプログラムの汎化能力を検証する。テスト環境 TWa～TWd は学習用環境（図 3）を若干変更したものに留め，特に，TWa は TWO，TW4 と酷似した環境とした。逆に，TWe は TWO～TW9 のどれとも類似しないように設計している。また，TWf～TWh は，それぞれ学習用環境 TWO を，上下対称，左右対称，回転したものである。テスト環境における平均適合度を表 3に示す。表 3中の表記はシミュレーション 1および 2における条件にそれぞれ対応している。

表 3から，絶対処理，相対処理のどちらの場合でも，10種類のタイルワールドで進化を行った GNP プログラムは 1種類のタイルワールドのみで学習したプログラムよりも明らかに高い汎化能力を持つ傾向が見られる。テスト用環境 TWO，TW4 と特に酷似した TWa では高い適合度を得ており，少々の環境変動ではプログラムの有用性が極端に低下することはないようだ。この結果から，同時に多くの環境に順

†図中の適合度曲線のラベルに見られる '*Rel*' は相対処理ノード関数を使用したことを示す。

表3　テスト用タイルワールドにおける平均適合度
Table 3. Averaged fitness value in the test tileworlds.

	$AbsN_1^{ini}1$	$AbsN_5^{ini}1$	$AbsN_{10}^{ini}1$	$AbsN_1^{ini}10$	$AbsN_5^{ini}10$	$AbsN_{10}^{ini}10$
TWa	214.0	166.0	140.0	297.0	397.0	399.6
TWb	134.0	94.0	156.0	128.0	142.8	192.0
TWc	- 8.0	- 16.0	- 12.0	20.0	110.0	156.0
TWd	152.0	110.0	100.0	166.0	102.0	112.0
TWe	0.0	0.0	- 20.0	- 22.0	- 10.0	0.0
TWf	30.0	52.0	12.0	22.0	62.0	66.0
TWg	8.0	38.0	46.0	52.0	90.0	14.0
TWh	62.0	42.0	66.0	44.0	130.0	80.0
	$RelN_1^{ini}1$	$RelN_5^{ini}1$	$RelN_{10}^{ini}1$	$RelN_1^{ini}10$	$RelN_5^{ini}10$	$RelN_{10}^{ini}10$
TWa	84.0	42.0	60.0	238.0	114.0	213.0
TWb	48.0	102.0	104.0	68.0	130.0	122.0
TWc	4.0	10.0	0.0	52.0	142.7	78.0
TWd	128.0	58.0	72.0	170.0	138.0	152.0
TWe	- 4.0	10.0	10.0	- 10.0	- 18.0	24.0
TWf	126.0	96.0	104.0	130.0	148.0	114.0
TWg	126.0	96.0	104.0	130.0	148.0	110.0
TWh	194.0	439.6	593.7	280.0	343.7	353.3

応できるよう進化を重ねることにより，類似した環境において汎化性のあるプログラムを獲得することができたと言える。

一方，どのテスト用環境とも類似しないTWeでは，絶対処理，相対処理を用いて進化したどのプログラムも有効に機能しなかった。これは，本論文で使用した学習用環境は，初期状態におけるエージェントの位置，エージェントの向き，障害物の位置に関してほとんど同一であるため，偏った知識の獲得に収束しやすく，必要な一般知識を十分に獲得することができなかったためである。したがって，様々な環境において通用する高い知識を含蓄するプログラムを獲得するためには，偏りのない学習用環境を使用し，学習環境数を増加させる必要がある。

テスト環境TWa～TWdにおいては，絶対処理の方が相対処理よりも若干高い適合度を示しているが，テスト環境TWOを対称変換，回転変換したTWf～TWhでは突然汎化能力が低下しており，絶対処理では知識の獲得というよりは，タスク達成手順の獲得に留まっていると考えられる。一方，相対処理ではその性質上，回転変換に対しては高い適合度を得ており，対称変換にたいしても絶対処理に比して高い適合度を得た。このため，シミュレーション2で述べたように学習の速度，局所解へ陥りやすい欠点があるものの，タスク達成手順の獲得ではなく，断片的ではあるものの知識の獲得に至っていると言える。

5. まとめ

進化型GNPによるエージェントの行動系列の生成を，タイルワールド環境で行った。十分なノード数を用意することにより，単一の環境下でタスクを達成する行動系列を容易に生成可能であることを明らかにした。また，複数の環境でタスクを達成することができるプログラムも生成可能であり，得られたプログラムは類似環境において高い汎化能力を持つことを実証した。これは，GNPが異なる環境に共通する知識を抽出し，巧みにプログラム内部に組み込むことが可能であることを表している。

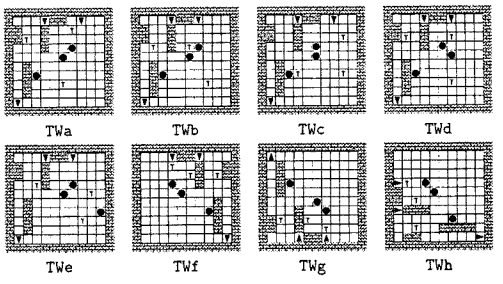


図8　テスト用タイルワールド
Fig. 8. The Tileworlds for Test.

ノード関数による違いとしては，絶対処理は機能が単純である為，学習速度が速く，局所解にも陥りにくいことを示した。しかし，絶対処理により得られたプログラムの汎化能力は類似環境においては高いものの，環境の向きが変わると極端に低下する。逆に相対処理では，類似環境での汎化能力は絶対処理に劣るものの，環境の向きの変化に関してはロバストであった。また，ノード関数の機能の細分化が，探索能力の大幅な低下に繋がることを示した。

ノード数の影響としては，複雑な問題に対処するためには十分な知識を蓄える為に多数のノードを必要とすること，またノード数の冗長性が進化を助長することを示した。しかし，探索空間の増大による弊害も存在するため，ノード数の選択に関しては慎重に決定する必要がある。

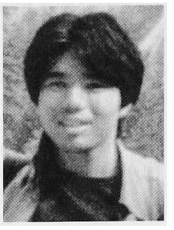
今後の課題は，より多くの様々な環境で進化を行うことにより，本論文で得られたプログラムよりもさらに高い一般的な知識を獲得することである。また，進化型GNPとオンライン学習型GNPの統合は非常に興味深い所である。

（平成12年4月24日受付，同14年6月19日再受付）

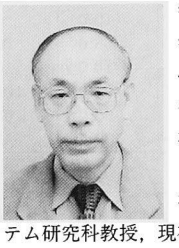
文　献

（1） R. A. Brooks: "Robust layered control system for a mobile robot", *IEEE Journal of Robotics and Automation*, Vol.2, No.1, pp.14-23 (1986-1)
（2） 山田誠二: 適応エージェント, 共立出版, (1997)
（3） 伊庭斉志: 進化論的計算の方法, 東京大学出版会, (1999)
（4） 伊庭斉志: 遺伝的プログラミング, 東京電気大学出版局, (1996)
（5） J. R. Koza: "Evolution of Subsumption Using Genetic Programming", Computer Science Department, Stanford University, (1992)
（6） W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone: Genetic Programming ～ An Introduction, Morgan Kaufmann Pulbishers, Inc.
（7） H. Iba: Evolutionary Learning of Communicating Agents, Information Science, 108(1-4), pp.181-205 (1998)
（8） H. Katagiri, K. Hirasawa, and J. Hu: "Genetic network programming-application to intelligent agents-". *Proc. IEEE International Conf. Syst., Man and Cybernetics*, pp.3829–3834 (2000)
（9） H. Katagiri, K. Hirasawa, J. Hu, and J. Murata: "Network structure oriented evolutionary model-genetic network programming-and its comparison with genetic programming". *2001 Genetic and Evolutionary Computation Conference Late Breaking Papers*, pp.219–226, San Francisco, California, USA, 9-11 July (2001)
（10） K. Hirasawa, M. Okubo, H. Katagiri, J. Hu, and J. Murata: "Comparison between Genetic Network Programming and Genetic Programming using evolution of ant's behaviors", *T. IEE Japan*, Vol.121, No.6, pp.1001-1009 (2001-6) (in Japanese)
平澤・大久保・片桐・胡・村田:「蟻の行動進化における Genetic Network Programming と Genetic Programming の性能比較」, 電気学会論文誌 C, **121**, 6, pp.1001-1009 (2001-6)
（11） S. Mabu, K. Hirasawa, J. Hu, J. Murata: "Online Learning of Genetic Network Programming", *T. IEE Japan*, Vol.122, No.3, pp.355-362 (2002-3) (in Japanese)
間普・平澤・胡・村田:「遺伝的ネットワークプログラミングのオンライン学習」, 電気学会論文誌 C, **122**, 3, pp.355-362 (2002-3)
（12） X. Yao: "Evolutionary Programming Made Faster", *IEEE Trans. Evolutionary Computation*, Vol.3, No.2, pp.82-102 (1999-2)
（13） D. B. Fogel: "An introduction to simulated evolutionary optimization", *IEEE Trans. Neural Networks*, Vol.5, No.1, pp.3-14 (1994-1)
（14） A. teller and M. Veloso: PADO: Learning Tree-Structured Algorithms for Orchestration into an Object Recognition System, Carnegie Mellon University Technical Report Library, (1995)
（15） A. Teller and M. Veloso: PADO: A New Learning Architecture for Object Recognition, Symbolic Visual Learning, pp.81-116 (1996)
（16） 沼岡・大沢・長尾: マルチエージェントシステム, 共立出版, 1998 自律分散システム・シンポジウム資料, pp.407-410 (2000)

片桐広伸（非会員） 1976年2月21日生。1998年3月九州大学工学部電気系学科卒業。2000年3月同大学大学院システム情報科学研究科電気電子システム工学専攻修士課程修了。現在，同大学大学院システム情報科学府電気電子システム工学専攻博士後期課程在学中。



平澤宏太郎（正員） 1942年1月26日生。1964年3月九州大学工学部電気工学科卒業。1966年3月同大学大学院工学研究科修士課程電気工学専攻修了。同年4月（株）日立製作所入社 日立研究所勤務，1989年8月同研究所副所長。1991年8月同大みか工場主管技師長。1992年12月九州大学工学部教授，1996年5月同大学院システム情報科学研究科教授，2002年9月早稲田大学大学院情報生産システム研究科教授，現在に至る。計測自動制御学会，電気学会，情報処理学会，IEEEの各会員。工学博士。



胡　敬炉（正員） 1985年中国中山大学大学院修士課程修了。同年，同大学電子工学科助手，1988年同講師。1993年来日，1997年九州工業大学情報工学研究科博士後期課程修了。同年4月九州大学ベンチャービジネスラボラトリ非常勤研究員を経て，同年8月同大学システム情報科学研究科助手，現在に至る。計測自動制御学会電気学会の各会員。情報工学博士。

村田純一（正員） 1959年1月19日生。1986年3月九州大学大学院工学研究科博士後期課程電気工学専攻修了。同年4月同大学工学部助手。1988年4月同大学工学部助教授，現在に至る。計測自動制御学会，電気学会，システム制御情報学会，IEEEの各会員。工学博士。