論　文

ノード数可変型 Genetic Network Programming

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*

Variable Size Genetic Network Programming

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

Genetic Network Programming (GNP) is a kind of evolutionary methods, which evolves arbitrary directed graph programs. Previously, the program size of GNP was fixed. In the paper, a new method is proposed, where the program size is adaptively changed depending on the frequency of the use of nodes. To control and to decide a program size are important and difficult problems in Evolutionary Computation, especially, a well-known crossover operator tends to cause bloat. We introduce two additional operators, add operator and delete operator, that can change the number of each kind of nodes based on whether a node function is important in the environment or not. Simulation results shows that the proposed method brings about extremely better results compared with ordinary fixed size GNP.

キーワード：遺伝的プログラミング，遺伝的ネットワークプログラミング，進化論的計算，有向グラフ，プラニング，タイルワールド
Keywords: Genetic Programming, Genetic Network Programming, Evolutionary Computation, arbitrary directed graph, planning, the tileworld

1. はじめに

生物界における進化現象を規範とした進化論的計算手法によるプログラム合成では，木構造により表現された遺伝子の進化が主流である。木構造の利点は，交叉を容易に実現することが可能である点にあり，Genetic Programming (GP)[1][2]の枠組みで多数の研究が進められている。一方，グラフ構造のプログラム合成を進める研究も進められており（Parallel Algorithm Discovery and Orchestration[3], Parallel Distributed Genetic Programming[4], Cartesian Genetic Programming[5], cellular encoding[6], edge encoding[7], Evolutionary Programming[8]~[10]），それらはグラフ構造が木構造よりも一般性の高い表現能力を持つ点に端を発している。一般に，グラフ構造は以下の点で木構造よりも優れた面を持ち，これらの特徴はプログラムの表現および進化の過程において非常に有益なものである。

（1） ループや繰り返しを自然に表現可能
（2） グラフ上のノードの再利用／共有が可能†
（3） 適合度に中立なノード群が進化を蓄積することによる大進化が可能

我々の提唱する手法 Genetic Network Programming (GNP)[11]~[15]もグラフ構造により表現されるプログラムの合成を行う。GNPは有向リンクで接続された単純な機能を持つ関数群により表現されるプログラムの自動合成を目的とする手法であり，他手法との相違点は，グラフ構造の利点の着目点にある。有向グラフを形成するGNPプログラムは，あるノードが活性化されるためには必ずそのノードに接続しているノードが活性化されていなければならない。そのため，あるノードが活性化したという事実は，そのノードに接続しているノードが過去に活性化したことを示唆する。同様な議論を繰り返すことにより，活性化したノードは，過去に活性化したノード系列を暗に記憶してい

*九州大学大学院 システム情報科学府
〒 812–8581 福岡市東区箱崎 6–10–1
Graduate School of Information Science and Electronical Engineering, Kyushu University.
6–10–1, Hakozaki, Higashi–ku, Fukuoka, Japan 812–8581
**早稲田大学大学院 情報生産システム研究科
〒 808–0135 福岡県北九州市若松区ひびきの 2–2
Graduate School of Information, Production and Systems of Waseda University.
2–2, Hibikino, Wakamastu–ku, Kitakyushu–shi, Fukuoka, Japan 808-0135

†木構造の GP では ADF と呼ばれる自動関数定義機能により実現可能。

ると考えられる。換言すると，グラフ構造上に配置された各ノードは，それぞれのノードが活性化した時点において，その事実以上の情報を持つことを意味する。これをグラフ構造の**暗黙的記憶機能**と呼ぶことにする。暗黙的記憶機能による過去の情報の有効活用により，多数の情報が氾濫する環境内において，無駄な環境観測を省略し，真に必要な情報のみを利用することが期待される。即ち，GNP は有限オートマトンとは異なり，部分情報の活用による行動系列を獲得することができ，部分観測マルコフ決定過程環境下において高い性能を発揮すると考えられる。

これまでの研究では，GNP プログラム内のノード数は進化を通して固定値であった。本論文では GNP プログラム内のノード数を適合度に対するノードの貢献度に応じて世代毎に制御する手法を提案する。GP ではプログラムサイズは交叉や突然変異の結果として変化するが，それに伴う bloat と呼ばれるプログラムサイズの爆発的増加が起こり，プログラムサイズは研究の焦点となっている[16]。提案手法はプログラムサイズを適応的に制御することにより，小さなプログラムの利点である探索空間の縮小にともなう進化の高速化，大きなプログラムの利点である表現能力の増大および局所解の回避間の妥協点の発見を期待する。

本論文の構成は以下の通りである。次章では GNP の基本的構造および定義について簡単に述べる。第 3 章では提案手法を実現する 2 つの遺伝的オペレータの導入を行う。第 4 章はシミュレーション環境および条件，第 5 章はシミュレーションの結果および考察を述べ，第 6 章で結論を述べる。

2. Genetic Network Programming[11]~[15]

〈2·1〉 基本構造　本節では GNP の基本構造を示す。既に述べたように，GNP は進化論的計算手法の 1 種であり，有向グラフで表現されるプログラムの自動合成を行う。一般に GNP システムは次の 6 文字組 $\langle \mathcal{I}, \mathcal{R}, \mathcal{S}, \mathcal{G}, \mathcal{F}, \mathcal{D} \rangle$ で与えられ，GNP プログラムは $\langle \mathcal{N}, \mathcal{C} \rangle$ で同定される。以下にその概要を示す。

- $\mathcal{I}$ ： 個体の初期化規則
- $\mathcal{R}$ ： 再生規則
- $\mathcal{S}$ ： 再生における個体の選択規則
- $\mathcal{G}$ ： 個体に適用される遺伝的オペレータ集合
- $\mathcal{F}$ ： 適合度評価関数
- $\mathcal{D}$ ： ノード関数定義集合
  - ▷ 開始ノード S
  - ▷ 判定ノード集合 $\mathcal{J} = \{J_1, J_2, \cdots, J_{n_j}\}$
  - ▷ 処理ノード集合 $\mathcal{P} = \{P_1, P_2, \cdots, P_{n_p}\}$
  - ▷ イントロンノード IN
- $\mathcal{N}$ ： プログラム内のノード集合
- $\mathcal{C}$ ： プログラム内のノード間接続集合

ここで，ノード関数定義集合 $\mathcal{D}$ は GNP プログラムの基本要素であるノード関数を定義し，それらの具体的機能および個数（$|\mathcal{J}|$, $|\mathcal{P}|$）は環境や問題に応じてあらかじめ決定される。

GNP プログラムは $|\mathcal{N}|$ 個のノードが，ノード間接続集合 $\mathcal{C}$ で定義される弧（有向リンク）によって接続された有向グラフとして表現される。特に，ノード間接続集合 $\mathcal{C}$ はプログラムの構造を決定する重要な要素である。$\mathcal{N}$ は $\mathcal{D}$ により定義された複数のノードからなり，一般に，$\mathcal{N}$ は 1 つの**開始ノード**と $\mathcal{J}$ に属する**判定ノード**，$\mathcal{P}$ に属する**処理ノード**，**イントロンノード** IN の重複を許したノード集合で構成される。判定ノードは，プログラムの実行において定められた判定処理を行い，その判定結果にしたがった条件分岐の役割を果たす。一方，処理ノードは，プログラムの実行において定められた行動処理を行う機能を持つ。GNP の判定ノード，処理ノードは，その機能上，GP の非終端記号，終端記号にそれぞれ対応づけることができる。イントロンノードはノード間の橋渡しの役割を果たすことを目的としたノードであり，いかなる判定／処理も行わない。即ち，進化の過程における接続の柔軟性の実現のみを目的とする。GA，GP で議論されているイントロンとは異なり，進化の過程で自然発生するものではなく，プログラムに人為的に組み込まれたノードである。

GNP プログラムの実行は次のようにして実現される。まず，開始ノード（start node）が活性化され，以下に示す遷移法則に従い，有向グラフ上の制御の流れが決定される。開始ノードはその弧によって指示されたノードに制御を遷す。判定ノード J_m は $\mathcal{D}$ で定義された m 番目の判定処理を行い，その判定結果にしたがって，複数の弧で指示されるノード群から 1 つのノードを選択し，制御を遷す。処理ノード P_n は $\mathcal{D}$ で定義された n 番目の行動処理を行い，その弧によって指示されるノードへ制御を遷す。イントロンノード IN は具体的な動作を行わず，単に，弧によって指示されるノードへ制御を遷す。ある時点で制御権をもつノードの状態を**活性化状態**と定義すると，活性化状態は有向グラフ上のノードを次々に遷移することになる。この制御遷移はプログラムの実行中を通して行われ，次の判定／行動は常に以前の判定／行動の制約を受けることになる。この点が木構造のプログラムや終了ノードをもつグラフ構造のプログラムとの大きな相違点であることに注意していただきたい。

〈2·2〉 プログラムの表現　図 1 に GNP プログラムの表現型と遺伝子型を示す。図 1 に示されるように，集団中の全ての個体は多数のノードからなる有向グラフで表される。ここで，個体に含まれるノードの総数 $|\mathcal{N}|$ を**プログラムサイズ**と呼び，以下の式が成り立つものとする。

$$|\mathcal{N}| = \sum_{d \in D} N_d \quad \cdots\cdots (1)$$

ここで，N_d は，個体内に存在するノード関数 d ($\forall d \in \mathcal{D}$) の総数であり，個体初期化時点ではそれぞれ**初期ノード数** N^{ini} 個のノードを持つ（$N_d = N^{ini}$, $\forall d \in \mathcal{D} \backslash S$）。これまでの GNP に関する研究では N_d はそれぞれ同数かつ固定値であった。

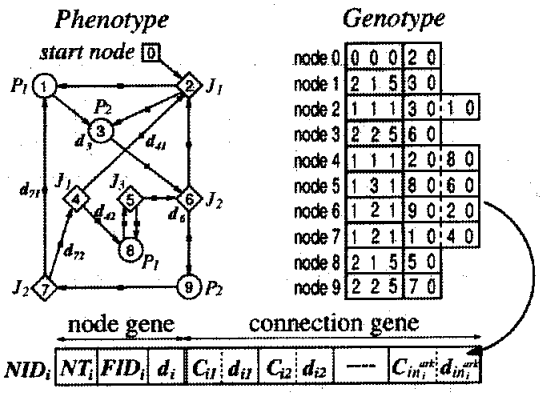

図 1　GNP プログラムの表現型と遺伝子型

Fig. 1.　Phenotype and genotype of GNP program.

表現型と遺伝子型間のコード化には直接的なコード化を採用しており，遺伝子型は個々のノード遺伝子の集合からなる。図 1 に示されるノード遺伝子 i の各遺伝子座はそれぞれ次のような情報を持つ。NID_i はノード識別番号であり，プログラム内のそれぞれのノードを識別するために各ノード毎にノード固有の値が設定されている。NT_i はノード型を表し，ノードの持つ関数の型を示す遺伝子座である（0：開始ノード，1：判定ノード，2：処理ノード，3：イントロンノード）。FID_i は関数識別番号を表し，NT_i と FID_i の組によってノード関数を特定する。例えば，J_2 は $NT_i = 1$ と $FID_i = 2$ によって特定される。d_i はノード i における判定／行動処理に要する遅れ時間を示す。C_{ij} $(j = 1, 2, \cdots, n_i^{arc})$ はノード i からの j 番目の弧によって接続されているノードのノード識別番号を表す。n_i^{arc} はノード i が持つ弧の本数であり，ノード i のノード関数に依存する。一般に判定ノードでは複数の判定結果があるため $n_i^{arc} \geq 2$ であり，起動ノード，処理ノード，イントロンノードでは遷移が一意に決定されるため $n_i^{arc} = 1$ である。n_i^{arc} 個の判定結果が存在する判定ノード i は，n_i^{arc} 個のそれぞれの判定結果に対応したノード遷移規則を持ち，その遷移規則は C_{ij} により決定される仕様となる。d_{ij} はノード i からノード C_{ij} へ制御を遷移する際に要する遅れ時間を示す。

集団内の全個体間において，上記遺伝子は接続遺伝子のみ異なるものとする。即ち，ある世代における全ての個体間において，N_d $(\forall d \in \mathcal{D})$ は互いに等しく，同じノード識別番号を持つノード同士は同じノード関数を持つものとする。上記規則は，次節に述べる交叉の実現を容易にするために設定されている。

〈2·3〉 プログラムの進化　GNP では 2 種類の単純な遺伝的オペレータ，交叉および**突然変異**を用いてプログラムの進化を実現する。集団内の総個体数を**集団数**と呼び，各世代毎に集団数と同数の個体を生成し，個体の入れ替えを行う。各世代に交叉および突然変異によって生成される個体数をそれぞれ**交叉個体数**，**突然変異個体数**とし，常に一定数の新しい個体を生成する。また，最も適合度の高い 1 個体を次世代に保存する。

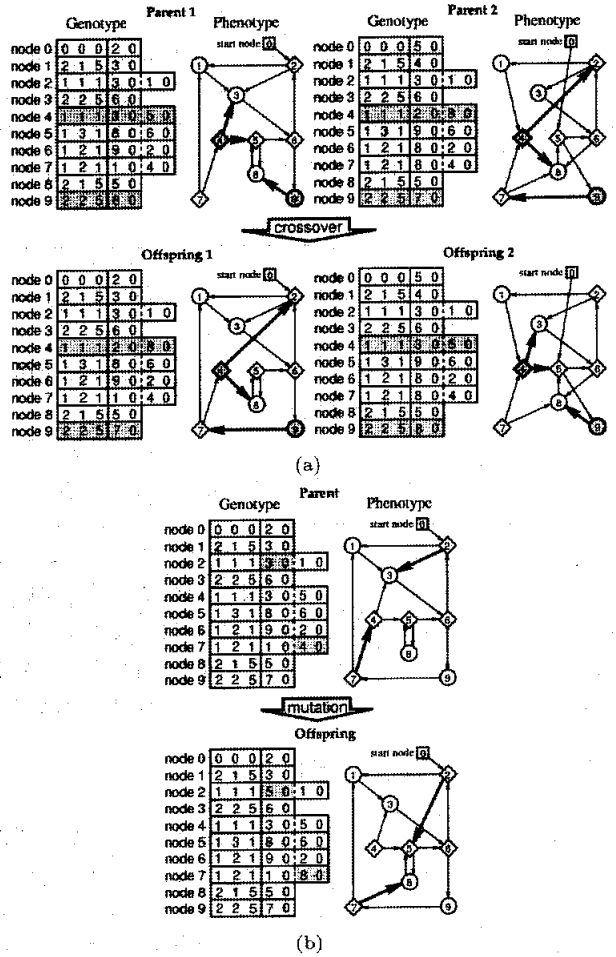

図 2　(a) 交叉オペレータと (b) 突然変異オペレータ

Fig. 2.　(a) Crossover and (b) Mutation operators.

〈2·3·1〉 交叉オペレータ　交叉は 2 つの親個体に作用し，2 つの子個体を生成する。具体的には，図 2 (a) に示されるように一様交叉を用いる。交叉のアルゴリズムは次のようになる。

交叉のアルゴリズム

（1）トーナメント選択を 2 回行い，親となる 2 個体を選択する

（2）交叉比率 P_c にしたがって，各ノード毎に交叉を行うノードを選択する

（3）（2）で選択された 2 個体間の対応するノード遺伝子（ノード識別番号が同一）を互いに交換する

（4）生成された 2 個体を次世代の個体とする

本構造の GP では「部分木」[1] を，幾つかのグラフ構造を進化させる手法では「部分グラフ」[4][17] を考慮することにより交叉を行うが，GNP では対応するノードの接続の交換をすることにより，「部分プログラム」の交叉を実現し

ている。

〈2・3・2〉 突然変異オペレータ 突然変異は1つの親個体に作用し，1つの子個体を生成する。具体的には，図2（b）のように弧の接続をランダムに変更することにより個体の進化を行う。突然変異のアルゴリズムは次のようになる。

突然変異のアルゴリズム

（1） トーナメント選択を1回行い，親となる1個体を選択する
（2） 突然変異率 P_m にしたがって，各弧毎に突然変異を行う弧を選択する
（3） （2）で選択された弧をランダムに変更する
（4） 生成された1個体を次世代の個体とする

これらの遺伝的オペレータは生成されるプログラムの文法的な正しさを保証するものである（致死遺伝子を生成しない）。ノード関数の突然変異や，より複雑な交叉オペレータの適用も考えられるが，それらは破壊的な進化を引き起こす可能性が高いとみなし，上記オペレータを採用している。

3. ノード数可変型 GNP

〈3・1〉 基本概念 一般的に，大きなプログラムは表現能力や進化能力の点で小さなプログラムより優れていると考えられる。これは，GNPプログラムにおいてプログラム内のノード数が増えることは，様々な状態を間接的に記憶する領域が増える事に繋がるためである。また，適合度に影響を持たないノード（活性化されなかった過剰なノード）は，中立遺伝子として進化を蓄積し，有効なノード群の一部になる可能性も秘めている。他方，大きなプログラムは進化効率の面では小さなプログラムに劣ると考えられる。これは単純に探索空間が飛躍的に拡大することによる。また，大きなプログラムはメモリ空間と計算時間を浪費する欠点も持つ。あらかじめ最適なプログラムサイズを知ることは困難であるため，プログラムサイズを適切な指標に基づいて制御することは様々な利点を生むと期待できる。

GPではプログラムサイズは交叉や突然変異の結果により変更されるが，bloatと呼ばれるプログラムサイズの爆発という問題を持つ。そこで，提案手法ではプログラムサイズを制御するための2つの遺伝的オペレータを新たに導入し，積極的なプログラムサイズの制御を試みる。

GNPプログラムを実行すると判定ノードや処理ノードの使用頻度がノード関数の種類によって大きく異なる傾向がある。この使用頻度がノードの貢献度の指標になると仮定し，プログラムサイズの制御に利用する。即ち，頻繁に活性化されるノード関数は追加し，活性化されなかったノードは削除する方針にしたがってノード数の制御を試みる。

〈3・1・1〉 追加オペレータ ノード関数 $d\,(\forall d \in \mathcal{D}\backslash S)$ の**活用度** α_d をノード関数 d がプログラム内にしめる空間的占有率に対する時間的発火率の比によって定義する。

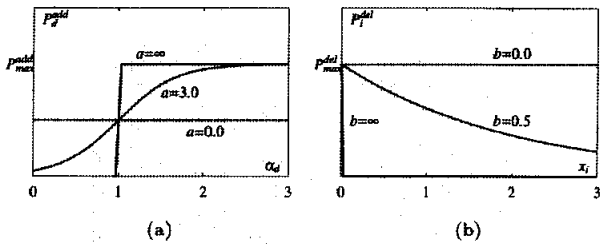

図3 （a） P_d^{add} と （b） P_i^{del} の概形
Fig. 3. Sketches of (a) P_d^{add} and (b) P_i^{del}.

$$\alpha_d = \frac{X_d/T}{N_d/|\mathcal{N}|} = \frac{X_d|\mathcal{N}|}{N_d T} \quad (\forall d \in \mathcal{D}\backslash S) \cdots\cdots\cdots (2)$$

ここで，X_d はノード関数 k が活性化された回数，T は活性化状態の総遷移回数（$T = \sum_{k\in\mathcal{D}\backslash S} X_k$），$N_d$ はプログラム内に存在するノード関数 d の個数である。（2）式により，$\alpha_d > 1.0$ のときノード関数 d は不足しているとみなすことができる。

追加オペレータは α_d の関数で示される**追加確率**（P_d^{add}）にしたがって，ノード関数レベルで追加を行う。ここで，

$$P_d^{add} = \frac{P_{max}^{add}}{1+\exp(-a(\alpha_d - 1))} \cdots\cdots\cdots\cdots\cdots (3)$$

であり，図3（a）にその概形を示す。P_{max}^{add} はプログラムサイズが拡大する強さを，a は追加されるノードの選択におけるランダム度を設定するパラメータである。このとき追加オペレータのアルゴリズムを以下のように定義する。また，プログラム内に存在する各ノード関数の最大許容数を**最大ノードサイズ** N^{max} であらかじめ設定する。

追加オペレータのアルゴリズム

各ノード関数 $d\,(\forall d \in \mathcal{D}\backslash S)$ 毎に，$N_d < N^{max}$ であるならば，(3) 式で示される追加確率 P_d^{add} の確率で以下の操作を行う。ここで，活用度 α_d は最良個体を基準に算出したものを用いる。
（1） $N_d \leftarrow N_d + 1$
（2） ノード関数 k を集団内の全個体に1個追加し，新たに固有のノード識別番号を割り当てる
（3） 追加されたノードから発する弧をランダムに設定する

〈3・1・2〉 削除オペレータ **削除オペレータ**は適合度への貢献度の少ないノード，すなわち活性化回数の少ないノードを個体から取り除く。ここで，各ノード毎の活性化回数を $x_i (i = 1, 2, \cdots, |\mathcal{N}|)$ とすると，削除オペレータは x_i の関数として表される**削除確率**（P_i^{del}）にしたがって，ノードレベルで削除を行う。ここで，

$$P_i^{del} = P_{max}^{del} \exp(-bx_i) \cdots\cdots\cdots\cdots\cdots\cdots (4)$$

であり，図3（b）にその概形を示す。P_{max}^{del} はプログラムサイズが縮約する強さを，b は削除されるノードの選択におけるランダム度を設定するパラメータである。プログラ

ム内に存在する各ノード関数の個数の下限を**最小ノードサイズ** N^{min} であらかじめ設定し，削除オペレータのアルゴリズムを以下のように定義する。

削除オペレータのアルゴリズム

各ノード $i\,(i=1,2,\cdots,|\mathcal{N}|)$ 毎に，$N_d > N^{min}$ であるならば，(4) 式で示される削除確率 P_i^{del} の確率で以下の操作を行う。ここで，活性化回数 x_i は最良個体を基準に計算を行う。

（1） $N_d \leftarrow N_d - 1$

（2） 選択されたノードに対応するノード（ノード識別番号が同じ）を集団内の全ての個体から 1 個削除する

（3） 削除されたノードに接続していた弧をランダムに変更する

（4） 削除されたノードを考慮して，ノード識別番号の再割り当てを行う。この際，弧の接続もそれにともなって矛盾なく修正する

〈3·2〉 GNP の実行–進化の流れ　GNP の進化過程全体のアルゴリズムは次のようになる。

GNP 全体のアルゴリズム

（1） 集団中の全個体の初期化

（2） 各個体毎の適合度の計算

（3） 遺伝的オペレータの適用

（a） 削除（全個体）

（b） 追加（全個体）

（c） 交叉（交叉個体数）

（d） 突然変異（突然変異個体数）

（e） エリート保存（1 個体）

（4） 遺伝的オペレータにより生成された個体を次世代の個体とする

（5） 2–4 を繰り返す（終了条件を満たすまで）

4. シミュレーション環境

〈4·1〉 タイルワールド [18]　提案手法の性能を実証するために，タイルワールドと呼ばれる環境におけるエージェントのプラニング問題においてシミュレーションを行う。タイルワールドは図 4 に示されるように，複数の**エージェント**，**障害物**，**タイル**，**穴**，**床**からなる 2 次元格子状の仮想世界である。エージェントは幾つかのセンサ機能と行動機能を持ち，それを駆使して全てのタイルをできるだけ速く穴に落とすことを目的とする。そのため，限られたセンサ機能と行動機能を，エージェントの状況に応じて上手く使い分けることが要求される。GP においてテストベッド問題として用いられる事の多い「蟻の行動問題」も類似問題と考えられる。蟻の行動問題は GP にとって難問であるとの研究報告があるが [2][19]，タイルワールド問題はセンサの種類，センシング結果の多さから，それに勝るとも劣らない複雑さを含有する。蟻の行動問題では，センサ機能として IF-FOOD-AHEAD というタスクに密接した非終端記号しか用意されていない。我々は，タスクに直接密接していない様々なセンサ機能や行動機能を組合せることにより，汎用性の高い高度な知識を GNP プログラムにより構成することを試みたい。タイルワールド問題では様々なセンサ機能や行動機能が用意されているが，それぞれ単独ではタスク達成に密接した機能としては成り立たず，それらの工夫された組合せによってのみタスク達成に必要な情報の抽出が可能となっている。また，エージェントはタイルワールドの部分的な情報しか知りうることができないため，部分観測マルコフ決定過程問題とみなすことができ，単純な反射行動的プログラムではタスク達成は困難である。このため，暗黙的記憶機能を始めとする判定／行動の一連の流れを重視する GNP の能力検証にふさわしい問題と言えよう。

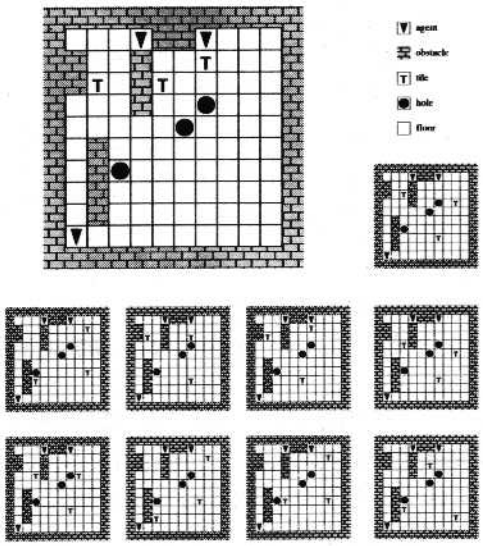

図 4　タイルワールド
Fig. 4.　The tileworlds.

既に，1 種のタイルワールドにおいて高い適合度を持つプログラムの生成は容易に達成できることを確認しており，単純な GP に比して高い収束性を示した [12]。本論文中のシミュレーションでは，複数のタイルワールドに対して高い適合度を持つ，一般性のある高度な知識を表現するプログラムの生成を行う。具体的には，図 4 のようにタイルや穴の位置に若干の変動を加えたタイルワールドを用意した。ここで元となる環境（図 4 左上）は文献 [20] による。

〈4·2〉 ノード関数集合　エージェントは前方のセルへ移動（MF），右に 90° 度回転（TR），左に 90° 度回転（TL），何もしない（ST）の行動機能を持つ。また，エージェントの前方，右方，左方，後方のセルに何が存在するか（それぞれ CF，CR，CL，CB），エージェントから最も近いタイルの方向（NTD），2 番目に近いタイルの方向（SNT），最も近い穴の方向（NHD），エージェントから最も近いタイルから見た最も近い穴への方向（THD）を知ることができる。CF，

表1　ノード関数集合

Table 1.　The node function sets.

Node Type	Node Functions	Arity
Judgement node	CR , CF , CL , CB	5
	NTD, NHD, THD, SNT	4
Processing node	MF , TR , TL , ST	1
Intron node	IN	1

CR, CL, CBは5種類の判定結果（エージェント，障害物，タイル，穴，床）を，NTD, SNT, NHD, THDは4種類の判定結果（前，後，左，右）を持つ。表1にシミュレーションで使用するノード関数集合と引数の数（弧の本数）を示す。

〈4·3〉 適合度の評価方法　それぞれのエージェントは同一のGNPプログラムを基準に行動を行うが，エージェント毎に活性化状態の遷移系列は異なる。エージェントは交互に動作を繰り返し，その動作単位時間を**1ステップ**と定義する。各エージェントは各々の行動順番において，GNPプログラムに従い様々な判定／行動を行う。判定／行動は2章で述べたように，GNPプログラム上の活性化状態の流れに従い決定される。その間，活性化状態の遷移毎に遅れ時間が加算され，**内部閾値**を超えた時点で1ステップの終了とする。全てのタイルが穴に落とされるか，エージェントの行動が**最大ステップ数**に達した後に適合度の評価を行う。なお，以降のシミュレーションでは最大ステップ数を60としている。

適合度は（1）落としたタイルの数,（2）タイルを穴にどれだけ近づけたか,（3）全てのタイルを落とすのに何ステップ要したかを考慮し，式（5）で計算される。

$$fitness = \sum_{t \in TW} (100 \times DT_t + 20 \times AD_t + 1 \times RS_t) \cdots\cdots (5)$$

ここで，TWは使用した環境（タイルワールド）の添字の集合，DT_t，AD_t，RS_tは，それぞれ環境tにおけるエージェントが落としたタイルの数，タイルを穴に近づけた距離，タスク完了時に余ったステップ数（タスク未完了時は0）である。

〈4·4〉 実行条件　次章では表2の条件下で提案手法の検討を行う。遅れ時間と内部閾値の設定は，エージェントが1ステップ内に行える判定／処理を制限している。表2の設定（d_i, d_{ij}，内部閾値）では，エージェントは1ステップ内に最大で5つの判定を行うか，4つの判定と1つの行動を行うことができる。エージェントが1ステップ内に知覚できる判定結果は，用意した判定ノードの種類に比べて高々半数程度であるため，エージェントが1ステップ内に行動するためには，必要最低限の判定に止めることが要求される。実験1は提案手法の性能を追加／削除の基準において一切のランダム性を排除した状態で検証し，実験2はノード追加の際の基準にランダム性を加え，実験3では削除の際の基準にもランダム性を加えたものである。

表2　実行条件

Table 2.　Parameter conditions.

最大世代数 = 5000	
集団数 = 301，交叉個体数 = 120，突然変異個体数 = 180 交叉確率 $P_c = 0.1$，突然変異確率 $P_m = 0.01$	
$N^{ini} \in \{1, 5, 10, 30\}$，$N^{min} = 1$，$N^{max} = 30$	
$d_i = 1$（判定ノード），$d_i = 5$（処理ノード） $d_i = 0$（イントロンノード），$d_{ij} = 0$（全ノード間） 内部閾値 = 5	
実験1	$a = \infty$（追加オペレータ），$b = \infty$（削除オペレータ） $P_{max}^{add} \in \{0.01, 0.05, 0.1\}$，$P_{max}^{del} \in \{0.001, 0.005, 0.01\}$
実験2	$a = 3.0$（追加オペレータ），$b = \infty$（削除オペレータ） $P_{max}^{add} \in \{0.05\}$，$P_{max}^{del} \in \{0.001, 0.005, 0.01\}$
実験3	$a = 3.0$（追加オペレータ），$b = 0.5$（削除オペレータ） $P_{max}^{add} \in \{0.05\}$，$P_{max}^{del} \in \{0.001, 0.005, 0.01\}$

5. シミュレーション結果

〈5·1〉 実 験 1　表3に従来手法および実験1の条件で行ったシミュレーション結果を示す。表中のMin，Ave，Maxはそれぞれ10種の独立した試行の最大世代数における最良適合度の最小値，平均値，最大値を表し，s/fはタスク（全てのタイルワールドにおいて全てのタイルを落とす）の成功／失敗回数，AvePSは最良個体の平均プログラムサイズである。表中の太字の数字は，ノード数固定型GNPと比較した際に良好な結果を得たことを示す。また，図5は表3中の最良適合度の最大値（Max）をグラフで比較したものであり，図中の太線は提案手法を用いなかった場合の最良適合度の最大値を，細線は各パラメータ（$P_{max}^{add}, P_{max}^{del}$）毎の結果を表している。

得られた結果は提案手法がもたらす明らかな改善を支持するものであると言える。特に$N^{ini} = 30$の場合はほとんど全ての場合において飛躍的な改善が見られた。唯一，$N^{ini} = 10$の場合は悪い結果とは言わないまでも，ノード数固定型GNPと比較して若干劣る傾向にある。これは，元々適切な値に近い状態にある初期ノード数を提案手法により増減したために，進化の面で若干の不利を被る形となったためと推測される。特に，プログラムが十分に進化していない初期世代におけるノード数の増減は，理に適っていないものである可能性が高く，結果的に$N^{ini} = 10$の場合は進化を妨げる方向に働いたのであろう。即ち，表現能力の向上による利点を受けないようなノードの増加が行われ，その結果，進化効率の減少による不利のみが表出したと考えられる。$N^{ini} = 30$の場合はこれ以上ノード数を増やすことができないため，初期状態におけるノード数の増減は概ね削除オペレータによるものが中心となり，たとえ理に適っていない削除が行われたとしても進化の弊害にはなりにくかったと思われる。また，$N^{ini} = 1, 5$の場合は初期ノードの少なさから，ノード数の増減は概ね追加オペレータによるものが主流だったようである。このため，たとえ理に適っていない追加が行われたとしても，表現能力の向上による利点と進化効率の減少による不利では表現能力の

表 3　実験 1：シミュレーション結果

Table 3.　Simulation 1: Simulation results.

	$P_{max}^{del}=0.001$					$P_{max}^{del}=0.005$					$P_{max}^{del}=0.01$				
	Min	Ave	Max	s/f	AvePS	Min	Ave	Max	s/f	AvePS	Min	Ave	Max	s/f	AvePS
$N^{ini}=1$ (Min = 2380, Ave = 3419.6, Max = 5401, s/f = 0/10, Program Size =13 when $P_{max}^{add}=P_{max}^{del}=0$)															
$P_{max}^{add}=0.01$	**3335**	**4275.8**	**5486**	**2/8**	**89.9**	**2700**	**3608.5**	4901	0/10	**37.5**	2142	**3943.0**	**5596**	**1/9**	**29.3**
$P_{max}^{add}=0.05$	**4055**	**4900.1**	**5590**	**4/6**	**163.8**	**3863**	**4593.7**	**5688**	**1/9**	**93.6**	**3532**	**4476.2**	**5840**	**1/9**	**58.6**
$P_{max}^{add}=0.1$	**4485**	**5011.3**	**5435**	**1/9**	**189.1**	**3579**	**5035.2**	**5720**	**4/6**	**145.5**	**3282**	**4515.6**	5368	0/10	**95.1**
$N^{ini}=5$ (Min = 3936, Ave = 4380.6, Max = 5264 s/f = 0/10, Program Size =66 when $P_{max}^{add}=P_{max}^{del}=0$)															
$P_{max}^{add}=0.01$	3476	**4638.2**	**5679**	**2/8**	**106.3**	3381	**4864.1**	**5862**	**1/9**	55.5	3243	4247.1	**5488**	**1/9**	43.9
$P_{max}^{add}=0.05$	3406	**4604.8**	**5915**	**2/8**	**195.1**	3543	**4537.8**	**5555**	0/10	**112.8**	2875	4210.8	**5734**	**1/9**	**73.5**
$P_{max}^{add}=0.1$	3756	**4665.7**	**5522**	0/10	**225.4**	**3998**	**5011.3**	**5787**	**3/7**	**162.4**	**4151**	**4742.3**	**5489**	**2/8**	**110.0**
$N^{ini}=10$ (Min = 3665, Ave = 4646.5, Max = 5592 s/f = 2/8, Program Size =131 when $P_{max}^{add}=P_{max}^{del}=0$)															
$P_{max}^{add}=0.01$	3331	4383.7	5406	0/10	**135.3**	3360	4361.1	5256	0/10	75.7	**3907**	**4975.7**	**5736**	2/8	62.4
$P_{max}^{add}=0.05$	**3753**	4522.0	5385	1/9	**224.9**	2480	4450.9	5406	1/9	126.0	2720	4330.8	5188	0/10	85.1
$P_{max}^{add}=0.1$	2640	4577.3	5552	1/9	**218.6**	3386	4632.0	5466	1/9	**172.9**	3636	**4724.5**	**5748**	1/9	123.2
$N^{ini}=30$ (Min = 3550, Ave = 4664.0, Max = 5303 s/f = 0/10, Program Size =391 when $P_{max}^{add}=P_{max}^{del}=0$)															
$P_{max}^{add}=0.01$	**4327**	**4921.2**	**5671**	**2/8**	185.0	**4259**	**5203.8**	**5914**	**4/6**	93.7	**3723**	**4903.0**	**5559**	**3/7**	98.9
$P_{max}^{add}=0.05$	**3668**	4537.1	**5372**	**1/9**	237.4	**4576**	**5141.8**	**5492**	**2/8**	161.4	**4388**	**5198.8**	**5732**	**4/6**	99.0
$P_{max}^{add}=0.1$	**4383**	**5087.8**	**5677**	**3/7**	256.3	**3583**	**4898.4**	**5751**	**2/8**	184.9	**4184**	**5020.6**	**5628**	**4/6**	145.5

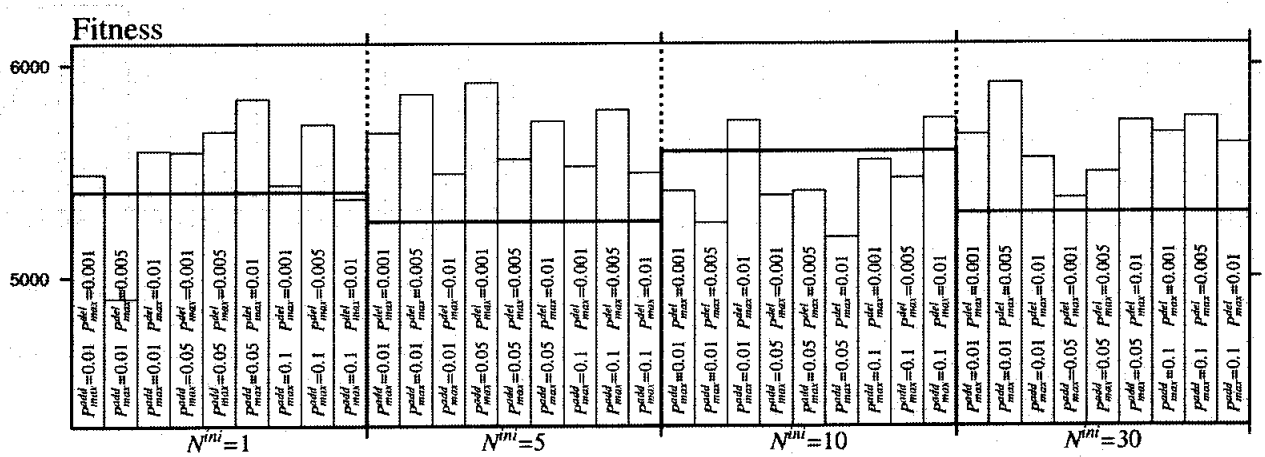

図 5　実験 1: 10 回の試行における最良適合度の最大値（表 3 中の Max）

Fig. 5.　Simulation 1: The maximum value of the best fitness over ten runs in table 3.

向上による利点の方が強く表れた結果になったものと思われる。

また，プログラムサイズに関しては，P_{max}^{add} および N^{ini} に対する単調増加性，P_{max}^{del} に対する単調減少性がはっきりと見られる。この結果から，真に最適なプログラムサイズの獲得には P_{max}^{add}, P_{max}^{del} および N^{ini} の調整がある程度は必要であることになる。しかし，極端な値の P_{max}^{add} と P_{max}^{del} の組合せを除けば，平均プログラムサイズが明らかな異常値に定着することは少ないようである。

なお，全般的に，MF，NTD，NHD のノード数は多く，SNT，ST は少なくなる傾向が見られる。厳密に全てのノード数 N_k の大小関係およびその順序が毎回同一になるわけではないが，上記のような典型的な傾向を適応的に獲得したことは明らかであり，提案手法が主張する適合度への貢献度を基準とした適切なノード数の増減が行われたと考えてよい。

〈5·2〉 実 験 2　表 4 に従来手法および実験 2 の条件（実験 1 の条件に，追加されるノード関数の選択に対して若干のランダム性を付加）における結果を，図 6 に最良適合度の最大値（Max）をグラフで比較したものを示す。表，図中の表現は実験 1 と同様である。全体としてはノード数固定型 GNP よりも良好な結果を維持しているものの，実験 1 に比べ $N^{ini}=30$ では，タスク成功回数（s/f）において見劣りする結果となった。全体的に実験 1 に比して平均プログラムサイズが増加する傾向にあり，弱い削除圧力（P_{max}^{del}=0.001）においてはプログラムサイズの発散傾向を押さえることができていない。これらの原因は活用度

表 4　実験 2：シミュレーション結果

Table 4.　Simulation 2: Simulation results.

	$P_{max}^{del}=0.001$					$P_{max}^{del}=0.005$					$P_{max}^{del}=0.01$				
	Min	Ave	Max	s/f	AvePS	Min	Ave	Max	s/f	AvePS	Min	Ave	Max	s/f	AvePS
$N^{ini}=1$ (Min = 2380, Ave = 3419.6, Max = 5401, s/f = 0/10, Program Size =13 when $P_{max}^{add}=P_{max}^{del}=0$)															
$P_{max}^{add}=0.05$	**4163**	**4878.6**	**5779**	**1/9**	**290.9**	**3913**	**4688.4**	**5437**	**1/9**	**109.4**	**4445**	**5043.8**	**5494**	**2/8**	**76.4**
$N^{ini}=5$ (Min = 3936, Ave = 4380.6, Max = 5264 s/f = 0/10, Program Size =66 when $P_{max}^{add}=P_{max}^{del}=0$)															
$P_{max}^{add}=0.05$	**4042**	**4596.5**	**5729**	**1/9**	**282.4**	**4325**	**4903.4**	**5632**	**2/8**	**145.2**	3234	4282.3	**5488**	**2/8**	**89.3**
$N^{ini}=10$ (Min = 3665, Ave = 4646.5, Max = 5592 s/f = 2/8, Program Size =131 when $P_{max}^{add}=P_{max}^{del}=0$)															
$P_{max}^{add}=0.05$	3470	4189.4	4731	0/10	**320.6**	**3888**	**4708.9**	**5757**	2/8	**150.9**	2840	**4649.6**	5558	0/10	102.9
$N^{ini}=30$ (Min = 3550, Ave = 4664.0, Max = 5303 s/f = 0/10, Program Size =391 when $P_{max}^{add}=P_{max}^{del}=0$)															
$P_{max}^{add}=0.05$	**3828**	4623.6	**5426**	0/10	344.5	**3781**	4568.6	**5789**	**2/8**	188.1	**3746**	**4714.9**	**5552**	**1/9**	127.3

α_k が 1.0 より低い値でもノードを追加したことが原因と考えられ，ノードの追加においてはランダム性はできるだけ排除し，真に必要なノードを加えた方が良好な結果が得られることを示唆している。この結果から明らかとなったランダム性の付加による進化能力の劣化は，提案手法で用いる追加オペレータの選択基準となる指標（α_k）が合理的であることの実験的証明となる。

〈5·3〉 実 験 3　表 5 に従来手法および実験 3 の条件（実験 2 の条件に，削除されるノードの選択基準に若干のランダム性を付加）における結果を示す†。また，図 7 に最良適合度の最大値（Max）をグラフで比較したものを示す。実験 3 では，適合度に貢献しているノードも削除する可能性があるため，適合度曲線は世代数に対して単調に増加しない。このため，進化の面で若干の不利を受けるように感じる。しかし予想に反して，実験 2 よりも良好な結果，実験 1 にほぼ匹敵する結果が得られた。特に，得られた最良適合度の最大値が全体的に高い値を得る結果となった。プログラムサイズに関しては，N^{ini} による影響が実験 1，実験 2 に比べてやや弱い傾向にあるといえる。しかし，依然として $P_{max}^{del}=0.001$ におけるプログラムサイズの発散傾向を押さえることはできていない。

実験 3 における典型的な適合度曲線††を図 8 に示す。全体の傾向として，適合度にわずかながらも貢献しているノードの削除により適合度の減少が起きるが，再び元の適合度に達するために要する世代数はそれまでの進化過程で要した世代に比べて明らかに短いことがわかる。むしろ，削除後はさらなる適合度の上昇がみられ，実験 3 の最良適合度が比較的高い理由はこの点にあると思われる。GNP プログラムにおいて，使用頻度の少ないノードは，例外処理部分と考えることができる。図 8 に見られる現象は，例外処理部を取り除き，より一般的なプログラムへの進化の助長を行っていることに相当すると思われる。

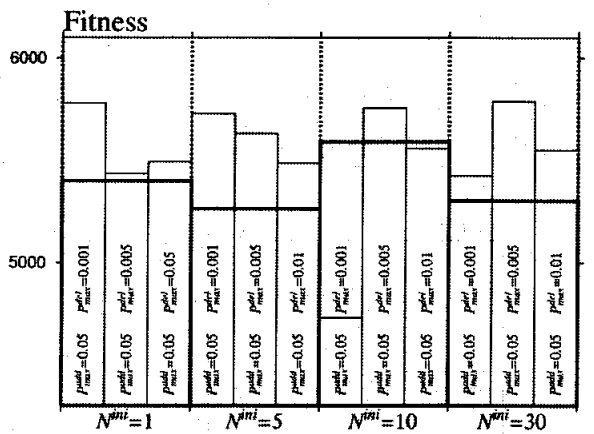

図 6　実験 2: 10 回の試行における最良適合度の最大値（表 4 中の Max）

Fig. 6.　Simulation 2: The maximum value of the best fitness over ten runs in table 4.

6. 結　　論

有向グラフにより表現されるプログラムの進化を行う進化論的計算手法 GNP のプログラムサイズを，ノードの適合度への貢献度（活用度 α_d，活性化回数 x_i）を基準に制御する方式を提案した。提案手法は新たに導入された追加オペレータと削除オペレータにより実現され，シミュレーション結果は提案手法の有効性を明確に支持するものであった。また，適合度に影響のあるノードを取り除いても，その活性化回数が少ないならば，より高い適合度への進化を助長する場合があることを明らかにした。一方，プログラムサイズは常に最適な値へ収束するとは限らず，事実上，パラメータ（$P_{max}^{add}, P_{max}^{del}, N^{ini}$）に大きく依存するため，これらの妥協点を探す必要性から完全に解放された訳ではない。

†表中の値は最大世代数における値であるため，実験 3 では進化過程において獲得された真の最良適合度を必ずしも表していない。

††上から $\{N^{ini}=10,\ P_{max}^{add}=0.05,\ P_{max}^{del}=0.005\}$, $\{N^{ini}=10,\ P_{max}^{add}=0.05,\ P_{max}^{del}=0.001\}$, $\{N^{ini}=30,\ P_{max}^{add}=0.05,\ P_{max}^{del}=0.001\}$ における最良適合度の変化例。

表 5　実験 3：シミュレーション結果

Table 5.　Simulation 3: Simulation results.

	$P_{max}^{del}=0.001$					$P_{max}^{del}=0.005$					$P_{max}^{del}=0.01$				
	Min	Ave	Max	s/f	AvePS	Min	Ave	Max	s/f	AvePS	Min	Ave	Max	s/f	AvePS
$N^{ini}=1$　(Min = 2380, Ave = 3419.6, Max = 5401, s/f = 0/10, Program Size =13 when $P_{max}^{add}=P_{max}^{del}=0$)															
$P_{max}^{add}=0.05$	**3878**	**4637.6**	**5696**	**3/7**	**261.5**	**3646**	**4668.1**	**5800**	**2/8**	**97.6**	**2760**	**4317.0**	**5747**	**1/9**	**62.3**
$N^{ini}=5$　(Min = 3936, Ave = 4380.6, Max = 5264 s/f = 0/10, Program Size =66 when $P_{max}^{add}=P_{max}^{del}=0$)															
$P_{max}^{add}=0.05$	3440	**4480.3**	**5537**	**1/9**	**277.0**	**4013**	**1656.1**	**5268**	0/10	**119.7**	3392	4374.3	**5657**	0/10	**74.8**
$N^{ini}=10$　(Min = 3665, Ave = 4646.5, Max = 5592 s/f = 2/8, Program Size =131 when $P_{max}^{add}=P_{max}^{del}=0$)															
$P_{max}^{add}=0.05$	3638	**4756.3**	**5629**	1/9	**300.3**	**3941**	**4818.6**	**5830**	2/8	118.6	3663	**4877.2**	**5875**	1/9	78.1
$N^{ini}=30$　(Min = 3550, Ave = 4664.0, Max = 5303 s/f = 0/10, Program Size =391 when $P_{max}^{add}=P_{max}^{del}=0$)															
$P_{max}^{add}=0.05$	**4256**	**4859.7**	**5636**	**2/8**	320.4	**3834**	4606.5	**5585**	**1/9**	131.5	**3700**	**4927.3**	**6016**	**3/7**	81.4

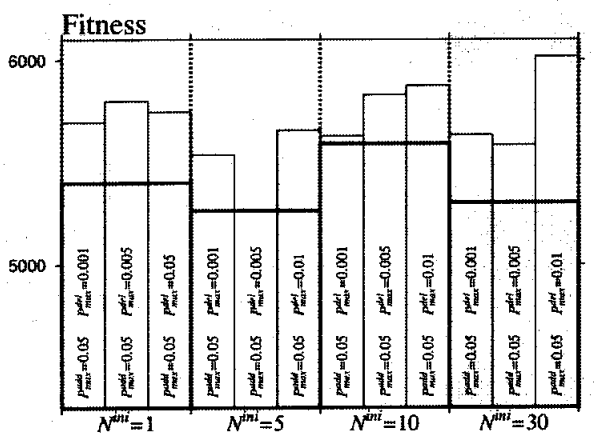

図 7　実験 3: 10 回の試行における最良適合度の最大値（表 5 中の Max）

Fig. 7.　Simulation 3: The maximum value of the best fitness over ten runs in table 5.

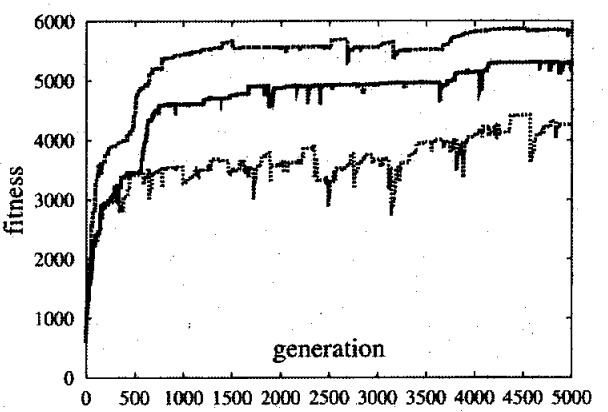

図 8　実験 3：典型的な適合度曲線の例

Fig. 8.　Simulation 3: Typical examples of the fitness curve

しかし，妥協点はそれほど厳密ではなく，さらにどのノード関数が必要とされ，どのノード関数が必要でないかいう点に関しては提案手法により十分対処することができる。以上のことから，提案手法は GNP プログラムの進化において効果的な手法であると結論づける事ができる。

（平成 14 年 3 月 4 日受付，同 14 年 6 月 12 日再受付）

文　献

（1）John R. Koza: *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, (1992)

（2）John R. Koza: *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, Cambridge Massachusetts, May (1994)

（3）A. Teller and M. Veloso: PADO: "A new learning architecture for object recognition. In Katsushi Ikeuchi and Manuela Veloso", editors, *Symbolic Visual Learning*, pp. 81–116. Oxford University Press, (1996)

（4）Riccardo Poli: Parallel distributed genetic programming. Technical Report CSRP-96-15, School of Computer Science, University of Birmingham, B15 2TT, UK, September (1996)

（5）J.F. Miller and P. Thomson: Cartesian genetic programming. In Riccardo Poli, Wolfgang Banzhaf, William B. Langdon, Julian F. Miller, Peter Nordin, and Terence C. Fogarty, editors, *Genetic Programming, Proceedings of EuroGP'2000*, Vol. 1802 of *LNCS*, pp. 121–132, Edinburgh, 15-16 April 2000. Springer-Verlag.

（6）F. Gruau: "Neural Network Synthesis using Cellular Encoding and the Genetic Algorithm", PhD thesis, Laboratoire de l'Informatique du Parallilisme, Ecole Normale Supirieure de Lyon, France, (1994)

（7）S. Luke and L. Spector: "Evolving graphs and networks with edge encoding": Preliminary report. In John R. Koza, editor, *Late Breaking Papers at the Genetic Programming 1996 Conference Stanford University July 28-31, 1996*, pp. 117–124, Stanford University, CA, USA, 28–31 July (1996) Stanford Bookstore.

（8）L.J. Fogel, A.J. Owens, and M.J. Walsh: "Artificial Intelligence through simulated Evolution", John Wiley & Sons, (1966)

（9）D.B. Fogel: "An introduction to simulated evolutionary optimization", *IEEE Transactions on Neural Networks*, Vol. 5, No. 1, pp. 3–14, January (1994)

（10）P. J. Angeline and J. B. Pollack: "Evolutionary module acquisition", In D. Fogel and W. Atmar, editors, *Proceedings of the Second Annual Conference on Evolutionary Programming*, pp. 154–163, La Jolla, CA, USA, 25-26 (1993)

（11）H. Katagiri, K. Hirasawa, and J. Hu: "Genetic network programming-application to intelligent agents-", In *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*, pp. 3829–3834 (2000)

（12）H. Katagiri, K. Hirasawa, J. Hu, and J. Murata: "Net-

work structure oriented evolutionary model-genetic network programming-and its comparison with genetic programming", In Erik D. Goodman, editor, *2001 Genetic and Evolutionary Computation Conference Late Breaking Papers*, pp. 219–226, San Francisco, California, USA, 9-11 July (2001)

(13) K. Hirasawa, M. Okubo, J. Hu, and J. Murata: "Comparison between genetic network programming (GNP) and genetic programming (GP)", In *Proceedings of CEC International Conference*, Vol. 2, pp. 1276–1282 (2001)

(14) 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-C, No. 6, pp. 1001–1009 (2001-6) (in Japanese)
平澤宏太郎・大久保雅文・片桐広伸・胡　敬炉・村田純一：「蟻の行動の進化における genetic network programming と genetic programming の性能比較」, 電学論誌, **121-C**, 6, pp. 1001–1009 (2001-6)

(15) S. Mabu, K. Hirasawa, J. Hu, and J. Murata: "Online Learning of Genetic Network Programming", *T. IEE Japan*, Vol. 122-C, No. 3, pp. 355–362 (2002-3) (in Japanese)
間普真吾・平澤宏太郎・胡　敬炉・村田純一：「遺伝的ネットワークプログラミングのオンライン学習」, 電学論誌, **122-C**, 3, pp. 355–362 (2002-3)

(16) T. Soule, J.A. Foster, and J. Dickinson: "Code growth in genetic programming", In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pp. 215–223, Stanford University, CA, USA, 28–31 (1996) MIT Press.

(17) F.B. Pereira, P. Machado, E. Costa, and A. Cardoso: "Graph based crossover-A case study with the busy beaver problem", In Wolfgang Banzhaf, Jason Daida, Agoston E. Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela, and Robert E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, Vol. 2, pp. 1149–1155, Orlando, Florida, USA, 13-17 July (1999) Morgan Kaufmann.

(18) M. E. Pollack and M. Ringuette: "Introducing the tile-world: Experimentally evaluating agent architectures", In *Proceedings of the Conference of the American Association for Artificial Intelligence*, pp. 183–189 (1990)

(19) W. B. Langdon and R. Poli: "Why ants are hard. In John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, David E. Goldberg, Hitoshi Iba, and Rick Riolo, editors", *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pp. 193–201, University of Wisconsin, Madison, Wisconsin, USA, 22-25 July (1998) Morgan Kaufmann.

(20) C.V. Goldman and J.S. Rosenschein: "Emergent coordination through the use of cooperative state-changing rules", In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pp. 408–413, Seattle, WA, (1994)

片　桐　広　伸

（学生員）1976 年 2 月 21 日生。1998 年 3 月九州大学工学部電気工学科卒業。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。1966 年 3 月九州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。電気学会，計測自動制御学会，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。