適応型 GP-オートマトンによるエージェントの行動制御

片岡 寛明[†,☆] 原 章[†,☆☆] 長尾 智晴[††]

エージェントの行動制御において，同一の入力に対して場合に応じて異なる出力をとる必要がある問題に対しては，エージェントの内部状態を考慮し，その状態の違いに応じて行動を選択する必要がある．遺伝的プログラミング（Genetic Programming; GP）を用いたエージェントの行動制御において，これを実現するため，Ashlock は GP-オートマトン（GP-Automata）を提案している．GP-オートマトンではエージェントが利用できる内部状態数を固定して問題に適用する．あらかじめ設定した状態数が適切ならば問題を解くことができるが，状態数が足りなければ問題を解くことはできず，多すぎれば冗長な部分が増え最適化が困難になる．すなわち，GP-オートマトンの性能は固定した状態数に左右されることになる．そこで，本論文では進化過程で，扱う状態数を増減させる適応型 GP-オートマトン（Adaptive GP-Automata; AGPA）を提案する．本手法は問題解決に必要な状態数を進化により自動的に獲得するため，未知の環境に対しても状態数に対する試行錯誤を必要としない．AGPA の性能を検証するため迷路探索問題を設定し実験を行った結果，要求される状態数が異なる様々な迷路に対して最適解を得ることに成功した．また，従来の GP-オートマトンとの比較を行い，提案手法の有効性を示した．

Action Control of an Agent Using Adaptive GP-Automata

Hiroaki Kataoka,[†,☆] Akira Hara[†,☆☆] and Tomoharu Nagao[††]

Recently, many researches on action control of an agent by Genetic Programming have been made. There are problems such as maze problems, which require different actions even if the agent gets the same inputs from its sensors. In order to solve these problems, the agent has to select its action according to the difference of its internal states. As the solution, GP-Automata have been proposed by Ashlock. In GP-Automata, the number of internal states in an agent is fixed beforehand. If the number is appropriate, the agent can solve the problem. If the number is smaller than the required number, however, the agent can not solve the problem. On the other hand, if the the number is larger than the required number, the search efficiency declines. That is, the performance of GP-Automata depends on the number of the internal states prepared beforehand. Therefore, we propose Adaptive GP-Automata (AGPA), in which the number of the internal states is optimized in evolutionary process. This method does not need trial and error for decision of the number of the internal states. We showed experimentally that AGPA are more effective for perceptual aliasing problems in comparison with GP-Automata.

1. はじめに

近年，木構造プログラムを進化的に獲得する手法である遺伝的プログラミング（Genetic Programming; GP）[1]を用いて，知的なエージェントやマルチエージェントシステムの行動制御を行う研究が注目されている[2)~10)]．GP を適用する場合は，各問題に応じて，終端・非終端記号を設定しなければならない．同一の入力に対して，状況に応じて異なる出力をすることが求められる不完全知覚問題に対しては，エージェントの内部状態を考慮し，同一の感覚入力に対しても内部状態の違いにより適切な行動を選択することが必要となる[11]．これを実現するためには，状況に応じた内部状態の設定と，現在どの状態にあるかを判断可能な非終端記号の用意をあらかじめ行わなければならない．しかし，問題が複雑になると，あらかじめ適切な状態設定を知ることは困難になる．そこで，このよう

† 東京工業大学大学院総合理工学研究科物理情報システム創造専攻
Department of Information Processing, Tokyo Institute of Technology
†† 横浜国立大学大学院環境情報研究院
Faculty of Environment and Information Sciences, Yokohama National University
☆ 現在，日本テレコム株式会社
Presently with JAPAN TELECOM Co., Ltd.
☆☆ 現在，広島市立大学情報科学部知能情報システム工学科
Presently with Faculty of Information Sciences, Hiroshima City University

な問題の解決策として Ashlock は GP-オートマトン（GP-Automata）[12]という手法を提案した．GP-オートマトンでは，利用できる状態数をあらかじめ定め，各状態での行動用に別々の木構造プログラムを利用する．あらかじめ設定した状態数が適切ならば問題を解くことができるが，状態数が足りなければ問題の解は得られず，多すぎれば冗長な部分が増え探索領域が膨大になり最適化が困難になる．そのため，GP-オートマトンの性能はあらかじめ設定した状態数に左右されることになる．そこで，本論文では状態数をあらかじめ設定するのではなく，利用する状態数を進化過程で増減させる適応型 GP-オートマトン（Adaptive GP-Automata; AGPA）を提案する．本手法は問題を解くために必要な状態数を進化過程で自動的に獲得するため，未知の環境に対しても状態数に対する試行錯誤を必要としない．本論文では，迷路探索問題に AGPA を適用し，従来の GP-オートマトンとの比較を行った．

本論文の構成は以下のとおりである．2 章では，不完全知覚問題に対する従来研究である GP-オートマトンについて述べる．3 章では，本論文で提案する AGPA について述べる．4 章では，今回実験を行う迷路探索問題の設定を行い，5 章で GP-オートマトンとの比較実験により得られた結果に対して考察を行う．6 章では，まとめと今後の課題について述べる．

2. GP を用いたエージェントの行動制御に関する従来研究

2.1 不完全知覚問題

同一の感覚入力に対して異なる行動をとる必要がある問題として，次のようなエージェントによる迷路探索問題があげられる[11]．

エージェントへの入力は上下左右の壁の有無によって与えられ，出力は上下左右のいずれかへの進行方向とする．エージェントがスタート地点からゴール地点までの最短路を進むことがここで与えられたタスクである．このような環境におけるエージェントの行動制御プログラムの生成に GP を適用する場合，非終端記号として上下左右の壁の有無による分岐，終端記号として上下左右いずれかの進行方向，という記号を設定する．しかし，この設定では**図 1** のような単純な迷路ですら解くことはできない．

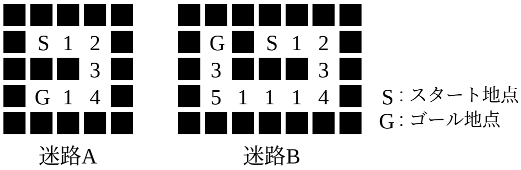


図 1　迷路の例
Fig. 1　Examples of maze.

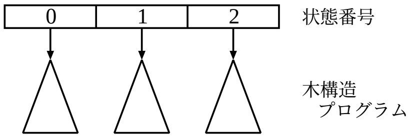


図 2　GP-オートマトンの個体構造の例
Fig. 2　An individual of GP-Automata.

なぜならば図 1 の迷路 A では，1 の場所においてエージェントに上下に壁があるという入力が与えられる．エージェントは S の右側の 1 では右へ，G の右側の 1 では左に移動しなくてはならない．しかし，今の設定では，上下に壁があるという入力を与えられたとき，エージェントは S の右側の 1 なのか G の右側の 1 なのかを識別できないためこの迷路を解くことはできない．GP でこの迷路を解くためには，S の右側の 1 なのか G の右側の 1 なのかを内部状態で区別し，それに基づいて状況を識別する非終端記号を設定しなければならない．図 1 の迷路 B においても，1，3 の場所でエージェントは同一の入力に対して適切な進行方向を出力しなければならない．そのため，適切な内部状態設定，およびそれらを利用した非終端記号を新たに設定する必要がある．例にあげたような単純な迷路ならば少しの設定の追加で解くことはできるが，迷路が複雑になれば，必要な状況設定およびそれに基づく非終端記号をすべて設定することは困難である．したがって，このような問題へは通常の GP は適用できない．

2.2 GP-オートマトン

前節であげたような問題を GP で解くために，Ashlock は GP-オートマトン[12]という手法を提案した．GP-オートマトンは，有限オートマトンと GP を組み合わせたものである．GP-オートマトンではあらかじめ有限の状態数を設定し，その各状態番号ごとに GP の木を作る．これが GP における 1 個体となる．**図 2** に状態数 3 の個体構造を示す．

木構造プログラムの終端記号は，行動とそれによって遷移する次の状態番号の組となる．GP-オートマトンでは，現在の状態番号に対応する GP 木を参照し行動を決定する．そして状態遷移を行い，次ステップでは新たな状態番号に対応する GP 木を参照する，という動作を繰り返す．全体の処理の流れは GP と同じである．前節の迷路 1 に状態数 2 の GP-オートマトンを適用すると，S の右側の 1 の箇所では初期状態のプ

表 1 内部状態に対する操作
Table 1 Operations to internal states.

終端記号	状態番号に対する操作
Inc	現在の状態番号を 1 増やす
Dec	現在の状態番号を 1 減らす（ただし 0 に対しては操作を行わない）
Stay	現在の状態番号を維持する
Set_i	定義されている状態番号 i に遷移する

表 2 内部状態の使用例
Table 2 An example of the use of internal states.

状態番号	使用回数
0	5
1	3
2	1
3	1
4	3

ログラムを参照し，その後状態遷移を行い G の右側の 1 の箇所では別のプログラムによる行動をとれば問題解決が可能である．

3. 適応型 GP-オートマトン（AGPA）

3.1 GP-オートマトンの問題点

GP-オートマトンでは用いる状態数をあらかじめ設定して問題に適用する．はじめに設定した状態数が問題を解くために必要な状態数に満たなければ，問題を解くことはできない．一般に問題が複雑になると，解くために必要な状態数を事前に知ることはできないため，あらかじめ適切な状態数を設定することは困難になる．このため GP-オートマトンでは十分と思われる状態数を設定することになるが，この値が適切な値より大きくなりすぎると冗長な部分が増え最適化が困難になる．GP-オートマトンの性能はこの固定する状態数に左右されることになり，問題ごとに適切な状態数を設定するために試行錯誤を繰り返すことになる．

3.2 適応型 GP-オートマトン（AGPA）

本論文では，前節で述べた問題点を解決するため，状態数をあらかじめ固定するのではなく，扱う状態数を進化過程で増減させる手法である適応型 GP-オートマトンを提案する．本手法は，問題を解くために必要な状態数を進化過程で自動的に獲得するため，未知の環境に対しても状態数に対する試行錯誤を必要としない．

本手法の初期個体は初期状態番号のときに参照する木と，それ以外の状態番号のときに参照する木（Default 木）の 2 つから構成される．初期状態番号はつねに 0 とする．木構造の葉に相当する部分は現在の状態番号に対する操作と行動の組となる．状態番号に対する操作を**表 1** に示す．参照する木がない状態番号を未定義の状態番号と呼び，参照する木を持つ状態番号を定義されている状態番号と呼ぶ．実行中，未定義の状態番号は Default 木を参照する．

AGPA における遺伝操作は，選択，状態数の増減，交叉，突然変異の順で行われる．状態数の増減方法については，次節で述べる．交叉は，個体集団の中から順に親個体として 2 個体ずつを選び，あらかじめ定められた交叉率により，交叉を行うかどうかを決定する．交叉方法は 1 点交叉であり，交叉点は，個体が持つ全ノードの中からランダムに選択し，選択されたノード以下の部分木を交換する．特定の状態番号の木どうしに交叉を制限することは行わない．また，突然変異は，木の各ノードが突然変異を起こすかどうかを，あらかじめ定められた突然変異率により決定する．

なお，状態操作記号とエージェントの行動の組を木構造の出力とする必要があるため，たとえば，状態操作記号がエージェントの行動記号に置き換わるとその木構造は致死遺伝子となり実行できない．このようなことを防ぐため，AGPA では，交叉や突然変異において，致死遺伝子を生成しないように交換可能なノードを制限する．この具体例については，4.3.2 項の迷路探索問題における遺伝操作の設定で述べる．

3.3 状態数の増減

3.3.1 状態数の増加

状態番号に対する操作に Inc，Dec という終端記号があるため，ある個体の実行中に，未定義の状態番号を使用することがある．このような場合，この個体が世代交代を行うときに状態数の増加が起こる．新たに定義される状態番号は，実行中に使用した未定義の状態番号の中からランダムに 1 つ選択される．その内部状態で参照する木は，すでに定義されている木の中からランダムに選ばれた木の終端記号を変えたものとする．このような木の設定方法を用いた理由は，同じ条件分岐でありながら終端記号が異なる木を持つことで，状態番号の違いによって同一の入力に対して異なる出力をさせることができるためである．

状態数が増加する様子の例を次に示す．状態番号 0 と Default からなる個体が，ある世代で**表 2** のような実行を行ったとする．この個体が世代交代を行うとき，未定義の状態番号 1，2，3，4 の中から新たに 1 つが定義され状態数が増加する．増加する状態番号として，状態番号 3 が選択されたとすると，個体は**図 3** のように構造が変化する．

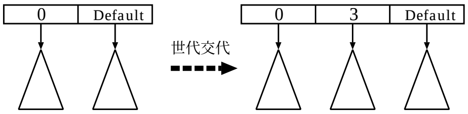


図 3　状態数が増加する様子の例
Fig. 3　An example of increment of internal states.

表 3　内部状態の使用例（その 2）
Table 3　An example of the use of internal states (Part 2).

状態番号	使用回数
0	7
1	0
4	0
5	6

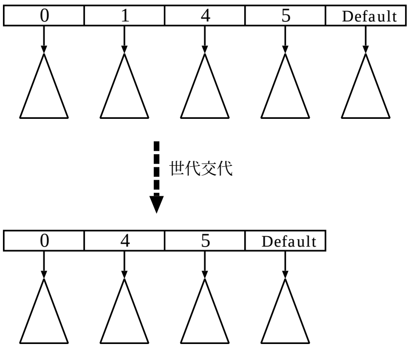


図 4　状態数が減少する様子の例
Fig. 4　An example of decrement of internal states.

3.3.2　状態数の減少

ある個体の実行中に定義されている状態番号の中で 1 度も使われなかった番号がある場合，この個体が世代交代を行うときに状態数の減少が起こる．削除される状態番号は，実行中に一度も使われなかった状態番号の中からランダムに 1 つ選択される．

状態数が減少する様子の例を次に示す．状態番号 0，1，4，5 と Default からなる個体が，ある世代で**表 3** のような実行を行ったとする．定義されている状態番号の中で 1 と 4 が 1 度も使用されなかったので，この個体が世代交代を行うときにどちらかの状態番号が削除され状態数が減少する．減少する状態番号として，状態番号 1 が選択されたとすると，個体は**図 4** のように構造が変化する．

3.4　AGPA による問題解法の流れ

AGPA を問題に適用した場合，初期個体では初期状態とそれ以外の状態しか区別はできないため，簡単な解しか得られない．しかし，進化過程で状態数の増加が起こるため進化が進むと数多くの状態を区別できるようになり，より複雑な解を作り出すことができる．また，使っていない状態は削除されていくため，無駄な木を多数保持して性能が悪化することはない．このような状態数の増減が起こることにより，つねに問題に必要な状態数の近傍で探索を行い解を得ることができると考えられる．

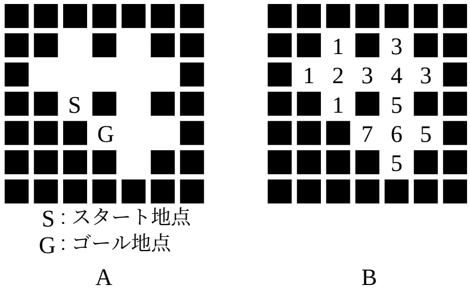


図 5　フィールドに与えられたポイント
Fig. 5　Score of each position in maze.

4.　実験環境の設定

4.1　迷路探索問題の設定

迷路探索問題を次のように設定する．フィールドは壁によって迷路となっており，スタート地点とゴール地点が 1 カ所ずつ設定されている．迷路を探索するエージェントは次のことを認識できる．

- エージェントの周囲 4 方向（N，S，E，W）の壁の有無
- エージェントの保持している状態番号

エージェントは 1 ステップごとに 4 方向のいずれかに 1 マス移動し，状態番号を更新する．ただし，進行方向が壁の場合はその場にとどまる．エージェントは決められたステップ数だけ行動できることとし，その数はスタート地点からゴール地点までの最短経路の移動に要するステップ数に 3 を加えた値とする．この迷路探索問題の解は，スタート地点からゴール地点までの最短路を最短ステップ数で移動することである．このような環境では，同一の入力に対して，状況に応じて異なる出力を行わなければならない状況が発生する．

4.2　適応度の計算

同一の入力に対しては同じ出力を行うという一般的な GP の性質上，単純に探索ステップ数を増加したとしてもゴールに到達する可能性は低いと考えられる．初期の世代でゴールに到達できる個体が発生せず進化が停滞してしまうことを防ぐため，ここでは，ゴールに到達するかどうかにかかわらずゴールに近づいた個体ほど高い適応度を持つように設定を行う．具体的な計算方法を以下に示す．

表 4 実験で用いる終端・非終端記号
（＊：N，S，E，W の 4 方向のいずれかを表す）
Table 4 GP functions and terminals.

記号	引数	記号の意味
If	3	第 1 引数に条件をとり真なら第 2 引数，偽なら第 3 引数を返す．
Wall_*	0	If の条件にくる．＊方向に壁があれば真，なければ偽．
Dummy	2	第 1 引数に状態番号に対する操作，第 2 引数に行動をとる．
Inc	0	Dummy の第 1 引数．現在の状態番号を 1 増やす．
Dec	0	Dummy の第 1 引数．現在の状態番号を 1 減らす．
Stay	0	Dummy の第 1 引数．現在の状態番号を維持する．
Set_i	0	Dummy の第 1 引数．状態番号を i にする．
Move_*	0	Dummy の第 2 引数．＊方向へ移動する．

フィールドの各位置には，式 (1) で計算されるポイント p が与えられている．

$$p = (m - s + 1) \quad (1)$$

m：スタート地点からゴール地点までの最短路を移動する最短ステップ数

s：各位置からゴール地点までの最短路を移動する最短ステップ数

適応度は，エージェントが各ステップで通過する位置のポイントをすべて加えたものとする．適応度計算の例を**図 5** の迷路を使って次に示す．図 5 の A が与えられた迷路とすると，式 (1) を使って計算される各位置のポイントは B のようになる．たとえばエージェントが N，S，N，E，S，E，S，S，W という移動をしたとする．5 ステップ目の S では進行方向が壁のため移動はできない．このときのエージェントの適応度は 1+2+1+2+3+3+4+5+6+7 と計算され 34 となる．最初の 1 は 0 ステップ目の初期位置のポイントである．

4.3 GP に関する設定

4.3.1 終端・非終端記号の設定

実験で用いる GP の終端・非終端記号を**表 4** に示す．非終端記号 If は (If A B C) という形をとる．A は条件を表し，Wall_* 以外の記号をとることはない．Wall_* で表された方向に壁があれば B，なければ C を返し，B，C は If か Dummy をとる．Dummy はプログラムの都合上用意した記号であり，第 1 引数に状態番号に対する操作 {Inc, Dec, Stay, Set_i} をとり，第 2 引数に Move_* をとる．それぞれの引数にこれら以外の記号をとることはない．これらの記号で表されるプログラムの例を以下に示す．Dummy は状態番号に対する操作と行動の組を表し，それ自体に意味はないため，その組を [ ] で表す．

```
(If Wall_N (If Wall_W [Stay Move_N]
[Set_1 Move_N]) [Inc Move_E])
```

上記のプログラムは次のように動作する．もし N に壁があり，かつ W に壁があれば状態番号はそのままで，N に移動する．また，N に壁があり，W に壁がなければ状態番号を 1 にし，N に移動する．もし N に壁がなければ状態番号を 1 増やし，E に移動する．

表 5 交叉の制限
Table 5 Restriction of crossover.

ノードの種類	対応する交叉可能なノード
If	If，Dummy
Wall_*	Wall_*
Dummy	If，Dummy
Inc，Dec，Stay，Set_i	Inc，Dec，Stay，Set_i
Move_*	Move_*

表 6 突然変異の制限
Table 6 Restriction of mutation.

ノードの種類	突然変異の制限
If	変異を起こす If の部分木と同じ高さの範囲内の部分木に変異
Wall_*	異なる Wall_* に変異
Dummy	Dummy の部分木または，高さ 5 の範囲内の部分木に変異
Inc，Dec，Stay，Set_i	この 4 種類の中の別の記号に変異
Move_*	異なる Move_* に変異

4.3.2 遺伝操作の設定

実験で用いた GP のパラメータについて述べる．初期個体の木は高さ 5 の範囲でランダムに作成する．選択方法はトーナメント選択にエリート保存戦略を組み合わせた方法を用いる．保存するエリートの数は 10 個体とし，トーナメントサイズは問題ごとに設定する．また，交叉率は 70%，突然変異率は 1%とする．なお，致死遺伝子の発生を防ぐため，ノードの種類別の交叉可能なノードを**表 5** のように，また突然変異の制限を**表 6** のように設定する．

5. 実験と考察

5.1 迷路探索問題への適用

この節で扱う問題はすべて，個体数を 200 とし，トー

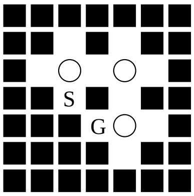


図 6　迷路 A
Fig. 6　Maze A.

```
State[0] :
(If Wall_N (If Wall_S (If Wall_E [Dec Move_W] [Inc
Move_E])(If Wall_W [Inc Move_E] [Dec Move_S]))(If Wall_E
(If Wall_S [Stay Move_N] [Set1 Move_S]) [Dec Move_E]))
State[1] :
(If Wall_N (If Wall_W [Set0 Move_E] [Stay Move_S])
(If Wall_W (If Wall_S [Inc Move_E] (If Wall_E
[Inc Move_S] [Dec Move_W]))(If Wall_S [Stay Move_N]
[Set1 Move_S])))
Default :
(If Wall_N (If Wall_W (If Wall_E [Set0 Move_N]
[Inc Move_S]) [Stay Move_E])(If Wall_W (If Wall_S
[Set1 Move_E] [Set0 Move_E]) [Stay Move_W]))
```

図 7　迷路 A を解くプログラムの例
Fig. 7　An example of programs which solve Maze A.

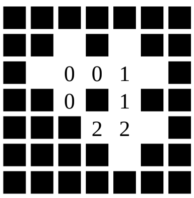


図 8　迷路 A での状態遷移の様子
Fig. 8　State transition in Maze A.

ナメント選択におけるトーナメントサイズは 5 として実験を行った．

5.1.1　迷　路　A

図 6 のような迷路 A を用意する．この迷路では○の位置でそれぞれ同一の状況となるが，最短路を進むためにはそれぞれ適切な方向へ移動しなければならない．よってこの迷路を解くためには状態数 3 以上が必要となる．

迷路 A に対する実験の結果得られた，最短路を移動するプログラムの例（冗長部分を取り除き簡単化したもの）を**図 7** に，このプログラムに従ってエージェントの状態番号を追跡したものを**図 8** に示す．この例では，定義されている状態は 0 と 1 の 2 つとなっているが Default 木を合わせると，迷路 A を解くために使用した状態数は 3 つといえる．使用した状態番号を見ると，図 6 で示した○の位置の手前で状態番号を遷移させ，同一の状況を区別しているのが分かる．この例では使用した状態数は迷路 A を解く最小状態数の 3 であったが，他の解として，状態数が 3 から 6 の範囲のものが存在し様々なパターンで解を導いていた．

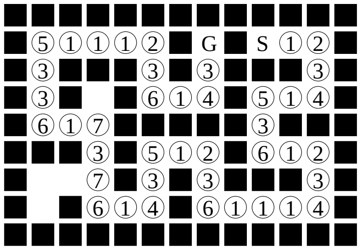


図 9　迷路 B
Fig. 9　Maze B.

5.1.2　迷　路　B

次に**図 9** のような迷路 B に対して実験を行った．迷路に記した同じ数字の位置で同一の状況となり，それぞれ適切な方向へ移動しなくてはならない．たとえば①は，壁が N と S にあるという状況を表し，最短路を進むためには 1 回目の①で E，2 回目の①で W，3 回目の①で再び E，...，というように E と W どちらかの進行方向をすべての位置において適切に選択しなければならない．このような状況が①から⑦まで 7 種類あるため，この迷路は迷路 A に比べより難しい迷路であるといえる．

迷路 B の最短路を移動するプログラムを**図 10** に，このプログラムに従ってエージェントの状態番号を追跡したものを**図 11** に示す．定義されている状態数は 0，1，2，3 の 4 つであるが，実行中状態番号 4，5 が Default 木で処理されていることを考えると迷路 B を解くために使用した状態数は 5 つである．このプログラムを獲得したときの各世代における最大適応度のグラフを**図 12** に，状態数に関するグラフを**図 13** に示す．図 13 の状態数とは，定義されている状態数のことをいう．この状態数のグラフから，状態数 8 までの範囲で幅広く探索を行いながらも，最良個体が持つ状態数の近傍を集中して探索していると考えられる．

5.1.3　迷　路　C

次に**図 14** のような迷路 C に対して実験を行った．この迷路は①から⑩までのすべての状況で適切な移動を行わなくてはならない．また，この迷路の最短路は 1 通りではなく，必要な状態数も簡単には判断できない．この迷路に本手法を適用したところ，**図 15** のようなプログラムを獲得した．このプログラムに従って

```
State[0] :
(If Wall_S (If Wall_E [Inc Move_W] (If Wall_W
[Dec Move_E] [Set3 Move_E]))(If Wall_N [Stay
Move_E] (If Wall_W [Dec Move_S] [Inc Move_W])))
State[2] :
(If Wall_N (If Wall_W [Stay Move_E] [Stay Move_S])
[Dec Move_N])
State[1] :
(If Wall_W (If Wall_N [Dec Move_S] (If Wall_S (If
Wall_E [Stay Move_E] [Set2 Move_N]) [Dec Move_N]))
(If Wall_N [Stay Move_W] [Dec Move_N]))
State[3] :
(If Wall_S (If Wall_W (If Wall_E [Set0 Move_N]
[Stay Move_N])(If Wall_E (If Wall_N [Set3 Move_W]
[Set1 Move_N])(If Wall_N [Inc Move_E]
[Set0 Move_W])))[Set0 Move_S])
Default :
(If Wall_S (If Wall_N [Inc Move_E] [Dec Move_W])
(If Wall_E [Set0 Move_S] (If Wall_W [Stay Move_E]
[Inc Move_W])))
```

図 10 迷路 B を解くプログラムの例
Fig. 10 An example of programs which solve Maze B.

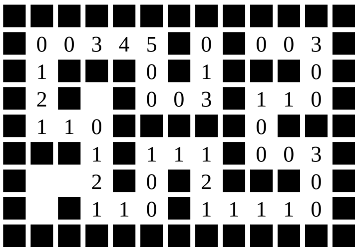


図 11 迷路 B での状態遷移の様子
Fig. 11 State transition in Maze B.

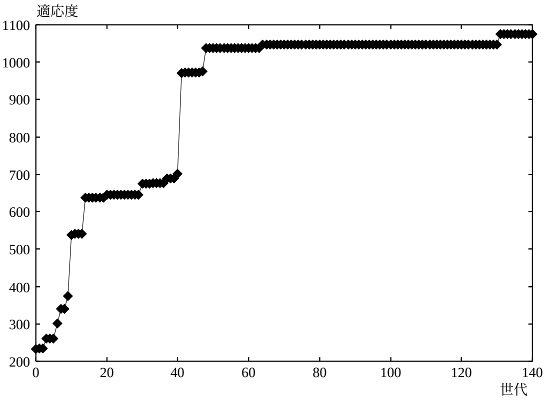


図 12 各世代の最大適応度
Fig. 12 Best fitness in each generation.

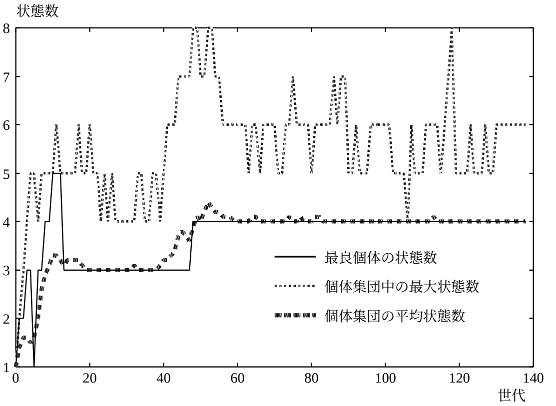


図 13 状態数の変化
Fig. 13 Changes of the number of internal states.

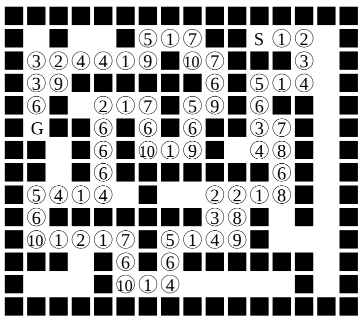


図 14 迷路 C
Fig. 14 Maze C.

エージェントの状態番号を追跡したものを**図 16** に示す．定義されている状態番号は 0，1，2，3，4，5 であり，実行中一度も Default 木を参照していないため迷路 C を解くために使用した状態数は 6 つといえる．この状態遷移の様子を見るとかなり複雑な状態遷移をしており，このような人手による作成が困難なプログラムも進化により生成できていることが分かる．

5.1.4 迷路探索問題のまとめ

必要な状態数の異なる迷路 A，B，C に対して本手法を適用した結果，いずれの迷路に対しても進化過程において必要な状態数を獲得し解を導くことに成功した．よって，本手法は必要とされる状態数によらず，どのような迷路に対しても有効であると考えられる．

5.2 従来手法との比較

5.2.1 比較実験の設定

本手法と従来手法との比較を行うために，難易度の異なる 3 つの迷路（図 6 の迷路 A，**図 17** の D，E）を用意した．それぞれの迷路の必要状態数は，迷路 A が 3，迷路 D が 4，迷路 E が 5 となっている．本手

```
State[0]:
(If Wall_W (If Wall_N [Inc Move_E] [Stay Move_N])
(If Wall_E (If Wall_N [Inc Move_S] [Stay Move_N])
(If Wall_S (If Wall_N [Set0 Move_E] [Stay Move_N])
[Stay Move_E])))
State[5]:
(If Wall_S (If Wall_E [Dec Move_W] [Dec Move_S])
(If Wall_W (If Wall_N (If Wall_E [Stay Move_W]
[Dec Move_N])(If Wall_E [Stay Move_N]
[Set1 Move_W])) [Dec Move_S]))
State[2]:
(If Wall_W (If Wall_E (If Wall_S [Dec Move_S]
(If Wall_N [Inc Move_E] [Stay Move_N]))
[Dec Move_E]) [Inc Move_W])
State[4]:
(If Wall_W [Dec Move_S] (If Wall_N (If Wall_E
[Set2 Move_S] (If Wall_S [Dec Move_W] [Stay
Move_S]))(If Wall_E [Dec Move_W] [Inc Move_E])))
State[1]:
(If Wall_N [Set5 Move_E] (If Wall_E (If Wall_S
[Inc Move_N] [Stay Move_S]) [Dec Move_E]))
State[3]:
(If Wall_W (If Wall_S [Set2 Move_N] [Inc Move_S])
(If Wall_S [Set3 Move_W] [Inc Move_W]))
Default:
(If Wall_W [Stay Move_N] (If Wall_S [Set0 Move_W]
[Dec Move_W]))
```

図 15　迷路 C を解くプログラムの例
Fig. 15　An example of programs which solve Maze C.

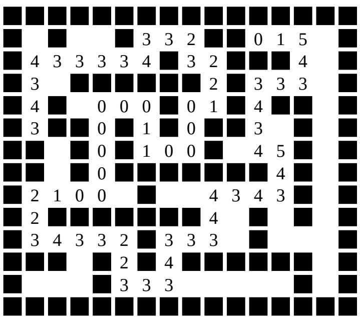
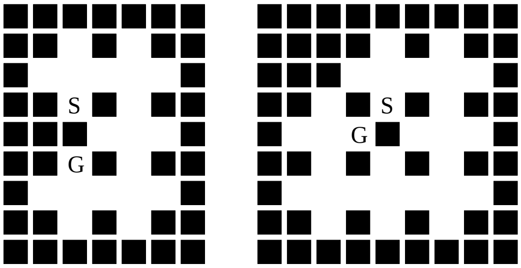


図 16　迷路 C での状態遷移の様子
Fig. 16　State transition in Maze C.

法と比較するための GP-オートマトンの状態数は，迷路ごとに 4〜6 種類用意する．各迷路において，制限時間内での解の検出率と，解を検出するまでの実行時間によって性能の比較を行う．各迷路における実験の設定を**表 7** に示す．なお，今回作成したプログラムは



迷路D：必要状態数4　　　迷路E：必要状態数5

図 17　比較実験に用いる迷路（D，E）
Fig. 17　Mazes D and E for comparison experiments.

C++で記述し，PentiumII 300 MHz の CPU を持つ PC 上で実行した．また，同一の条件で比較を行うため，GP-オートマトンでは，終端・非終端記号，遺伝操作の制限などはすべて本手法と同じものを用いた．

5.2.2　比較実験の結果と考察

迷路 A に対してはすべての実行において制限時間内に解を検出したため，60 回の実行の中から実行時間の上位 40 回の実行時間について比較を行うこととする．各迷路に対する実験結果として，制限時間内に解を検出した回数，解を検出したときの平均世代数，平均実行時間を**表 8**，**表 9**，**表 10** に示す．

これらの結果から状態数の違いによる GP-オートマトンの性能を比較する．どの迷路においても状態数が増えると少ない世代交代で解を獲得できるが実行時間は増加する，ということがいえる．また，迷路 A では状態数が少ないほど性能が良いという結果であったが，迷路 D，E では解の検出率，実行時間ともに，必要最小限の状態数よりも少し多い状態数で最も性能が良くなり，さらに状態数が増えると再び性能が悪くなるという結果が得られた．この迷路 A と迷路 D，E との結果の違いは迷路の難易度の違いによるものと考えられる．迷路 D，E にみられた，状態数が必要最小限でも，あるいは多くなりすぎても性能が落ちる原因としては次のようなことが考えられる．

- 状態数が少なければ，木の数が少ないため 1 回の世代交代にかかる時間は少ない．しかし，冗長な状態の使い方があまりできないため，解の表現方法に多様性がなく，解の検出の可能性が低くなってしまう．また，冗長な木が少なく多様性が乏しいため局所解に陥った場合にそこから抜け出すのが困難である．そのため解を見つけるためにはより多くの世代交代を必要とする．
- 状態数が多ければ，幅広い状態の使い方ができるため解の検出の可能性も高くなる．また，木の数が多くなり多様性が増すため少ない世代交代数で

表 7 比較実験の設定
Table 7 Experimental settings for comparison experiments.

	迷路 A	迷路 D	迷路 E
GP-オートマトンで用いる状態数	{3, 6, 9, 12}	{4, 6, 8, 12, 16, 24}	{5, 8, 10, 12}
実験回数（回）	60	25	20
個体数	200	200	400
トーナメントサイズ	5	5	10
制限時間（秒）	120	300	600

表 8 迷路 A における比較実験の結果（40 回の平均）
Table 8 Comparison of experimental results in Maze A.

	本手法	状態数 3	状態数 6	状態数 9	状態数 12
世代交代（回）	7.98	11.88	7.83	7.33	6.18
実行時間（秒）	9.28	9.63	15.98	22.50	25.73

表 9 迷路 D における比較実験の結果（解検出時平均）
Table 9 Comparison of experimental results in Maze D.

	本手法	状態数 4	状態数 6	状態数 8	状態数 12	状態数 16	状態数 24
検出回数（回）	15	3	9	15	16	9	4
世代交代（回）	30.07	113.00	74.22	36.33	38.13	29.33	26.50
実行時間（秒）	92.54	198.67	146.44	115.40	166.88	201.22	212.50

表 10 迷路 E における比較実験の結果（解検出時平均）
Table 10 Comparison of experimental results in Maze E.

	本手法	状態数 5	状態数 8	状態数 10	状態数 12
検出回数（回）	7	5	8	6	4
世代交代（回）	165.71	190.40	77.88	85.00	66.50
実行時間（秒）	187.57	274.60	165.00	206.50	196.50

表 11 迷路 D における比較実験の結果（失敗時の平均）
Table 11 Comparison of experimental results in Maze D.

	本手法	状態数 4	状態数 6	状態数 8	状態数 12	状態数 16	状態数 24
世代交代（回）	109.80	138.05	141.63	89.10	59.10	44.38	25.52

解を見つける可能性はある．しかし，状態数が多くなると 1 回の世代交代に時間がかかるようになる．そのため解を見つけるためにはより多くの実行時間を必要とする．

従来法である GP-オートマトンの性能は一般にこのような結果を示すと考えられる．**表 11** に迷路 D の実験で解を検出できなかったときの最終的な世代交代数の平均を示した．これは 300 秒で平均どのくらいの世代交代が行えたかを表している．この表より，状態数が増えると明らかに 1 回の世代交代に要する時間が多くなっているのが分かる．

また，本手法および従来手法における最適解の検出回数の結果を**表 12** に示す．ここでは，従来手法として，適切な状態数を設定した GP-オートマトンのほかに，メモリを利用した不完全知覚解決の手法である Index Memory [2),3)] による結果を示す．Index Memory における終端・非終端記号は文献 2) と同様のものを用いた．ただし，終端記号はエージェントの視覚に相当する上下左右のみとし，扱うメモリの数は 12 とした．また Index Memory における GP のパラメータは本手法および GP-オートマトンと同一である．Index Memory における記号設定は，本手法および GP-オートマトンと大きく異なるため，直接的な比較は困難であるが，本手法は Index Memory に比べて高い成績

表 12 従来手法との最適解検出回数の比較
Table 12 Comparison of detection frequencies of the optimal solution.

	迷路 A（60 試行）	迷路 D（25 試行）	迷路 E（20 試行）
本手法	60	15	7
GP-オートマトン（最適状態数）	60	16	8
Index Memory	55	6	5

をあげていることが分かる．また，本手法は最も性能の良い GP-オートマトンと同等の性能を示した．このことから，本手法は問題に必要な状態数が変化しても，必要に応じて状態数を変化させ効率良く探索を行うことができるといえる．

ただし，本手法においても，場合によっては局所解からの脱出に時間がかかることもみられた．状態数の違いによる GP-オートマトンの性能の比較実験でも述べたように，問題が難しくなると，ある程度冗長な部分があることで探索効率が上がることが分かったが，本手法でも状態数が少ないまま進化が進むと局所解に陥りやすくなると考えられる．このような場合の突然変異などによる対処方法の検討を今後行う必要がある．

5.2.3　比較実験のまとめ

難易度の異なる 3 つの迷路に対して，それぞれ状態数の異なるいくつかの GP-オートマトンと本手法を比較した．どの迷路の場合においても，本手法は適切な状態数の GP-オートマトンと同等の性能を示した．

従来法である GP-オートマトンは状態数の設定の仕方によってその性能は大きく左右される．ある程度冗長な部分があった方が良いということは分かったが，問題を解くために必要な状態数が分からない場合はやはり，状態数に対する試行錯誤を繰り返し探索効率の良い状態数を探さなくてはならない．本手法はどの迷路においても，適切な設定を行った GP-オートマトンと同等の性能を示したことから，必要な状態数が未知の問題に対しても有効であるといえる．

6. おわりに

本論文では，GP-オートマトンを拡張して問題に応じて必要な状態数を進化過程で獲得する適応型 GP-オートマトンを提案した．そして，不完全知覚が問題となる迷路探索問題において，本手法と従来の GP-オートマトンとの性能の比較を行った．実験の結果，本手法は，必要な状態数が異なる迷路すべてに対して解を得ることに成功した．また，従来の GP-オートマトンとの比較では，本手法は適切な状態数を設定した場合の GP-オートマトンと同等の性能を示した．適切な状態数を試行錯誤により決定する必要がないことは，本手法の利点の 1 つである．

今後は，より多くの状態数を必要とする問題に対して本手法を適用し有効性を示す必要がある．また，現在は単純な遺伝操作のみを行っているが，本手法により適した遺伝操作を設計することが課題となる．

参 考 文 献

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

2) Teller, A.: The evolution of mental models, *Advances in Genetic Programming*, pp.199–220, MIT Press (1994).

3) Andre, D.: The Evolution of Agents that Build Mental Models and Create Simple Plans Using Genetic Programming, *Genetic Algorithms: Proc. 6th International Conference* (*ICGA95*), pp.248–255, Morgan Kaufmann (1995).

4) Haynes, T., Wainwright, R., Sen, S. and Schoenefeld, D.: Strongly typed genetic programming in evolving cooperation strategies, *Genetic Algorithms: Proc. 6th International Conference* (*ICGA95*), Pittsburgh, PA, USA, pp.271–278, Morgan Kaufmann (1995).

5) Luke, S. and Spector, L.: Evolving Teamwork and Coordination with Genetic Programming, *Genetic Programming 1996: Proc. 1st Annual Conference*, pp.150–156, MIT Press (1996).

6) Luke, S.: Genetic Programming Produced Competitive Soccer Softbot Teams for RoboCup97, *Genetic Programming 1998: Proc. 3rd Annual Conference*, pp.214–222, Morgan Kaufmann (1998).

7) Iba, H.: Emergent Cooperation for Multiple Agents Using Genetic Programming, *Parallel Problem Solving from Nature IV, Proc. International Conference on Evolutionary Computation*, LNCS, Vol.1141, pp.32–41, Springer Verlag (1996).

8) Iba, H.: Multiple-Agent Learning for a Robot Navigation Task by Genetic Programming, *Genetic Programming 1997: Proc. 2nd Annual Conference*, pp.195–200, Morgan Kaufmann (1997).

9) Iba, H. and Terao, M.: Controlling Effective Introns for Multi-Agent Learning by Genetic Programming, *GECCO-2000: Proc. Genetic and Evolutionary Computation Conference*, pp.419–426, Morgan Kaufmann (2000).

10) 原　章，長尾智晴：自動グループ構成手法 ADG によるマルチエージェントの行動制御，情報処理学会論文誌，Vol.41, No.4, pp.1063–1072 (1999).

11) 福寄雅洋，原　章，長尾智晴：不完全知覚問題解決のための時系列依存分類システム（TCS）の提案，電気学会部門誌（電子・情報・システム），Vol.122-C, No.7, pp.1218–1225 (2002).

12) Ashlock, D.: GP-Automata for Dividing the Dollar, *Genetic Programming 1997: Proc. 2nd Annual Conference*, pp.18–26, Morgan Kauf-

mann (1997).

(平成 14 年 8 月 5 日受付)
(平成 15 年 7 月 3 日採録)

片岡　寛明

1975 年生．1999 年東京工業大学工学部情報工学科卒業．2001 年同大学大学院総合理工学研究科物理情報システム創造専攻修士課程修了．同年，日本テレコム株式会社入社．現在，広域イーサネットサービスのサービス企画・開発に従事．

原　　章（正会員）

1974 年生．1997 年東京工業大学工学部電気・電子工学科卒業．1999 年同大学大学院総合理工学研究科物理情報工学専攻修士課程修了．2002 年同大学院同研究科物理情報システム創造専攻博士後期課程修了．現在，広島市立大学情報科学部知能情報システム工学科助手．博士（工学）．進化的計算法，マルチエージェントシステム等に関する研究に従事．IEEE 会員．

長尾　智晴（正会員）

1959 年生．1985 年東京工業大学大学院博士後期課程中退．同年同大学工学部附属像情報工学研究施設助手．1995 年同大学工学部附属像情報工学研究施設助教授．2001 年横浜国立大学大学院環境情報研究院教授，現在に至る．工学博士．画像処理，進化的計算法，神経回路網，マルチエージェントシステム，進化経済学等に関する研究に従事．著書「進化的画像処理」（昭晃堂），「最適化アルゴリズム」（昭晃堂）等．電子情報通信学会，人工知能学会，計測自動制御学会，映像情報メディア学会，進化経済学会，IEEE 等会員．