博士論文

グラフ構造表現を用いたプログラムの進化的自動生成に関する研究

A Study on Evolutionary Automatic Generation of Programs Using Graph Structured Representation

国立大学法人　横浜国立大学大学院
環境情報学府

白川　真一
Shinichi SHIRAKAWA

2009年3月

あらまし

近年，計算機や機械の高性能化に伴って複雑なアルゴリズムやシステムの構築が求められるようになってきている．一般的に，人手でのアルゴリズム構築は多くの試行錯誤を要求するため膨大な労力が必要になると考えられる．これに対して，コンピュータプログラムを自動生成する自動プログラミングに関する研究が非常に活発に行われている．ここで，自動プログラミングとは人間によるプログラムの記述は行わず，全自動で目的のプログラム（アルゴリズム）を生成する方法を示す．

自動プログラミングの代表的な例として，遺伝的プログラミング（Genetic Programming; GP）が挙げられる．GP は木構造で表現されたプログラムを，遺伝的アルゴリズム（Genetic Algorithm; GA）を用いて自動的に生成する．GA は自然淘汰，交叉，突然変異などの遺伝操作を用いて新しい個体（探索点）を生成し，実用解あるいは最適解を高速に発見する探索手法である．一般的に GP では，ある解表現の評価値を与えることができれば，探索を行うことができるため，実現したいアルゴリズムの詳細な手続きが不明であっても実用的な解を得ることができると期待できる．これまでに GP は関数近似，回路生成，画像処理，自律ロボットの行動制御など，様々な分野に適用され，その有効性が示されている．GP の研究分野では非常に活発な研究が行われており，これまでに様々な改良や拡張が提案されている．しかし複雑なプログラムを自動生成しようとした場合，様々な問題が存在する．一般的な GP ではプログラムの表現形式として木構造を用いているが，木構造ではループや再帰構造の表現が困難である．また，複雑なプログラムでは複数のデータ型を扱う必要がある．さらに，GP には世代交代を繰り返しているうちに木が大きくなりすぎてしまうブロートという問題が潜在的に存在する．

本研究では，グラフ（ネットワーク）構造をプログラムの表現形式とする自動プログラミングに関する手法の提案を行う．グラフ構造に着目する理由は，その表現能力の高さにある．グラフ構造表現を用いることで，ノードの再利用やループの表現，時系列データの保持などが可能になると考えられる．

本研究では，まず問題領域として画像変換の自動構築を考え，グラフ構造表現を用いた画像変換の自動構築法である Genetic Image Network（GIN）とその拡張である Feed Forward Genetic Image Network（FFGIN）の提案を行う．いくつかの画像変換の自動構築実験を通して，GIN と FFGIN が従来手法より有効であることを示す．

次に，GIN を基に画像変換部を有する画像分類器の自動構築法として，Genetic Image Network for Image Classification（GIN-IC）を提案する．GIN-IC は画像変換部，特徴抽出部，演算部からなる構造を自動構築するため，対象とする画像分類問題に有効な画像変換や特徴量抽出が行える．多クラスのテクスチャ画像の分類実験を通して，GIN-IC を用いることで性能の高い分類器の自動構築が行えることを示す．

その後，より任意のプログラムの自動生成を目的とした Graph Structured Program Evo-

lution（GRAPE）の提案を行う．GRAPEではグラフ構造を用いた表現の拡大，複数のデータ型の取り扱いなどを可能にしている．GRAPEを用いて，階乗を求めるプログラムや任意の長さのリストをソートするプログラムなど，ループ構造を必要とするプログラムの自動生成を行う．その結果，GRAPEによって通常のGPでは自動生成が困難であったループ構造を必要とするプログラムの自動生成が行えることを示す．

最後に，GRAPEを用いて探索アルゴリズムの自動生成を行い，対象とする問題に対して有効な探索アルゴリズムの獲得を行う．ベンチマーク関数とテンプレートマッチングの問題を対象に，GRAPEによって有効な探索アルゴリズムの自動生成が行えることを示す．

Abstract

A construction of the complex algorithms and systems is becoming increasingly important along with development of the computers and the machines. In general, it is difficult to handed construction of algorithms because it requires much trial and error. On the other hand, automatic programming, which generates computer programs automatically, has been actively investigated in recent studies. Automatic programming is the methods for full-automatic generation of programs (algorithms) without handed programming.

Genetic programming (GP) is a typical example of automatic programming. GP evolves computer programs, which usually have a tree structure, and searches for a desired program using a genetic algorithm (GA). GA is a search algorithm which generates new individuals (searching points) by using genetic operators such as selection, crossover, and mutation, and then discovers practical or optimal solution fast. It is possible to search for GP if the evaluation value of a certain solution (program) can be given. Therefore, we expect to obtain practical solutions even if a detailed procedure of the algorithms that wants to be achieved is uncertain. GP have been applied to various fields and its effectiveness is demonstrated. The typical examples are symbolic regression, circuit design, image processing, and autonomous robots control. Recently, many extensions and improvements to GP have been proposed. However, various problems exist when the complex program is constructed automatically. The tree structure which is usually used in GP is difficult to represent loop and recursion, and it is necessary to handle multiple data types. Moreover, GP has a tendency to create programs with unnecessarily large size.

In this study, automatic programming methods whose representation is graph (network) structure are proposed. The reason to use the graph structure is its height description ability. It has various advantages such as reusing of nodes, loop structure and containing time series by using graph structure.

At first, the author targets automatic construction of image transformation as a problem domain. The author proposes an automatic construction method for image transformation by using graph representation, named Genetic Image Network (GIN) and its extended method, Feed Forward Genetic Image Network (FFGIN). From several experiments of automatic construction of image transformation, we verify the effectiveness of GIN and FFGIN.

After that, the author proposes a method of automatic construction of image classifiers based on GIN, designated as Genetic Image Network for Image Classification (GIN-IC). GIN-IC transforms original images to easier-to-classify images using image transformation nodes, and selects adequate image features using feature extraction nodes. The author applies GIN-IC to test problems involving multi-class categorization of texture images, and shows that the use of image transformation nodes is effective for image classification problems.

In addition, the author proposes Graph Structured Program Evolution (GRAPE) which is an automatic generation method for arbitrary programs. The author applies GRAPE to automatic generation of programs which need loop structure such as factorial and sorting a list. From the experimental results, the author shows that GRAPE enables to construct the complex programs which are difficult to automatic construction by usual GP.

Finally, the author proposes a method for evolving search algorithms using GRAPE. The author applies the proposed method to construct search algorithms for benchmark function optimization and template matching problems. Numerical experiments show that the constructed search algorithms are effective for utilized search spaces and also for several other search spaces.

目次

図目次

表目次

第1章　序論

1.1　本論文の背景と目的

近年，計算機や機械の高性能化に伴って複雑なアルゴリズムやシステムの構築が求められるようになってきている．一般に，人手でのアルゴリズム構築は多くの試行錯誤を要求するため，膨大な労力が必要になると考えられる．これに対して，コンピュータプログラムを自動生成する自動プログラミングに関する研究が非常に活発に行われている．ここで，自動プログラミングとは人間によるプログラムの記述は行わず，全自動で目的のプログラム（アルゴリズム）を生成する方法を指す．自動プログラミングの代表的な例として，遺伝的プログラミング（Genetic Programming; GP）が挙げられる．GP は木構造で表現されたプログラムを，遺伝的アルゴリズム（Genetic Algorithm; GA）を用いて自動的に生成する．その際，遺伝操作として部分木の交換による交叉やノードの突然変異などが用いられる．これまでに GP は関数近似，回路生成，画像処理，自律ロボットの行動制御など，様々な分野に適用され，その有効性が示されている．GP の研究分野では非常に活発な研究が行われており，これまでに様々な改良や拡張が提案されている．しかし複雑なプログラムを自動生成しようとした場合，様々な問題が存在する．一般的な GP ではプログラムの表現形式として木構造を用いているが，木構造ではループや再帰構造の表現が困難である．また，複雑なプログラムでは複数のデータ型を扱う必要がある．さらに，GP には世代交代を繰り返しているうちに木が大きくなりすぎてしまうブロートという問題が潜在的に存在する．

本研究の目的は，グラフ（ネットワーク）構造[†]をプログラムの表現形式とする自動プログラミングに関する手法の提案を行い，その有効性を示すことである．グラフ構造に着目する理由は，その表現能力の高さにある．グラフ構造表現を用いることで，ノードの再利用やループの表現，時系列データの保持などが可能になると考えられる．本研究では，まず最初に問題領域として画像変換の自動構築を考え，グラフ構造表現を用いた画像変換の自動構築法である Genetic Image Network（GIN）の提案を行う．次に，GIN を基に画像変換部を有する画像分類器の自動構築法である Genetic Image Network for Image Classification（GIN-IC）を提案する．その後，より任意のプログラムの自動生成を目的とした Graph Structured Program Evolution（GRAPE）の提案を行い，ループ構造を必要とするプログラムの自動生成の問題に適用する．最後に，GRAPE を用いて対象とする問題に対して有効な探索アルゴリズムの自動生成を行う．

[†]本論文では “グラフ構造” と “ネットワーク構造” という用語を同様の意味で用いることとする．

1.2 本論文の構成

本論文の構成は次のとおりである．まず第2章では，本研究と関連の深い従来研究について述べる．第3章では，グラフ構造表現を用いた画像変換の自動構築法としてGINを提案し，画像変換の自動構築実験を通して，その有効性を検証する．次に第4章では，GINを基に画像変換部を有する画像分類器の自動構築法GIN-ICの提案を行う．GIN-ICをテクスチャ画像の分類問題に適用し，画像変換部含む画像分類器を自動構築することの有効性を示す．第5章では，より任意のプログラムの自動生成を目的とした手法であるGRAPEの提案を行い，ループ構造が必要なプログラムの自動生成問題に適用し，その有効性を検証する．そして，第6章ではGRAPEを用いて対象とする問題に対して有効な探索アルゴリズムの自動獲得を行う．最後に第7章で結論を述べ，本論文を総括する．

第2章　本研究に関する従来研究

本章では，本研究と関連の深い進化計算，進化計算によるプログラムの自動生成，グラフ構造表現を用いた自動プログラミング，自動プログラミングの画像処理への応用に関する従来研究について述べる．

2.1　進化計算（Evolutionary Computation; EC）

本節では，生物の進化にヒントを得た探索手法である進化計算（Evolutionary Computation; EC）について述べる．進化計算の代表的なものとして，遺伝的アルゴリズム（Genetic Algorithm; GA）[1–5]，遺伝的プログラミング（Genetic Programming; GP）[6–10]，進化的プログラミング（Evolutionary Programming; EP）[11]，進化戦略（Evolution Strategy; ES）[12]などが挙げられる．他にも，生物の群行動から着想を得たAnt Colony Optimization（ACO）[13, 14]やParticle Swarm Optimization（PSO）[15, 16]などが提案され，現在でも活発に研究が行われている．これらの手法は，どのような問題に対しても汎用的に適用できるように設計された，アルゴリズムの基本的な枠組みであり，そのような手法を総称してメタヒューリスティクス（Meta Heuristics）という．

ここでは，進化計算の代表的な例であるGAと，プログラムを自動生成するGP，およびGPの改良について述べる．

2.1.1　遺伝的アルゴリズム（Genetic Algorithm; GA）

GA [1–5]は，J. H. Hollandによって提案された最適化および探索のためのアルゴリズムである．GAは生物の進化プロセスから着想された多点探索に基づく探索アルゴリズムであり，自然淘汰，交叉，突然変異などの遺伝操作を用いて新しい個体（探索点）を生成し，実用解あるいは最適解を高速に発見する手法である．生物の進化の過程にヒントを得た比較的単純な基本原理を基にしており，ほとんどあらゆる最適化，探索の問題に適用可能な枠組みである．このためGAは，現在非常に広範囲な分野で利用され，その有用性が示されている．

GAの処理の流れは図2.1のようになる．GAでは，初めに初期個体集団として決められた数の個体を作り出す．それぞれの個体は問題に対する解の候補であり，一次元の文字列として表現されることが多い．これは生物の染色体に相当する．適応度とは一般にその個体が問題の環境にどの程度適応しているかを表す評価値であり，個体が置かれた環境と染色体の情報から値が決まる．こうして得られた各個体の適応度を基にして新しい世代の個体集団を生成する．これを世代交代といい，その際各個体には遺伝操作が行われる．この世代交代を終了条件を満たすまで繰り返す．

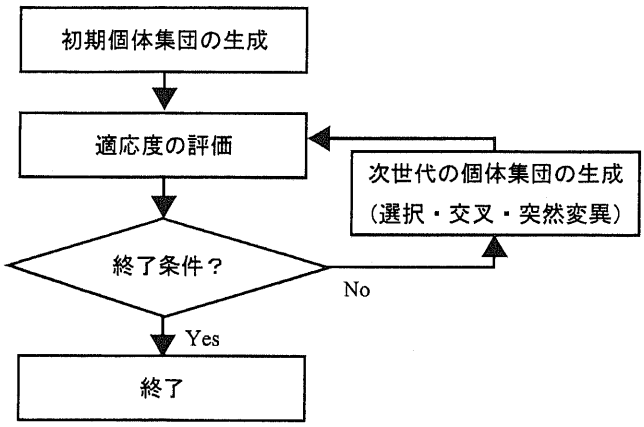


図 2.1: GA の処理の流れ

これまでに世代交代モデルとして様々な手法が提案されている．最も基本的な手法として Simple GA（SGA）がある．SGA ではまず，適応度に比例した選択確率を用いたルーレット選択によって，個体集団と同数の個体を重複を許して選択する．次に，選択した集団の個体を親個体として選択して交叉を行い，生成された子個体を次世代の個体集団とする．SGA には，高い選択圧下での早すぎる収束（初期収束）および低い選択圧下での停滞という問題がある．

この問題を克服するために提案された世代交代モデルの 1 つに Minimal Generation Gap（MGG）[17, 18]がある．MGG では，まず適応度を無視して個体集団からランダムに 2 個体を親個体として選択する（重複を許さない）．次に，この親個体に交叉を繰り返し実行し，子個体を複数個生成する．このようにして得られた親個体とすべての子個体の中から，適応度の最も高い 1 個体と，ルーレット選択によって選択された 1 個体を次世代に残す．このモデルでは，探索の後半まで個体集団の多様性が保持され，SGA に比べ初期収束が起こりにくいとされている．MGG はこれまでに多くの問題に適用され，その有効性が示されている [17–21]．

GA の基本操作には，次の 3 つがある．

選択（Selection）

個体の適応度に応じて繰り返し選択を行うことによって，淘汰あるいは増殖を行う．適応度に応じた選択を行うため，適応度の高い個体ほどより多くの子孫を残しやすくなる．代表的な選択方式として，次に示すルーレット選択，トーナメント選択，ランク選択，エリート保存などが挙げられる．

- ルーレット選択（Roulette Selection）

  適応度に比例した確率で個体を選択する．

- トーナメント選択（Tournament Selection）

個体集団からトーナメントサイズとして決められた数の個体をランダムに選択し，その中で最も適応度の高い個体をトーナメントの勝者として選択する．

- ランク選択（Rank Selection）

  各個体の適応度の順位で個体の選択確率を決める．

- エリート保存戦略

  決められた数の個体を適応度の高い順にそのまま次の世代に残す．

トーナメント選択には次のような特徴があるため，他の手法と比較して優れていると言われている．

- トーナメントサイズを変えることで，各個体の選択確率を変えることができる．
- 選択確率が適応度の値に依存しないので，適応度が接近している個体間の選択にも利用できる．

また，エリート保存戦略は遺伝操作によって起こる優良個体の消滅を防ぐことができる．エリート保存戦略は通常，他の選択と組み合わせて用いられる．

交叉（Crossover）

適当な個体の組を作り，お互いの染色体の遺伝子を組替え，両親の性質を兼ね備えた新しい個体を生成する．これによって両親の優れた部分が組み合わされ，より優れた個体が生成されることによって個体集団の進化が促進される．交叉にも，一点交叉，多点交叉，一様交叉，セグメント交叉など様々な種類がある．

突然変異（Mutation）

全遺伝子に対して非常に低い生起確率に基づいて，その遺伝子を変更する．突然変異の目的として，探索空間の探索範囲を限定してしまうことの回避と，局所解からの脱出が挙げられる．

2.1.2 遺伝的プログラミング（Genetic Programming; GP）

GP [6–10]は，進化計算によってプログラムや手続きを生成する．一般的な GP においては，木構造を用いてプログラムを表現する．GP では木構造を用いることで，Lisp 言語の S 式の形をしたプログラムを扱うことができる．

GP も GA と同様に交叉，突然変異などの遺伝操作によって，世代交代を行う．GP で用いられる標準的な交叉は二つの個体の部分木を交換するという方法である．GP では個体の形や大きさが決まっていないため，各個体で交叉させる場所をそれぞれ確率的に決める．突然変異では木のノードをランダムに別のノードや部分木に変更する．図 2.2 に遺伝操作の例を示す．

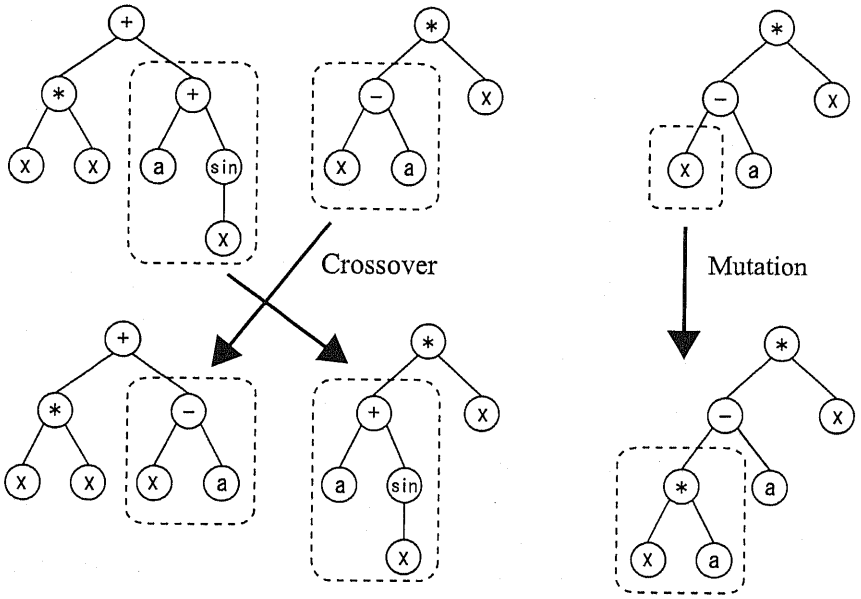


図 2.2: GP の遺伝操作の例

GP では個体の大きさが決まっていないため，世代交代を繰り返しているうちに木が大きくなりすぎることがある．これはブロート（bloat）と呼ばれ，GP の潜在的な問題の一つである．木が大きくなることで問題の解が得られればよいが，一般に木が大きくなりすぎると冗長な部分が増え，探索効率が悪くなる．このような現象を防ぐために，GP では木の大きさを制限する必要がある．次に一般的な方法を挙げる．

- 木の高さを制限する．
- ノード数を制限する．
- 論理的に意味をなさないノード，部分木を取り除く．

このような方法で木の大きさを制限することができる．しかし，あまり制限を厳しくすると，個体の多様性が失われ探索効率が悪くなってしまうので，ある程度の冗長性をもたせた制限にする必要がある．

2.1.3 遺伝的プログラミングの改良

モジュール化

アルゴリズムやプログラムを作成する場合，人間にとって自然なのはモジュール化手法を使って，小さなブロックに分割することである．これまでに GP に対する様々なモジュール化手法が提案されている．

サブルーチンの自動生成が GP の機能として有効であると考えられ，それに基づいて提案されたのが自動関数定義（Automatically Defined Functions; ADF）[7]である．ADF を

含んだプログラム（個体）は，通常の木構造GPで使用される1つの木である．その木は次の2つの部分から構成される：

- 本体部（適応度評価を受ける部分）
- 関数定義部（1つ以上のADFが定義されている部分）

ADFのこれら2つの部分は，両方とも進化の対象となり，ADFを使用したGPは実行結果として，関数定義を伴ったモジュール構造をしたプログラムを生成する．

AngelineとPollackによって提案されたModule Acquisition（MA）[22, 23]は，あらかじめ自動的に定義するための集団を設定しないという点でADFとは大きく異なる．MAの特徴をまとめると次のようになる．

- MAで用いられるサブルーチンはモジュールと呼ばれ，個体中の部分構造をランダムに抽出して得られる．
- 得られたモジュールはライブラリに保存される．
- モジュールは新たに非終端記号（関数）として登録され，本体で参照される．
- モジュールはグローバル関数として扱われる．すなわち，ある個体から得られたモジュールは全個体から参照可能である．

ライブラリ内でのモジュールの進化（更新）は行われず，1つのモジュールは個体から参照されている限りライブラリ内に保存される．また，ライブラリに保存された段階ではモジュールを参照する個体は集団内に1つだけであるが，参照した個体が良い評価を得ると世代交代によって集団内に広がり，その数は増加することになる．

GPに対するモジュール化の試みとして他にも，適応的モジュール生成（Adaptive Representation）[24]や自動的マクロ定義（Automatically Defined Macros; ADM）[25]などがある．適応的モジュール生成では問題としている分野のヒューリスティックスや集団に関する統計情報に基づいて，小規模の部分木がモジュールの候補として選択される．選ばれた部分木は集団から抽出され，新しい関数あるいはサブルーチンとして今後の進化に供される．

複雑なデータ構造と抽象データ型

GPは他のソフトコンピューティング手法とは違って，記号情報（例：コンピュータプログラム）を出力として生成する．また，記号情報を入力として受け付け，それを効率良く処理することができる．GPシステムは，必要となる関数記号集合を用意することができるならば，任意のデータ型を含んだ学習パターンを処理することができる．つまり，文字列，画像，木などコンピュータプログラムで扱うことのできる任意のデータ型を処理することができる．例えば，GPを用いて数学の証明を進化させる際，学習パターンには算術命題を表現した木が使われた[26]．また，リストのソートプログラムの進化などにもGPは適用されている[27, 28]．GPは抽象データ型の進化にも使うことができる．LangdonはGPシステムを使って，抽象データ型を進化させることに成功した[29, 30]．

型付き遺伝的プログラミング（Strongly Typed Genetic Programming; STGP）

型付き遺伝的プログラミング（Strongly Typed Genetic Programming; STGP）[31]はMontanaによって提案された．データ型をGPに導入する利点は，生成するプログラムに制約を設けることでGPの探索効率を向上させることである．例えば，型の合わないプログラムを生成しないように，GPオペレータの適用を制限する．

ループと再帰

ループと再帰は，プログラムの作成において重要な役割をもつ．繰り返し構造を使うことによって，コンパクトなプログラムになり，一般化が可能となる．しかし，繰り返し構造を伴うプログラムは，簡単に無限ループに陥ることがある．プログラム中の無限ループを発見することは理論的に不可能である．Turingの停止定理によると，“すべて”のプログラムの停止特性を決定することのできるプログラムは存在しない．機械的な手続きで停止特性を決定することのできるプログラムは数多く存在するが，一般的に停止問題を解くことはできない．この停止定理はGPシステムによるプログラムの進化に対して重要な示唆を与える．プログラムを進化させる場合，前もってどのプログラムが終了し，どのプログラムが終了しないかを知ることはできない．つまり，GPにおいて個体が無限ループに陥っているかどうかは，適応度評価を行わないと分からないということを意味する．たとえ無限ループを見つけることができるとしても，学習フェーズでは容認できないほどの長い実行時間が必要になる有限ループが存在する可能性がある．そのため，ループと再帰を含んだプログラムを進化させる際には，プログラムの実行を制御する方法を決めておく必要がある．

無限ループあるいは準無限ループを制御するアプローチとしては次のような方法が考えられる．

- 各プログラムに繰り返し回数の制限を設ける．
- プログラムに与える制限回数を学習パターン毎に分配し，各プログラムに解の質と実行回数とのトレードオフに関する判断を行わせる．
- 問題領域，関数記号集合および終端記号集合を無限ループあるいは非常に長いループに陥らないという保証ができるように設計する．ただしこの場合，表現を変更したりプリミティブを変更したりする必要があるかもしれない．

上記の1番目と2番目のアプローチは，時間制限実行（Time-bounded execution）であるといえる．GPにおける時間制限実行には，プログラムの実行回数や実行時間を制限したり，あるいは実行時間が長くなるとプログラムにペナルティを与えるなどといったものがある．

Kinnearは，特別に作ったループ構造（dobl）を使用してソートプログラムを進化させた[28]．ループ構造は限られた長さのリストに対して使用されるので，繰り返しは有限回に押さえられる．また，Kozaは再帰的な系列を進化させることによって，フィボナッチ数列を生成する実験を行った[6]．この実験は再帰プログラムの真の実例ではないが，再帰問題に迫っているといえる．Kozaのアプローチでは，生成されたプログラムは必ず終了することが保証されている．

2.2 進化計算によるプログラムの自動生成

進化計算による自動プログラミング（Automatic Programming）の代表例として，2.1.2で述べたGPが挙げられる．現在までに木構造を用いたGPの他に，進化計算による自動プログラミングの手法が数多く提案されている．

Kozaは“Genetic Programming – An Introduction; On the Automatic Evolution of Computer Programs and its Applications”の中で自動プログラミングという用語について次のような定義をしている[8, 9]．

1. コンピュータ上で走る実体（すなわち，コンピュータプログラムや簡単にプログラムに変わるなにものか）を生み出す．
2. 幅広い種類の問題を解く．
3. 問題固有の情報は最小限しか要求しない．
4. 特に最終的な解の大きさや形を前もって記述する必要がない．
5. 何らかの方法で，なじみがありかつ有益なプログラムの構成物（メモリ，繰り返し，サブルーチン，データ構造，再帰など）を実装する．
6. 前もって問題を分解する，部分ゴールを見分ける，オペレータを用意する，あるいは問題に対してシステムを一新するなどの必要がない．
7. かなり大きな問題にも対応可能である．
8. 人間のプログラマ，数学者，あるいは専門の設計者によって生成されたものと比べて遜色ないような結果を生成する．もしくはそれ自体で発表可能であるか，商業的に役に立つ結果を生成できる．
9. 十分に定義され，再現可能であり，隠れた段階が無く，実行時に人間の介入を必要としない．

本論文においても自動プログラミングという用語を用いる場合，上記のような特性を備えたシステムを意味する．

本節では，いくつかの自動プログラミングの手法について述べる．

2.2.1 GP with index memory（インデックスメモリ付きGP）

TellerはGPで状態やメモリを扱うためにインデックスメモリという手法を提案し，この手法がTuring完全であることを示した[32–34]．インデックスメモリをもつ個体は，GP木と一次元配列で表されるメモリをもつ．メモリは決められた範囲の整数値をとることができ，それらを終端記号として追加する．また，GPの関数セットにReadとWriteという非終端記号を加えることで，このメモリにアクセスを行う．ReadとWriteの処理例は次のようになる．

- (Read Y)：Memory[Y]の値を返す．

- （Write XY）: Memory[Y] の値を返し，Memory[Y] に値 X を書き込む．

メモリの数を 20，メモリのとる整数値の範囲を 20 とすると，この個体は 20^{20} 個の状態をとることができる．

2.2.2 Linear Genetic Programming（LGP）

Linear Genetic Programming（LGP）[35]は明確な線形のコンピュータプログラムを扱う．木構造を基にした LISP のような関数型プログラミング言語を用いる一般的な GP の表現の代わりに，LGP では C 言語のような手続き型のプログラムを進化させる．LGP の個体は可変の簡単な C 言語の命令列で表現される．命令はあらかじめ用意されたレジスタ r または定数 C について操作を行う．結果は目的のレジスタに格納される．LGP のプログラム例を図 2.3 に示す．LGP では図 2.3 のような手続き型のプログラムを交叉，突然変異によって進化させる．

```
void LGP(double r[8])
{
        r[0] = r[5] + 73;
        r[7] = r[0] - 59;
        r[2] = r[5] + r[4];
        r[6] = r[7] * 25;
        r[1] = r[4] - 4;
        r[7] = r[6] * 2;
}
```

図 2.3: LGP のプログラムの例

2.2.3 Grammatical Evolution（GE）

Grammatical Evolution（GE）[36–38]は任意の言語を使用できる自動プログラミングの手法である．GE では，BNF（Backus Naur Form）に従って記述された遷移規則によって遺伝子型から表現型へのマッピング（Genotype to Phenotype Mapping）を行う．GE の染色体は 8bit にエンコードされたビット列で構成され，BNF の文法ルールの適用順を示す．図 2.4 に示すような文法ルールを染色体の記述に従って適用していくことで，プログラムを作成する．交叉や突然変異などの遺伝操作は GA で用いられているものと同様である．

(A) <expr> ::= <expr> <op> <expr> (0)
| (<expr> <op> <expr>) (1)
| <pre-op> (<expr>) (2)
| <var> (3)
(B) <op> ::= + (0)
| - (1)
| / (2)
| * (3)
(C) <pre-op> ::= Sin
(D) <var> ::= X (0)
| 1.0 (1)

図 2.4: GE の文法ルールの例

2.2.4 その他の自動プログラミングの手法

上に挙げた以外にも様々な表現を用いた自動プログラミングの手法が提案されている．近年，新たな自動プログラミングの手法として PushGP [39–41] と Object Oriented Genetic Programming（OOGP）[42–44] と呼ばれる手法が提案されている．PushGP は Spector によって進化計算用に開発されたスタックベースのプログラミング言語である Push Language を進化させる．OOGP は LISP 言語の代わりに JAVA のようなオブジェクト指向型のプログラミング言語の進化を行う．両手法とも主に再帰呼び出しを必要とするような問題（例：階乗，フィボナッチ数列，ソートプログラムなど）に適用され，再帰構造を利用したプログラムの自動生成に成功している．

2.3 グラフ構造表現を用いた自動プログラミング

本節では，一般的なGPで扱うような木構造ではなく，グラフ構造で表現されたプログラムを進化的に自動生成する手法について述べる．

2.3.1 Parallel Algorithm Discovery and Orchestration（PADO）

Parallel Algorithm Discovery and Orchestration（PADO）[45–47]はグラフ構造を扱い，PADOのプログラムはいくつかのノードとスタック，インデックスメモリから構成される．図2.5はPADOの構造例を示している．各プログラムはスタックとインデックスメモリを使って中間的な情報を保持することができる．PADOの各ノードはアクション部と分岐決定部から構成され，ループの表現が容易に実現できる．PADOのプログラムは，スタートノードから開始し，各ノードを辿っていくことで処理を行い，ストップノードに達したらプログラムを終了する．PADOは主に信号分類に適用され有効性を示している．

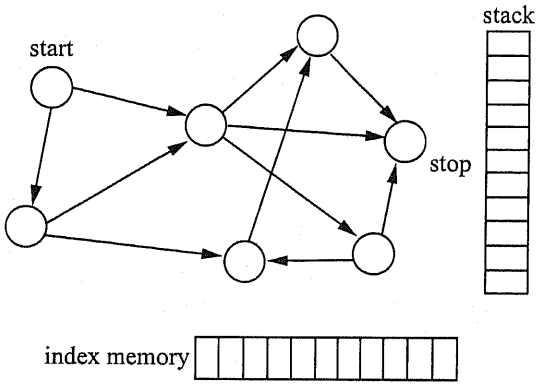


図2.5: PADOの構造例

2.3.2 Parallel Distributed Genetic Programming（PDGP）と Cartesian Genetic Programming（CGP）

Parallel Distributed Genetic Programming（PDGP）[48]とCartesian Genetic Programming（CGP）[49, 50]もグラフ構造を採用している手法である．これらの手法ではフィードフォワードなどのある程度制限されたグラフ構造を扱う．PDGPでは，遺伝操作を表現型に適用するのに対して，CGPでは遺伝子型から表現型へのマッピング（Genotype to Phenotype Mapping）を行い，遺伝操作を一次元の整数列に対して適用する．また，CGPに自動関数定義の概念を導入したEmbedded CGP（ECGP）も提案されており，通常のCGPと比べて性能が高いという実験結果が報告されている[51, 52]．

2.3.3 遺伝的ネットワークプログラミング (Genetic Network Programming; GNP)

遺伝的ネットワークプログラミング (Genetic Network Programming; GNP) [53–56]は，ノードをネットワーク状に接続することによって，プログラムの自動生成を行う手法である．GNP は，スタートノード (start node)，判定ノード (judgment node)，処理ノード (processing node) の 3 種類のノードからなり，有向グラフの構造になっている．スタートノードは，プログラムの開始位置を表す．判定ノードは定められた条件判定を行い，判定結果に従って次の遷移先を選択する．処理ノードは，定められた処理を行い，次のノードへ実行を遷移させる．これらのノード間の接続と遷移によってプログラムが生成される．GNP は主に自律エージェントの行動決定に適用され，一般的な GP よりも高い性能を示している．

2.3.4 その他のグラフ構造表現を用いた自動プログラミング

上に挙げた以外にもグラフ構造表現を用いる自動プログラミング手法が提案されている．

Linear-Graph GP [57]は LGP をグラフ構造表現に拡張したものである．Linear-Graph GP の各ノードは，“linear program” と “branching node” の 2 種類に大別される．“linear program” では演算などの処理を，“branching node” では条件分岐を行う．Linear-Graph GP ではループ構造は扱わないため，フィードフォワード型のグラフ構造となる．

遺伝的オートマトン (Genetic Automata Generation; GAUGE) [58]は，記号間にパスを張りめぐらしたグラフ構造をとり，内部状態の違いによる条件分岐を自動生成することによって問題解決を行う．GAUGE では問題に必要な状態数を進化過程で自動的に獲得するため，状態数に対する試行錯誤を必要としない．GAUGE は自律エージェントの行動決定問題に対して，インデックスメモリ付き GP などよりも性能が高いことが示されている．

2.4 自動プログラミングの画像処理への応用

本節ではGPに代表される自動プログラミングの画像処理分野への応用例について述べる．代表的な応用例として，画像分類問題や画像処理フィルタの設計への応用が挙げられる．TackettはGPを画像分類問題に適用しその有効性を検証した[59]．グラフ構造表現を用いた自動プログラミング手法であるPADOは物体認識や顔認識に適用されている[46, 47]．ZhangらはLGPを多クラスの画像分類問題に適用し，通常の木構造を用いたGPに比べて性能が高いことを示している[60]．さらに，GPを用いた画像からの特徴抽出アルゴリズムの自動生成に関する研究も行われており[61, 62]，自動生成された特徴量は人間が設計したものと同等以上であることが示されている．

ところで，一般的に画像処理において，目的とする画像変換処理を実現する画像処理フィルタの組み合わせを決定することは困難であるといえる．この問題に対して，進化計算を用いて既知の画像処理フィルタの組み合わせを最適化することで，実現したい画像変換を自動構築する手法が提案されている[63–65]．木構造状画像変換自動生成法（Automatic Construction of Tree-structural Image Transformation; ACTIT）[64, 65]はGPを用いて木構造状の画像処理フィルタを自動構築する．ユーザは処理の対象となる原画像と理想的な出力画像（目標画像），必要であれば，画像中の各画素の重要度を階調値の大きさで表した重み画像からなる教師画像セットを用意する．原画像は全ての葉ノードから入力され，各ノードでは子ノードが出力した画像を入力としてフィルタ処理を行い，親ノードへ処理画像を出力する．最終的に根ノードから処理結果が得られる．この根ノードで得られた処理結果の画像と目標画像を比較することで木構造を評価し，GPによる世代交代を繰り返すことで，有効な木構造状画像処理フィルタを獲得する．世代交代によって得られた木構造状画像処理フィルタは教師画像セットに対して有効な画像変換となっているため，これを教師画像セットと類似の未知画像に適用した際も同様の処理が行えることが期待できる．ACTITはこれまでにきず検出処理や文章画像からの手書き文字の除去，医用画像処理などの様々な画像処理の自動構築に適用され，その有効性が示されている．また，ACTITを3次元画像処理に応用した3D-ACTIT[66–72]や画像処理フィルタのパラメータを構造と同時に最適化する手法としてParameter Tunable ACTIT（PT-ACTIT）[73, 74]やGenetic Matrix Algorithm（GMA）[75]も提案されている．3D-ACTITは医用画像処理や動画像処理に対して良好な成果を挙げている．PT-ACTITやGMAは画像処理フィルタのパラメータと木構造の同時最適化に成功している．また，近年安価で取り扱いやすい汎用グラフィックスボードに搭載されているGPU（Graphics Processing Unit）を進化計算に活用する研究が活発に行われている[76–78]．文献[79]では，GPUを利用することで従来計算コストの高かったACTITの処理を高速化することに成功している．

2.5 まとめ

本章では，本研究に関連の深い従来研究として，進化計算や進化計算によるいくつかの自動プログラミング手法，自動プログラミングの画像処理への応用例について紹介した．次章以降では，本章で紹介したような自動プログラミング手法よりも複雑なプログラムや処理を自動生成することを目的とした手法の提案を行っていく．

第3章　Genetic Image Networkによる画像変換の自動構築

3.1　はじめに

現在までに様々な画像処理アルゴリズムが提案され，それらの有効性が示されている．しかし画像処理は，その取り扱う画像に強く依存している場合が多く，対象画像に依存せずに有効な処理を行う汎用的な方法は確立されていない．そこで，自動化，省力化を目的とした画像処理エキスパートシステムに関する研究が行われている．まず，サンプル図形で与えた処理要求から，画像処理手順の自動獲得を行う例として，IMPRESS [80, 81]の例が挙げられる．これは，サンプル図形で与えられたその図形特徴に応じて，あらかじめ用意された考え得るいくつかの具体的な処理手順の中から最も良い処理手順を求めるものである．また，与えられた原画像と目標画像から図形の処理手順をGAを用いて自動的に獲得する手法も提案されている [82]．この処理手順には収縮処理，膨張処理とAND+NOT，ORの構造の組合せを用いている．さらに，進化計算を画像処理に適用した例として，実現したい未知の画像変換を，既知の単純な画像処理フィルタの組合せとして表現し，GAやGPを用いて自動構築を行う手法が提案され，有効性が示されている [63]．画像処理フィルタを木構造状に組合わせることで，ユーザから提供された教師画像（原画像，目標画像）を参照し，GPを用いて画像変換の自動構築を行うACTIT [64, 65]では，複雑な画像処理を自動的に獲得することに成功している．ACTITはこれまでにきず検出処理や医用画像処理など様々な画像処理の自動構築に適用され，その有効性が示されている [66, 67]．

本章では，進化計算を用いた画像変換の自動構築に対して，画像変換の表現方法としてネットワーク構造を利用することを提案する．まず，3.2節では画像変換の表現方法としてネットワーク構造を用いるGenetic Image Network（GIN）を提案する．その後，3.3節でGINの構造をフィードフォワードネットワーク構造に制限したFeed Forward Genetic Image Network（FFGIN）を提案し，性能の評価を行う．

3.2　Genetic Image Networkの提案と評価

3.2.1　概要

一般的に，ネットワーク構造はフィードバックの表現や同じ構造の再利用，過去の情報の蓄積といった点を考えると，木構造よりも高い表現能力をもっているといえる．そこで，進化計算を用いて画像処理フィルタをネットワーク構造状に自動的に組み上げることで画像変換の自動構築を行う，GINを提案する．GINではネットワーク構造を表現形式として扱うため，木構造を含む複雑な構造の表現が可能である．さらに木構造では表現すること

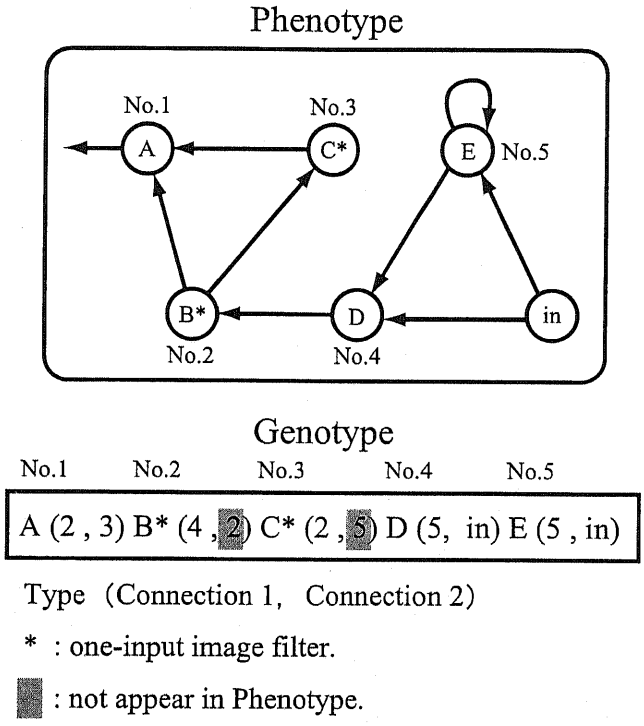


図 3.1: GIN の構造例

ができないフィードバックの表現や同じ構造の再利用，複数出力画像変換の自動構築が可能となる．ここでは，GIN を用いた画像変換の自動構築実験を行い，その性能を評価するとともに，木構造では表現できない構造の獲得ができることを確認する．

3.2.2 Genetic Image Network（GIN）

GIN の構造

GIN の構造例を図 3.1 に示す．GIN では，ネットワーク構造を扱うため，フィードバックの表現や複数出力などの任意の表現が可能である．各ノードは 1 入力または 2 入力の画像処理フィルタに対応しており，入力された画像に対して対応するフィルタ処理を行い，画像を出力する．本論文の実験では画像処理フィルタを 2 入力までに限定したが，3 入力以上の画像処理フィルタの取り扱いも原理的には可能である．

各ノードは同期的に画像の出力を行い，あらかじめ決められた回数の画像変換の後，出力部から画像を取り出す．ここではこの画像変換の回数を “ステップ数” と呼ぶこととする．画像変換の実行時に入力のないノードは出力を行わない．また，2 入力フィルタにおいて入力画像が一方からしか得られない場合は，画像変換を行わず一方から入力された画像をそのまま出力することとする．画像処理フィルタセットの中に，nop フィルタ（何もしない）を含めることで，ステップ数が多い場合でも実質の処理回数が少ない表現を実現

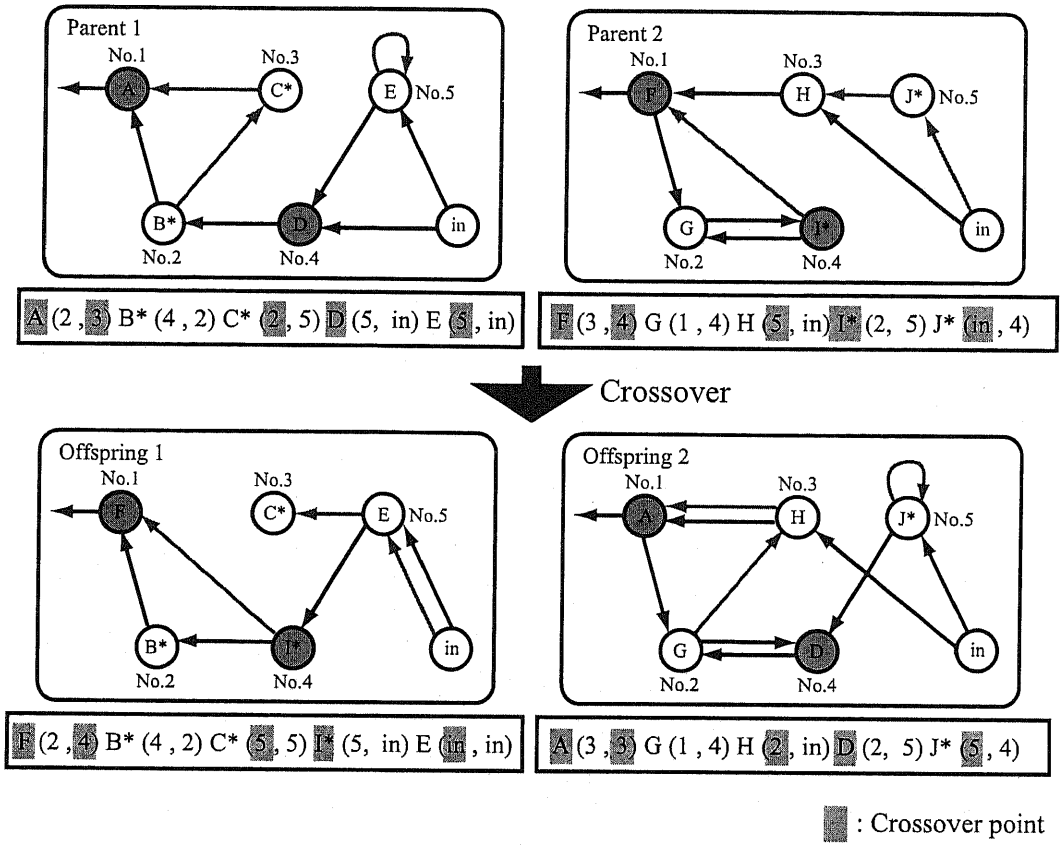


図 3.2: GIN における交叉の例

可能である．本論文の実験ではネットワークの実行方式として同期的な実行を採用したが，各ノードを順番に実行する方法なども考えられる．

図 3.1 のようなネットワーク構造を進化的な手法を用いて最適化を行う．染色体は各ノードに注目し，各ノードについて，フィルタの種類と入力元を記述していくことで表現される．対応するフィルタが 1 入力の場合は 2 つ目の接続は表現型には変換されない．ノード数は固定とするため，各個体の遺伝子型は固定長の文字列で表現される．初期個体は乱数によって生成されるが，遺伝操作を繰り返すことによって優れた個体が生成されることが期待される．

GIN における遺伝操作と世代交代モデル

GIN の各個体の遺伝子型は一次元の文字列として表現されるため，比較的簡単な遺伝操作を適用することが可能である．本論文では，遺伝操作として次のような交叉と突然変異を用いた．

- 交叉
  交叉は一次元の文字列に対する一様交叉を採用した．一様交叉では確率 P_c によって

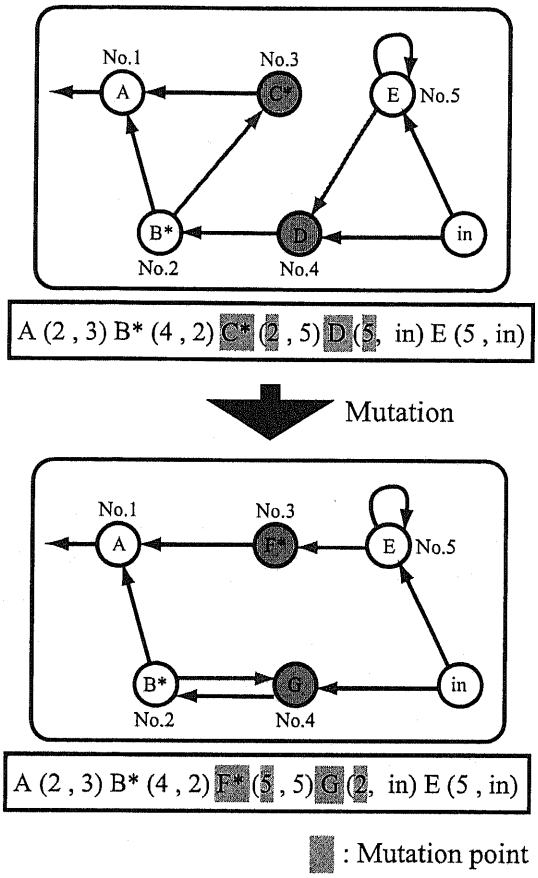


図 3.3: GIN における突然変異の例

マスクパターンを生成し交叉点を決定する．GIN における交叉の例を図 3.2 に示す．網掛けの部分が確率 P_c によって選択されたマスクパターンであり，親 2 個体の網掛け部を交換することで交叉が行われている．

- 突然変異
  突然変異は突然変異率 P_m によって遺伝子単位で発生するものとする．突然変異が起こるとその遺伝子の記号がランダムに変更される．GIN における突然変異の例を図 3.3 に示す．網掛けの部分が突然変異率 P_m によって選択された遺伝子であり，ランダムに記号が変更されている．

- 世代交代モデル
  世代交代モデルには MGG [18]を用いることとした．ただし，親個体と子個体の中から次世代に生存させる個体を，適応度の最も高い 1 個体と，トーナメント選択によって選択された 1 個体とすることとした．これは，選択の際に適応度の値の影響を排除するためである．

従来手法との相違点

ここではGINと従来手法との相違点を述べる．まず，木構造を扱うACTITとは画像処理の表現形式の点で異なり，構造的にGINはACTITを包含する表現が可能である．次に，GINではノード間の接続に制限がないため，ある程度構造を制限しているPDGPやCGPよりも複雑な表現が可能である．また，GINは同期的に各ノードを実行し出力部から処理画像を取り出すため，PADOやGNP，GAUGEといった手法と実行方法において異なる．さらに，表現型から遺伝子型への変換を行うことによって遺伝操作を遺伝子型に対して行うため，GPに潜在的に存在するブロートなどの問題を回避できると考えられる．

3.2.3 GINによる画像変換の自動構築

実験の設定

今回の実験で使用したパラメータ値を表3.1に示す．GINの実行時のステップ数は5，10，15，20の4パターンを用いてそれぞれ実験を行った．実験に使用した画像処理フィルタを付録Aに示す．本実験では1入力1出力フィルタ27種類，2入力1出力フィルタ11種類を用意した．これらは画像処理における基礎的かつ重要なフィルタとして筆者らが選択した．

各個体の評価関数には式3.1を用いた．この評価関数は原画像を変換して得られた出力画像と，原画像に対して手動などの手段によってあらかじめ作成した目標画像との差分を算出するものである．

$$fitness = \frac{1}{N}\sum_{n=1}^{N}\left\{1 - \frac{\sum_{i=1}^{W}\sum_{j=1}^{H} w_{ij}^{n}|o_{ij}^{n} - t_{ij}^{n}|}{V_{max}\sum_{i=1}^{W}\sum_{j=1}^{H} w_{ij}^{n}}\right\} \tag{3.1}$$

ここで，o_{ij}^{n}は出力画像の画素値，t_{ij}^{n}は目標画像の画素値，w_{ij}^{n}は重み画像の画素値であり，i，j方向の画素数をW，Hとする．Nは教師画像セット数，V_{max}は最大階調値である．重み画像は画素ごとの重要度を示すもので，最大を1.0，最小を0.0として目標画像に併せて用意する．なお，重み画像を用いないことも可能であり，その場合は全画素を均一に扱う（$w_{ij} = 1$）．式3.1から個体の評価値である適応度が算出される．適応度は[0.0, 1.0]の範囲で与えられ，1.0に近いほど優良な画像変換だといえる．

実験1：細胞壁の抽出処理の自動構築

本実験では図3.4に示す2種類の教師画像セット（原画像，目標画像，重み画像）を用いて，ACTITとGINによって画像変換の自動構築実験をそれぞれ行う．実験は表3.1のパラメータ値を用いて同一の条件で，GINの各ステップ数とACTITについてそれぞれ10回づつ行った．ACTITのパラメータは交叉，突然変異の発生確率をそれぞれ0.9，最大ノード数を20として，その他のパラメータについては表3.1と同一のものを用いた．画像変換の目的は画像中の細胞壁の部分を抽出することである．画像の画素数は128×128[pixels]，$V_{max} = 255$である．

表 3.1: GIN による画像変換の自動構築実験に用いた各パラメータ値

世代交代モデル	MGG
世代数	5000
個体数	150
MGG の子個体数	50
交叉率	0.9
一様交叉の交叉率（P_c）	0.1
突然変異率（P_m）	0.03
トーナメントサイズ	5
ノード数	20
ステップ数	5, 10, 15, 20

図 3.5 は 10 回試行の平均適応度の推移を示したものである．両手法とも世代数を重ねるにつれて適応度が上昇しているのがわかる．ACTIT に比べて GIN は初期段階における適応度の上昇が小さいが，これは ACTIT より GIN のほうが表現できる構造が多いため探索空間が大きいことによるためと考えられる．両手法で獲得した最も良い適応度は，それぞれ 0.9327（ACTIT），0.9331（GIN）でほぼ同等であり，このときの GIN のステップ数は 15 であった．ACTIT と GIN によって変換された出力画像を図 3.6 に示す．両手法とも細胞壁の抽出処理という画像変換を自動的に獲得できていることを確認することができる．

次に，実験によって得られた木構造状あるいはネットワーク構造状の画像処理フィルタを，教師画像に類似した未知画像に対して適用した結果を検証する．未知画像とそれに対する ACTIT と GIN の処理結果を図 3.7 に示す．ACTIT，GIN ともに良好な画像変換が行われていることがわかる．つまり，両手法とも細胞壁の抽出処理という画像変換を自動的に構築することができたといえる．

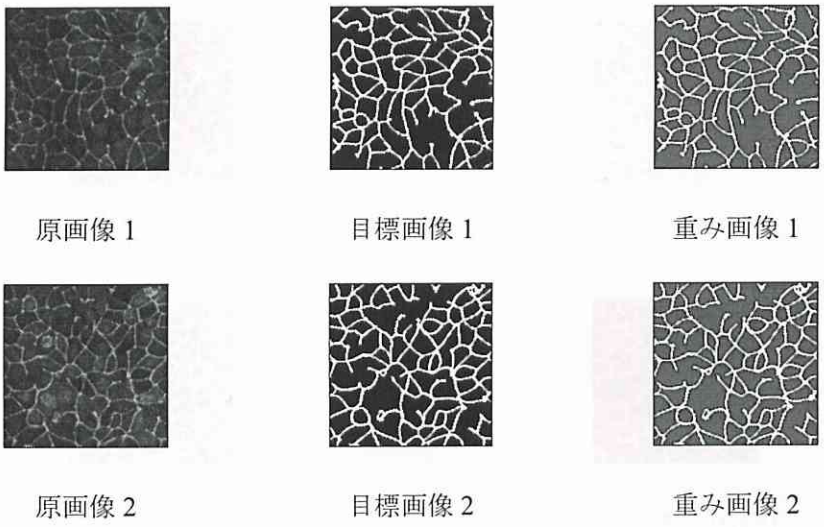


図 3.4: 実験 1 で用いた教師画像セット（細胞壁の抽出）

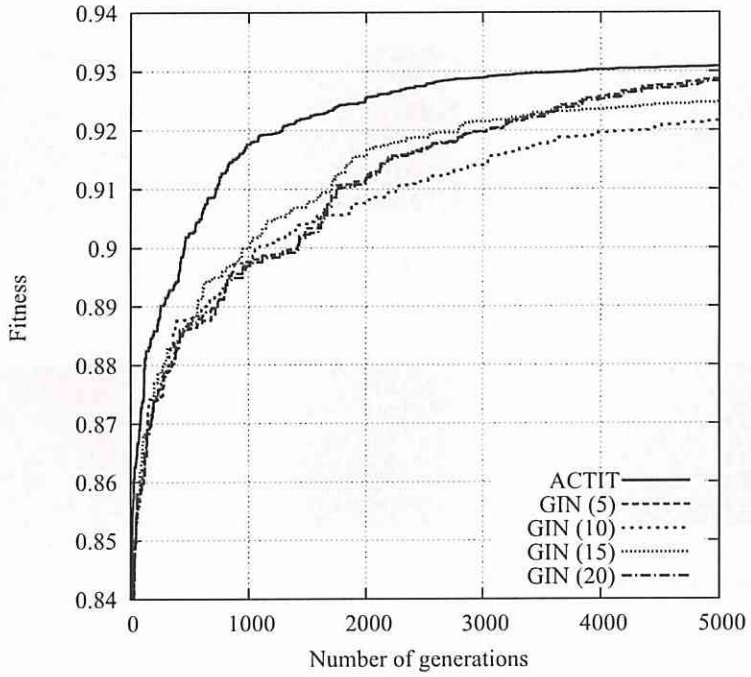
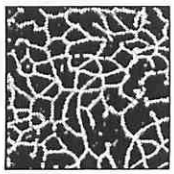
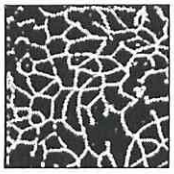


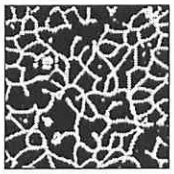
図 3.5: 適応度の推移（細胞壁の抽出）



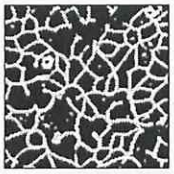
出力画像 1 (ACTIT)



出力画像 1 (GIN)

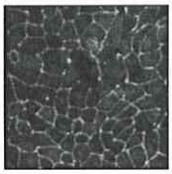


出力画像 2 (ACTIT)



出力画像 2 (GIN)

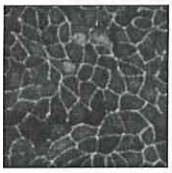
図 3.6: ACTIT と GIN による出力画像（細胞壁の抽出）



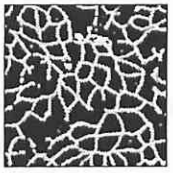
未知画像 1

ACTIT の未知画像 1
に対する出力画像

GIN の未知画像 1
に対する出力画像



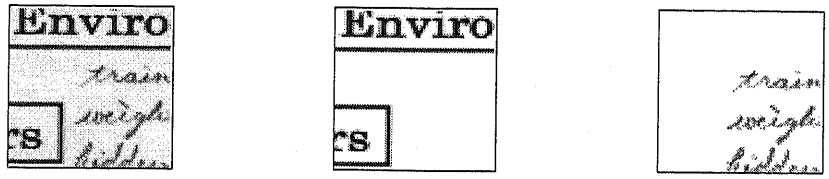
未知画像 2



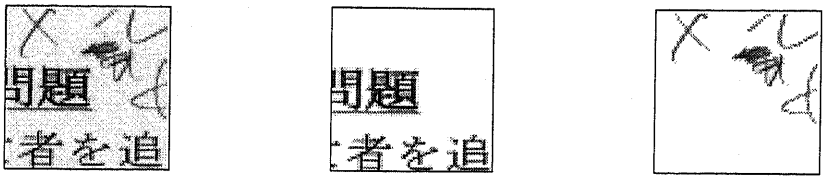
ACTIT の未知画像 2
に対する出力画像

GIN の未知画像 2
に対する出力画像

図 3.7: 未知画像に対する ACTIT と GIN の出力画像（細胞壁の抽出）



原画像 1　　目標画像 1 (a)　　目標画像 1 (b)



原画像 2　　目標画像 2 (a)　　目標画像 2 (b)

図 3.8: 実験 2 で用いた教師画像セット（複数出力画像変換）

実験 2：複数出力画像変換の自動構築

本実験では提案手法である GIN を用いて複数出力画像変換の自動構築を行う．GIN ではネットワーク構造を表現形式としているため複数出力の表現が可能である．複数出力表現は木構造を扱う ACTIT では表現することはできないため，GIN の大きな利点の 1 つである．本実験で用いた教師画像セットを図 3.8 に示す．画素数は 64×64[pixels]，$V_{max} = 255$ である．画像処理の目的は，手書き文字と印刷文字からなる画像から，“手書き文字除去” という処理と “印刷文字除去” という 2 つの処理を 1 つのネットワークで同時に自動獲得することである．この画像では文字除去の部分が比較的明瞭であるため，重み画像は用いなかった．GIN の実行時のステップ数は 10，15，20 を用いて，それぞれ 10 回の試行を行った．各種パラメータ値は表 3.1 に示したものを用いた．

図 3.9 は 10 回試行の平均適応度の推移を示したものである．GIN が獲得した処理の教師画像に対する出力画像の一例を図 3.10 に示す．このときの適応度は 0.9968，ステップ数は 10 である．“手書き文字除去” という処理と “印刷文字除去” という 2 つの処理を実現できていることを確認することができる．

次に実験によって得られたネットワーク構造状の画像処理フィルタを，教師画像に類似した未知画像に対して適用した結果を検証する．未知画像とそれに対する GIN の処理結果を図 3.11 に示す．GIN の画像変換は 2 つの処理に対して良好な結果を示しているが，一部の残すべき文字が消えている結果となった．

GIN によって構築されたネットワーク構造を図 3.12 に示す．各ノードの記号は付録 A のフィルタの記号と対応している．構築された構造を見ると，フィードバック構造や多出力のノードが現れていることがわかる．これによって処理された画像を再利用する構造となっている．また，同一のネットワーク上に 2 つの出力ノードがあり，処理された画像を両処理において利用している．このことから表現上は非常にコンパクトでありながら，各

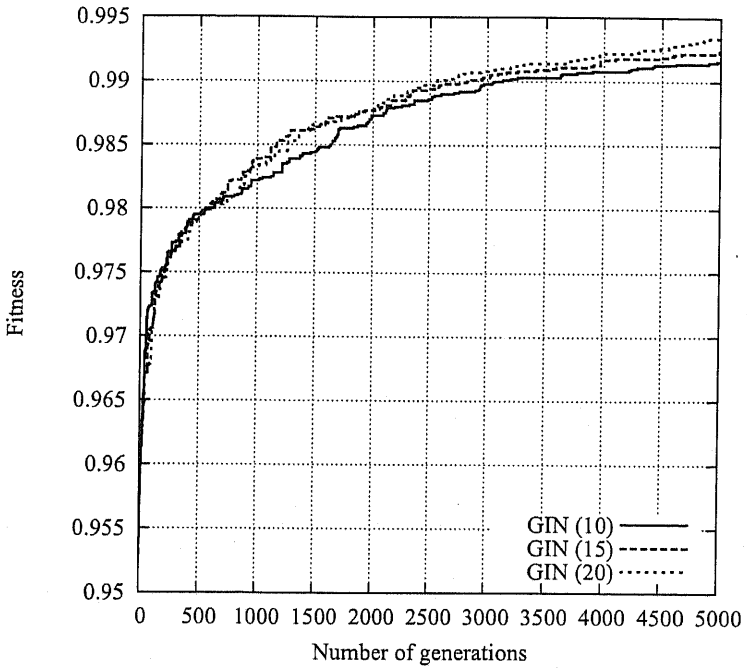


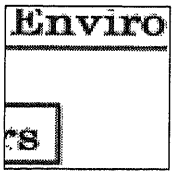
図 3.9: 適応度の推移（複数出力画像変換）

処理の実際の処理内容はきわめて複雑なものとなっていることがわかる．このようなネットワーク構造を用いた表現は従来手法の ACTIT では獲得することはできない．

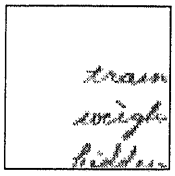
ACTIT で今回構築した処理を実現しようとした場合，木を 2 つ構築しなければならない．そこで，図 3.12 のネットワーク構造で行われている処理をステップ数 10 ということを考慮して木構造で表現すると，図 3.13 のような 2 つの木構造で表すことができる．ここで，図 3.12 の “s” から “U” へのフィードバック結合はステップ数 10 では処理に影響を与えないため，木構造には現れていない．図 3.13 から，ネットワーク内で行われている処理が非常に複雑なものであることがわかる．

次に，GIN が自動構築した画像変換についての考察を行う．図 3.13 には各画像を入力したときの処理の中間画像が示されている．まず “手書き文字除去” では，“最小値フィルタ”，“2 値化”，“小さい領域の削除”，“収縮処理” などの処理を行うことで印刷文字の部分だけを覆うようなマスク画像を作成し，原画像に近い画像と論理和を取ることで目的の処理を実現している．その際，未知画像に対しては，右側の処理系列において印刷文字の一部分が削除されてしまったため，結果の出力画像においても一部分が消えたしまったと考えられる．“印刷文字除去” においても，印刷文字だけを覆うようなマスク画像を使用して，“代数積”，“大きい領域の削除” という処理を施すことで “印刷文字除去” という処理を実現している．

ネットワーク構造で表現されたコンパクトな構造は，共通の処理プロセスを生成しやすいため，自動構築された画像処理アルゴリズムの構造の理解や汎用化にメリットがあると考えられる．また，フィードバックや変換画像の再利用によって同じ処理系列が繰り返して現れやすいということも GIN の特徴の一つである．



出力画像 1 (a)

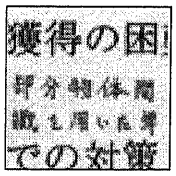


出力画像 1 (b)

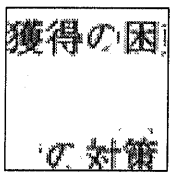
出力画像 2 (a)

出力画像 2 (b)

図 3.10: GIN による出力画像（複数出力画像変換）



未知画像



出力画像 1

出力画像 2

図 3.11: 未知画像に対する GIN の出力画像（複数出力画像変換）

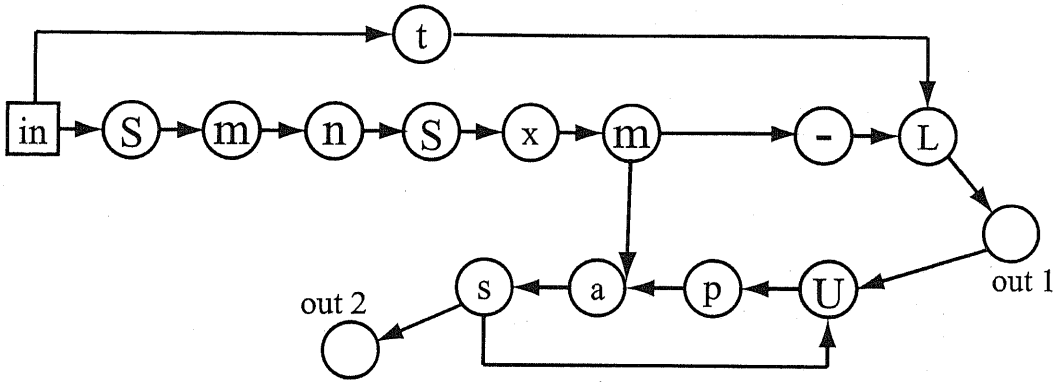


図 3.12: GIN によって自動的に構築された構造

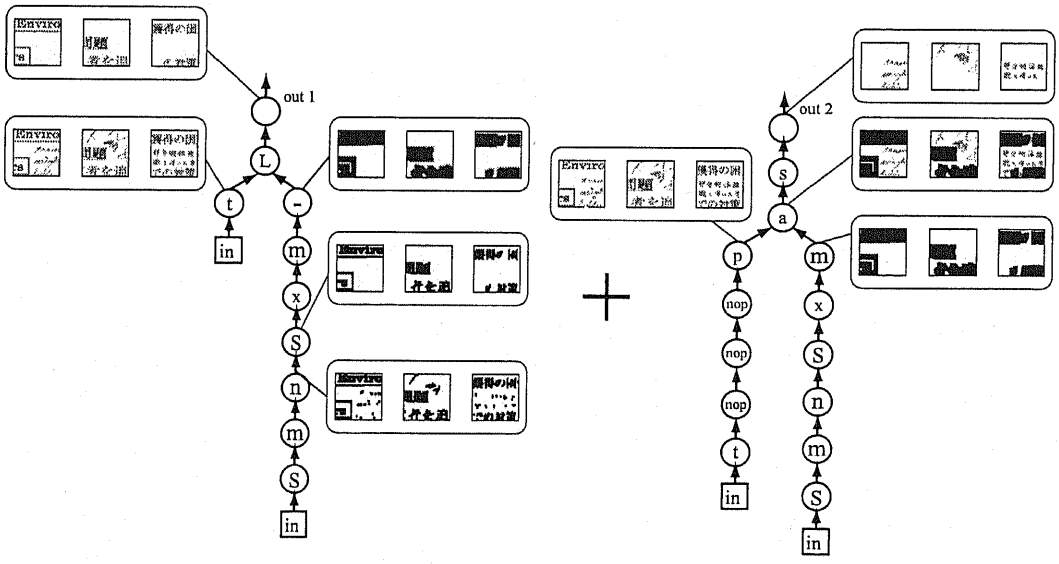


図 3.13: 図 3.12 の木構造による表現

図 3.14: 図 3.7 の未知画像 1，2 に対する目標画像と重み画像（細胞画像）

表 3.2: 細胞画像の実験における各評価値の比較

教師画像セットに対する評価値	0.9331
未知画像に対する評価値	0.9303
未知画像を教師画像とした場合の評価値	0.9359

評価関数による画像変換の評価

ここでは GIN によって獲得した画像変換の定量的な評価を式 3.1 を用いて行う．まず細胞壁の抽出処理の自動構築実験において，図 3.4 に示した目標画像と重み画像，図 3.6 に示した出力画像から式 3.1 によって評価値を算出する．これを “教師画像セットに対する評価値” とする．さらに，図 3.7 の未知画像 1，2 についても評価値を算出するために，図 3.4 に示した目標画像と重み画像の作成と同じ手順で教師画像セットを作成した．図 3.7 の未知画像 1，2 に対する目標画像と重み画像を図 3.14 に示す．図 3.14 に示した目標画像と重み画像を用いて，図 3.7 の未知画像に対する出力画像の評価値を算出し，これを “未知画像に対する評価値” とする．次に，図 3.7 の未知画像 1，2 とその目標画像，重み画像が既知であると想定して，これらを教師画像セットとして GIN による画像変換の自動構築を行った．各パラメータは先の実験と同様に表 3.1 に示したものを用い，同一パラメータで 10 回の試行を行った．ステップ数は先の実験で最も良い結果を示した 15 とした．その結果，得られた画像変換の評価値を “未知画像を教師画像とした場合の評価値” とする．以上のような方法で求めた各評価値を表 3.2 に示す．表 3.2 から，いずれの評価値も同程度であり GIN によって自動構築された画像変換に汎用性があることを確認することができる．

未知画像の目標画像 (a)

未知画像の目標画像 (b)

図 3.15: 図 3.11 の未知画像に対する目標画像（文字除去）

表 3.3: 複数出力画像変換（文字除去）の実験における各評価値の比較

教師画像セットに対する評価値	0.9968
未知画像に対する評価値	0.9907
未知画像を教師画像とした場合の評価値	0.9959

次に，複数出力画像変換の自動構築実験においても，同様の手順で各評価値の算出を行った．未知画像の評価値を算出するために用いた目標画像を図 3.15 に示す．図 3.11 の未知画像と図 3.15 の目標画像を用いて，GIN による画像変換の自動構築を行い，“未知画像を教師画像とした場合の評価値” を算出した．それぞれの評価値をまとめたものを表 3.3 に示す．表 3.3 から，“未知画像に対する評価値” が他の 2 つの評価値に比べてやや低いが，いずれの評価値も 0.99 を超えており高い値を示した．

両実験について，“教師画像セットに対する評価値”，“未知画像に対する評価値”，“未知画像を教師画像とした場合の評価値” のいずれも同程度の高い評価値を示していることから，本手法で獲得した画像変換には汎用性があると考えられる．

3.3 Feed Forward Genetic Image Networkの提案と評価

3.3.1 概要

3.2節で提案したGINではネットワーク構造を採用することで，複数出力などの複雑な画像変換の自動構築に成功している．しかし，GINでは出力画像の評価のためにステップ数をあらかじめ定める必要があり，ステップ数が少ないと十分な画像変換が行えず，大きすぎると表現が冗長になるという問題点がある．ここでは，先に提案したGINの構造をフィードフォワードネットワークに制限したFFGINの提案を行う．FFGINでは構造をフィードフォワードに制限しているため，ステップ数などのパラメータが必要ない．ここでは，FFGINとGINを用いて画像変換の自動構築を行い，各手法の性能を評価する．

3.3.2 Feed Forward Genetic Image Network（FFGIN）

FFGINではフィードバック結合のない構造を自動的に構築する．図3.16はFFGINの表現型（フィードフォワードネットワーク構造）と遺伝子型（表現型を表す文字列）の例を示している．FFGINの各ノードはあらかじめ用意された画像処理フィルタに対応している．各ノードにはあらかじめ番号が割り当てられており，自身より小さい数字の割り当てられたノードとだけ接続することができる．そのため，構造はフィードフォワードネットワークに制限される．フィードフォワードネットワーク構造の利点は，ノードの再利用が可能である点や，複数出力の表現が可能である点，GINで必要だったステップ数の指定が必要ない点などが挙げられる．FFGINでは図3.16のようなフィードフォワードネットワーク構造を進化的な手法を用いて最適化を行う．染色体は各ノードに注目し，各ノードについて，フィルタの種類と入力元を記述していくことで表現される．その際，ノードの接続に関してはフィードフォワードネットワーク構造を守るかたちで記述される．ノード数は固定とするため，各個体の遺伝子型は固定長の文字列で表現される．ノード数は固定だが，接続の状態によって使用されるノード（active node）と使用されないノード（inactive node）が存在するため表現上はノード数は可変となる．遺伝子長は，$N_{node} * (n_{in} + 1) + N_{out}$ である．ここで，N_{node} はノード数，n_{in} はあらかじめ用意した画像処理フィルタの最大入力数，N_{out} は出力ノード数である．FFGINの各個体の遺伝子型は，GINと同様に一次元の文字列として表現されるため，比較的簡単な遺伝操作を適用することが可能である．FFGINの遺伝操作についてもGINと同様に一様交叉と遺伝子に対する突然変異を用いた．ただし，突然変異は構造の制限を守るかたちで起こることとする．

3.3.3 FFGINによる画像変換の自動構築

ここでは，FFGINを用いて画像変換の自動構築を行うとともに，FFGINとGIN，ACTITの性能を比較する．

実験の設定

本実験では図3.17に示す4種類の教師画像セット（原画像，目標画像）を用いた．これらの画像は，進化計算最大の国際会議であるGenetic and Evolutionary Computation Conference

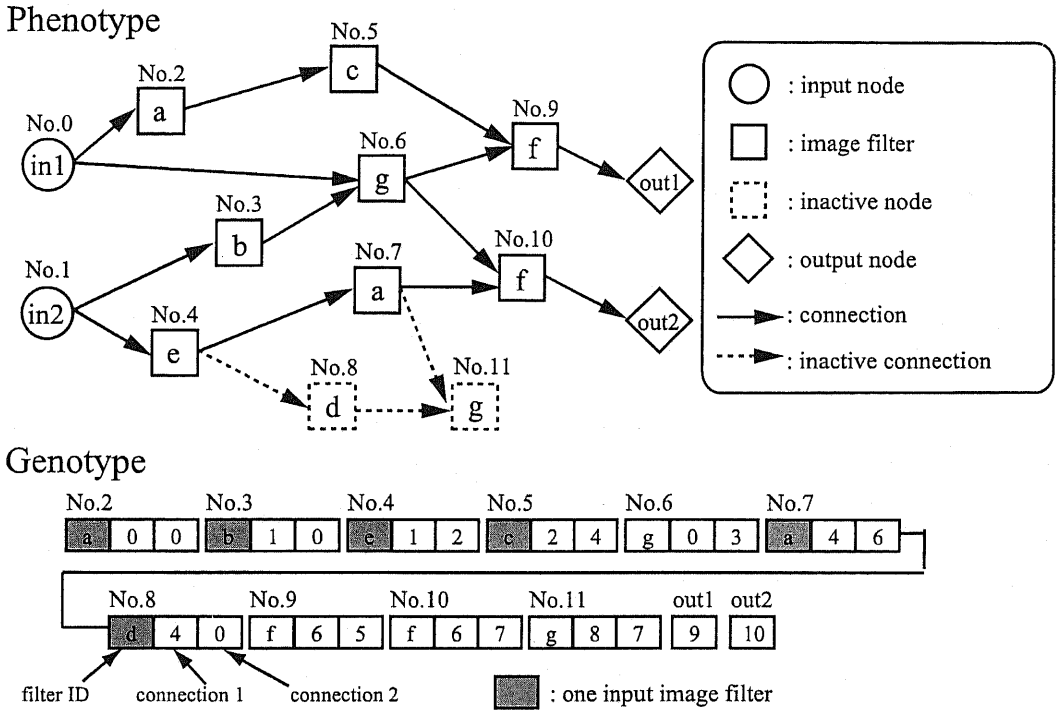


図 3.16: FFGIN の表現型の構造と接続，フィルタ番号を示した遺伝子型の例

2006 (GECCO '06)[†]で行われた competition の "pasta segmentation problem" で使用された画像である．この問題は，ランダムに置かれた様々なパスタの領域を検出するアルゴリズムを進化的に獲得する，というものである．用意されている画像には様々な照明条件のものや，小さなパスタが混入しているものなどがあり（これらのパスタは背景とラベル付けされなければならない），これらが問題を難しくしている．本実験では画像を全てグレースケールに変換して使用しており，画像サイズは 128×96[pixels] である．各個体の評価関数には式 3.1 に示した式を用い，重み画像は使用しないこととした．本実験で使用した各手法のパラメータ値を表 3.4 に示す．各手法で共通のパラメータは同一の値を使用している．画像処理フィルタには付録 A に示した 1 入力 1 出力フィルタ 27 種類，2 入力 1 出力フィルタ 11 種類を用いた．実験は同一のパラメータで乱数のシードを変えて 10 回の試行を行った．

実験の結果と考察

図 3.18 は FFGIN で獲得された画像変換の教師画像に対する出力画像である．目的とするパスタ部分の抽出処理が獲得できていることが確認できる．この画像変換の評価値は 0.9676 であった．出力画像，評価値の面から FFGIN によって目的に対して有効な画像変換が自動的に獲得できているといえる．

[†]http://cswww.essex.ac.uk/staff/rpoli/GECCO2006/pasta.htm

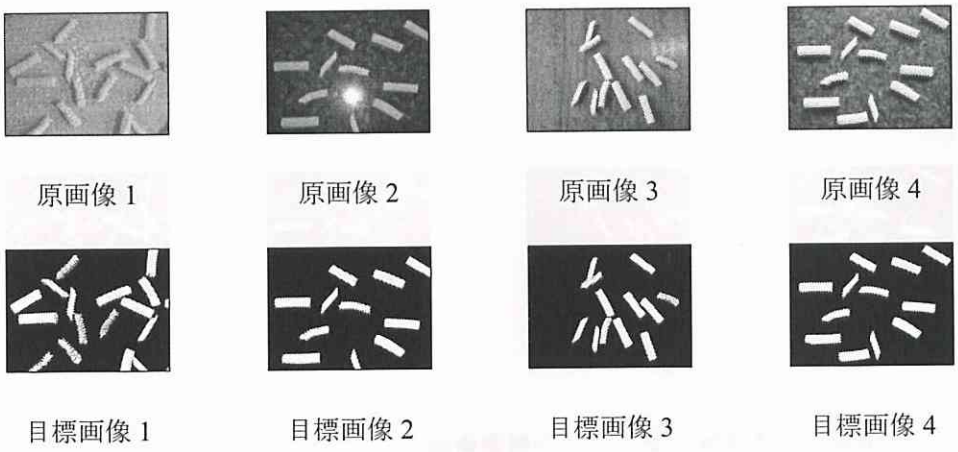


図 3.17: Pasta segmentation problem の実験で用いた教師画像セット

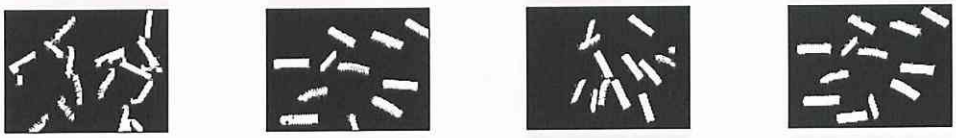
表 3.4: Pasta segmentation problem の実験で用いた各手法のパラメータ値

	FFGIN	GIN	ACTIT
世代交代モデル	MGG	MGG	MGG
世代数	5000	5000	5000
個体数	150	150	150
MGG の子個体数	50	50	50
交叉率	0.9	0.9	0.9
一様交叉の交叉率（P_c）	0.1	0.1	N/A
突然変異率（P_m）	0.03	0.03	0.9*
トーナメントサイズ	5	5	5
最大ノード数	50	50	50
ステップ数	N/A	10	N/A

*ACTIT の突然変異率は選択個体に対して
突然変異が起こる確率

図 3.19 は今回の実験で FFGIN によって獲得された構造の一例である．各ノードの名称は付録 A のフィルタ名と対応している．獲得した構造はネットワーク中でノードの再利用を行っていることがわかる．図 3.19 のような構造は木構造を表現形式としている ACTIT では構築することができず，FFGIN 特有の構造である．

次に，FFGIN によって獲得された画像変換アルゴリズムを図 3.20 に示した 4 種類の未知画像に適用する．結果の出力画像の例も同様に図 3.20 に示されている．図 3.20 の出力画像から，FFGIN によって獲得された画像変換アルゴリズムは，未知画像に対してもパスタを抽出するという理想的な処理を実現することができている．つまり，FFGIN によって汎用的な画像処理アルゴリズムが自動的に獲得されたといえる．



出力画像 1　出力画像 2　出力画像 3　出力画像 4

図 3.18: FFGIN で獲得された画像変換の教師画像に対する出力画像

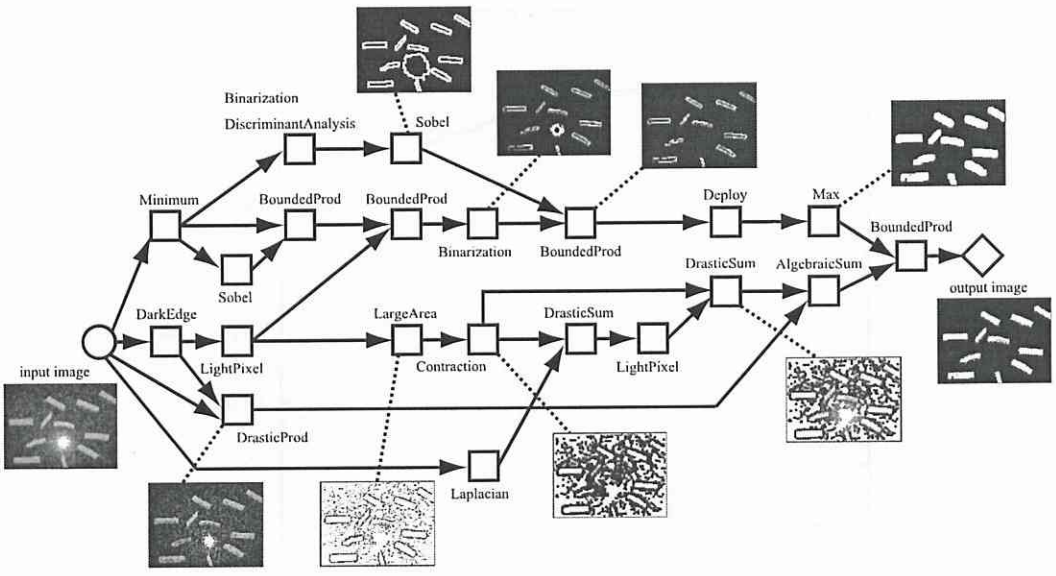


図 3.19: FFGIN によって獲得された構造の例

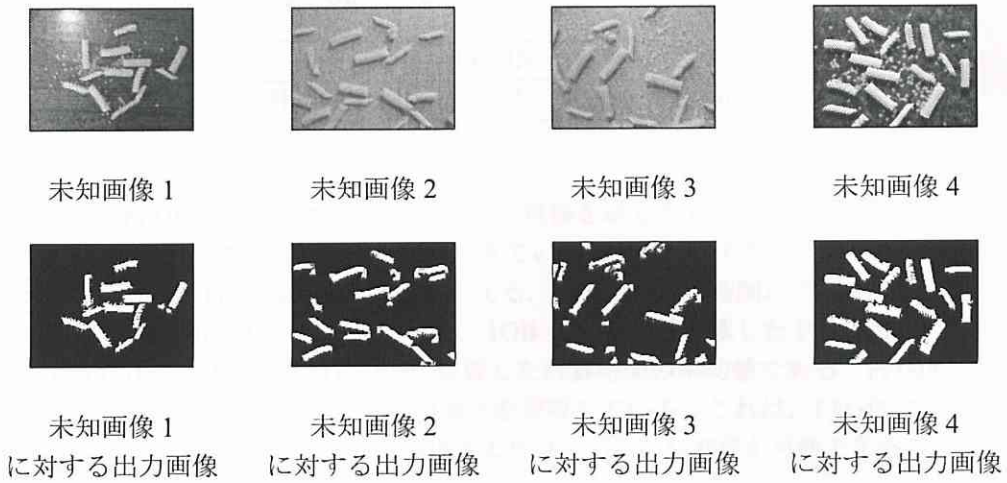


図 3.20: Pasta segmentation problem の実験で用いた未知画像と FFGIN によって獲得された画像変換アルゴリズムを適用した際の出力画像

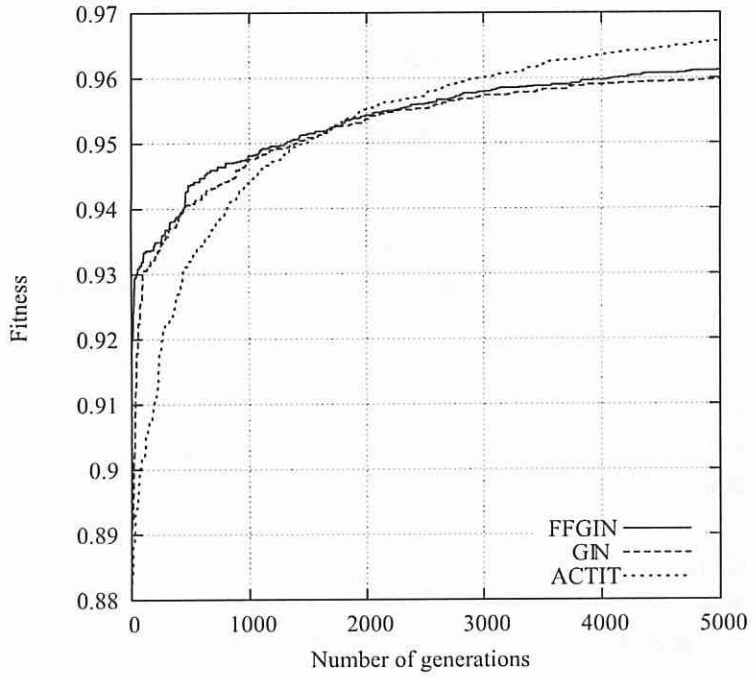


図 3.21: FFGIN，GIN，ACTIT の適応度の推移（10 回の試行の平均）

表 3.5: FFGIN，GIN，ACTIT の計算時間（10 回の試行の平均）

	FFGIN	GIN	ACTIT
計算時間 (秒)	5427	42620	28130

図 3.21 は FFGIN，GIN，ACTIT の適応度の推移を示したものである．このグラフから，すべての手法において高い評価値が獲得できていることが確認できる．つまり，FFGIN の性能は GIN や ACTIT と同程度であるといえる．最後に，計算時間についての考察を行う．本実験は CPU に Intel Core 2 Duo E6400，1GB のメモリを搭載した PC を用いて行った．表 3.5 は FFGIN，GIN，ACTIT の進化に要した計算時間の平均値である．FFGIN は GIN のおよそ 8 倍，ACTIT のおよそ 5 倍の速さを実現している．これは，FFGIN ではブートが起こらないことや，ノードの再利用によりコンパクトな表現が可能であることなどが原因であると考えられる．

3.4 まとめ

本章では，まず画像処理フィルタをネットワーク構造状に自動構築する GIN を提案し，その有効性を検証した．従来手法である ACTIT と比較して，1 出力画像変換の自動構築においては同程度の性能をもつことを示した．また GIN ではネットワーク構造を採用していることから，木構造よりも自由度の高い表現が可能である．そこで GIN を従来の木構造では表現することができない 2 出力画像変換の自動構築への適用を行い，ACTIT では獲得することができない複数出力の画像変換の自動構築が可能であることを示した．GIN によって構築された構造はフィードバックなどを含むネットワーク特有のものであり，共通のプロセスを含むユニークな構造であった．

次に，GIN の構造をフィードフォワードネットワーク構造に制限した FFGIN の提案を行い，画像変換の自動構築実験によってその有効性の確認を行った．実験の結果から，FFGIN の性能は GIN や ACTIT と同程度であったが，計算時間が GIN や ACTIT と比較して短縮されることが確認された．

本章の実験で用いた画像処理フィルタはグレースケールに対するものだけであったため，対象はグレースケール画像に限定されていた．しかし，カラー画像対応の画像処理フィルタを用意することや，原画像を色相，彩度，明度成分などに分割し，入力画像として使用する方法などを採用することで，本手法を用いてカラー画像処理の自動構築が可能になると考えられる．

GIN はネットワーク構造という特性上，過去の情報をネットワーク内に蓄積することが可能である．そのため，入力画像を時系列に沿って変化されることで，動画像に対して過去の情報も考慮した画像変換が表現できる．この特性を利用し，動画像への適用についても検討を行うことが今後の課題として挙げられる．

第4章　Genetic Image Networkに基づく画像分類法

4.1　はじめに

画像分類は製品検査や医用画像，セキュリティなどの様々な分野で必要とされる重要な技術である．現在までに画像分類に対して様々なアプローチが試みられているが，画像特徴量の選択，抽出は困難な問題の一つである．なぜなら，適切な画像特徴量は対象とする問題の画像に強く依存するからである．そこで本章では第3章で提案したGINを基に，画像変換部，特徴量抽出部，演算部から構成される画像分類器の自動構築法を提案する．提案手法では画像変換によって画像を分類し易いかたちに変換することができ，必要な特徴量を選択することができる．つまり，対象とする問題に合った特徴量を生成することができると考えられる．

4.2　Genetic Image Network for Image Classification（GIN-IC）

4.2.1　概要

ここでは，画像変換の自動構築手法であるGINを画像分類器へ拡張したGenetic Image Network for Image Classification（GIN-IC）について述べる．GIN-ICの画像分類器は画像変換部，特徴量抽出部，演算部から構成される．GIN-ICの最大の利点は画像変換部にあり，画像変換によって画像分類に必要な画像特徴を強調することが期待される．つまり，GIN-ICではノードの組み合わせによって適切な特徴量を生成，選択することができると考えられる．

4.2.2　GIN-ICの構造

本章の提案手法であるGIN-ICの構造と遺伝子型（構造を表す文字列）の例を図4.1に示す．GIN-ICではフィードフォワード型のネットワーク構造を採用しており，ノードの再利用が可能である．GIN-ICのノードは入力ノード，画像変換ノード，特徴量抽出ノード，演算ノード，出力ノードの5種類に分類される．入力ノードは原画像に対応している．画像変換ノードでは対応する画像処理フィルタによる画像変換が行われる．特徴量抽出ノードでは，入力画像から対応する特徴量を抽出する．演算ノードにおいては，入力値に対して対応する演算を施して出力する．画像の分類は出力ノードの値を用いて行われる．GIN-ICによる画像分類の処理は，原画像に対して画像変換を施し，画像から特徴量を抽出した後，

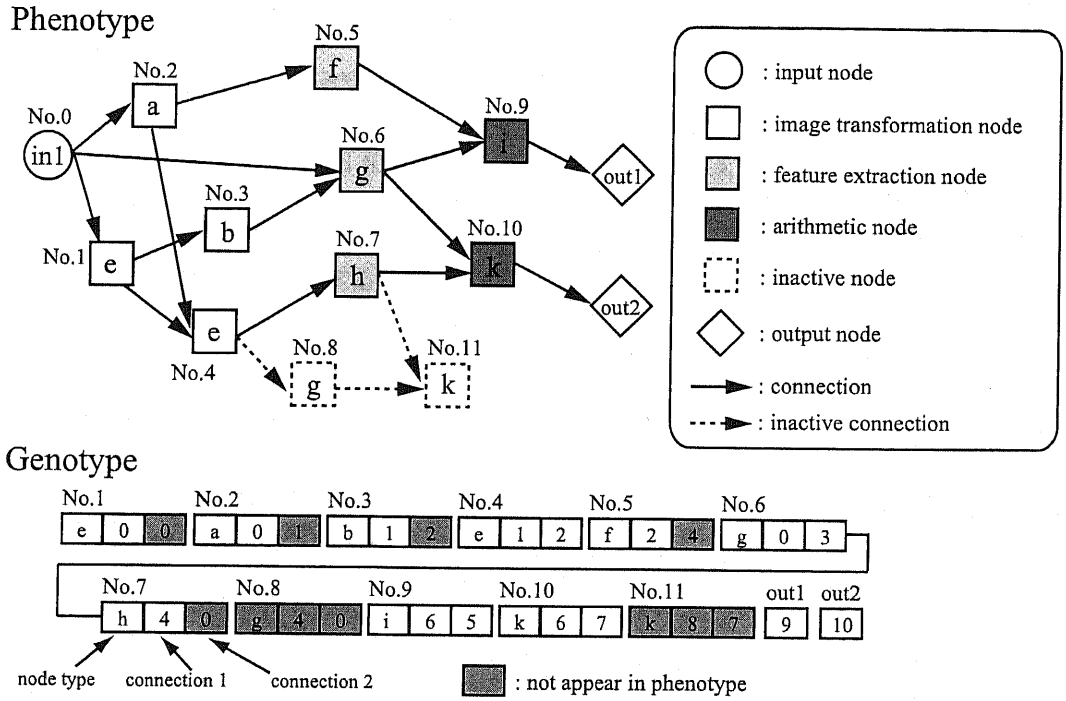


図 4.1: GIN-IC の構造とその遺伝子型の例

数値演算を行い分類を行う，という流れになる．この一連の処理過程を進化計算を用いて自動構築する．

GIN-IC では，図 4.1 のようなフィードフォワード型のネットワーク構造を進化計算を用いて自動構築する．各ノードにはあらかじめ番号が振られているものとし，入力ノード，画像変換ノード，特徴量抽出ノード，演算ノード，出力ノードの順に番号が割り当てられる．ここで，フィードフォワード構造を守るため入力元として選択できるノードを自分のノード番号より小さい番号に制限する．また，画像変換ノードの入力元は入力ノードまたは画像変換ノード，特徴量抽出ノードの入力元も同様に入力ノードまたは画像変換ノード，演算ノードの入力元は特徴量抽出ノードまたは演算ノードに制限する．これにより，実行することができない構造が発生しないようにする．

GIN-IC ではフィードフォワード型のネットワーク構造を一次元の文字列に変換し，その文字列に対して遺伝操作を施す．染色体の文字列では，各ノードについて，ノードの種類と入力元を記述していくことで表現される．対応するノードが n 入力のノードの場合は n+1 つ目以降の接続に関する遺伝子は表現型には変換されない．各ノード数を固定とするため，各個体の遺伝子型は固定長の文字列で表現される．各ノードの数は固定であるが，接続の状態によって使用されるノード（active node）と使用されないノード（inactive node）が現れるため表現上はノード数は可変となる．図 4.1 の例では，No.8 と 11 のノードが inactive node である．

4.2.3 GIN-IC の遺伝操作と世代交代モデル

GIN-IC の各個体の遺伝子型は GIN や FFGIN と同様に一次元の文字列として表現されるため，比較的簡単な遺伝操作を適用することが可能である．本章の実験では世代交代モデルとして進化戦略の (1+4)ES を採用し，遺伝操作には突然変異だけを用いることとした．突然変異は突然変異率 P_m によって遺伝子単位で発生するものとする．突然変異が起こるとその遺伝子の記号がランダムに変更される．その際，接続先については構造上の規則を守る接続に変更するものとする．本章の実験で用いる (1+4)ES の処理手順は次に示す通りである．

1. 世代数 $t = 0$ とする．親個体 M としてランダムに 1 個体を生成する．
2. 親個体 M に突然変異操作を施し，子個体 C を 4 個体生成する．
3. $M + C$ の個体の中から最良個体を 1 個体選択する（最良個体が複数存在した場合は，子個体を優先して選択する）．その後，親個体 M を選択した個体と置き換える．
4. 終了条件を満たしていれば終了し，そうでなければ，$t = t + 1$ としてステップ 2 へ戻る．

GIN-IC では接続状態によって使用されないノードが存在するため，遺伝操作によって適応度の変化が起こらない場合が存在する（これを neutrality と呼ぶ [49, 83–85]）．手順 3 で，最良個体が複数存在した場合は子個体を優先して選択するため，適応度が上昇しなかった場合でも探索点の移動が起こりうる．これによって，個体数の少ない単純な世代交代モデルでも効率的な探索が行えると考えられる．また，GIN-IC と同様にフィードフォワード型の表現をもつ CGP においてもこの戦略は採用されており，実験的にその有効性が示されている [49]．

4.3 テクスチャ画像の分類問題への適用

本節では GIN-IC を多クラスのテクスチャ画像の分類問題に適用し，有効性の検証を行う．

4.3.1 実験の設定

本実験では GIN-IC を用いて多クラスのテクスチャ画像の分類器を構築する．Vision Texture [†] で公開されているテクスチャ画像を用いて 6 クラスのテクスチャ画像の分類問題を作成した．各テクスチャ画像について 512×512[pixels] の画像 2 枚を分割して 64×64[pixels] の大きさの画像 128 枚を作成した．それぞれのテクスチャ画像について 10 枚ずつ，計 60 枚の画像を教師画像として用いた．本実験ではすべての画像をグレースケールに変換して使用している．教師画像を図 4.2 に示す．

GIN-IC を用いた画像分類では 1 つの構造で多クラス分類が可能である．本実験では出力ノードを 6 個用意して各ノードが各クラスに対応するものとした．画像分類の際には，出力ノードの出力値が最も高いノードに対応するクラスに画像を分類することとした．

[†] http://vismod.media.mit.edu/vismod/imagery/VisionTexture/vistex.html

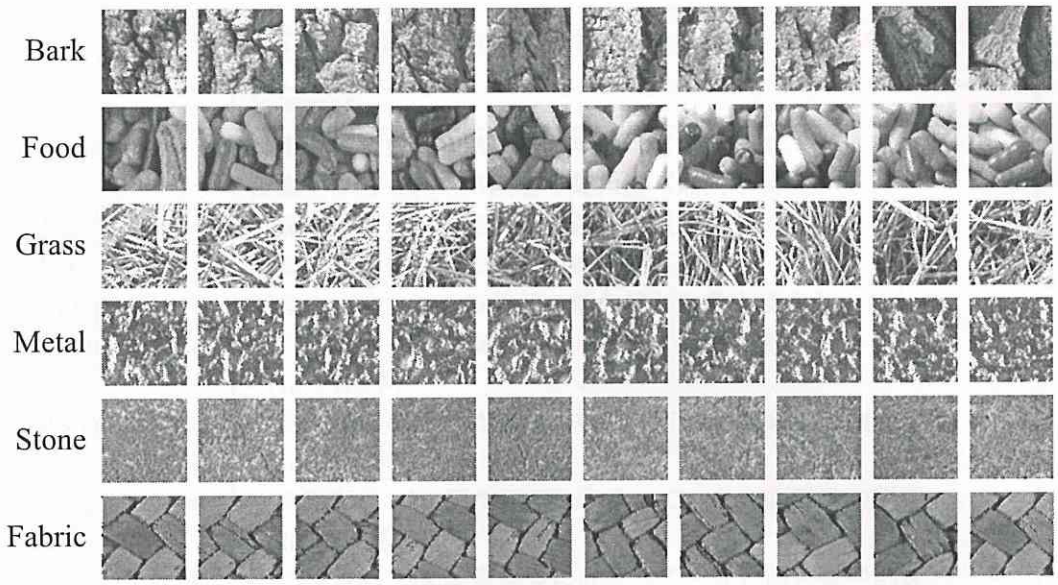


図 4.2: テクスチャ画像の分類実験に用いた教師画像

表 4.1: テクスチャ画像の分類実験で用いた GIN-IC のパラメータ値

世代交代モデル	(1+4)ES
世代数	112500
突然変異率 (P_m)	0.02
画像変換ノード数	100, 0
特徴量抽出ノード数	100
演算ノード数	100
出力ノード数	6

各個体の評価関数には式 4.1 を用いた．この評価関数は，教師画像を正しいクラスに分類できた枚数と使用されているノード（active node）数を考慮したものである．

$$f = N_c + \frac{1}{N_a} \tag{4.1}$$

ここで，N_c は教師画像を正しいクラスに分類できた枚数，N_a は使用されているノード数である．この評価関数では，高い値ほど良い評価値である．また，分類性能が同一であった場合，使用されているノード数が少ないほうが良い評価値となる．

今回の実験で使用した GIN-IC のパラメータ値を表 4.1 に示す．本実験では画像変換ノードを用いる効果を検証するために，画像変換ノード数を 0 とした場合についても実験を行った．画像変換ノード数が 0 である場合，通常の GP などを用いた画像分類と同様のものであると考えることができる．実験は同一のパラメータで 10 回の独立な試行を行った．

本実験では，画像変換ノードを 35 種類，特徴量抽出ノードを 17 種類，演算ノードを 20

表 4.2: GIN-IC による画像分類実験で用いた特徴量抽出ノードの一覧

名称	処理内容
Mean	画像中に含まれる階調値の平均（平均階調値）
Max	画像中に含まれる階調値の最大値（最大階調値）
Min	画像中に含まれる階調値の最小値（最小階調値）
SD	画像中に含まれる階調値の標準偏差
Min to max	画像中に含まれる階調値の最大値から最小値を引いたもの
1 sigma in rate	平均階調値 ± 標準偏差に収まっている画素数の割合（1σ 内確率）
2 sigma in rate	平均階調値 ±2× 標準偏差に収まっている画素数の割合（2σ 内確率）
2 sigma out rate	平均階調値 ±2× 標準偏差に収まっていない画素数の割合（2σ 外確率）
3 sigma out rate	平均階調値 ±3× 標準偏差に収まっていない画素数の割合（3σ 外確率）
First quartile	階調値を昇順に並べたとき下から 1/4 の位置に当たる階調値（第 1 四分位数）
Median	階調値を昇順に並べたとき中央の位置に当たる階調値（中央値）
Third quartile	階調値を昇順に並べたとき下から 3/4 の位置に当たる階調値（第 3 四分位数）
Mode	もっとも出現頻度の高い階調値（最頻値）
Skewness	階調値分布の非対称性を表す尺度（歪度）
Leptokurtion	階調値分布の尖り具合を表す尺度（尖度）
255 pixel rate	画像中に含まれる 255 階調の画素の割合
0 pixel rate	画像中に含まれる 0 階調の画素の割合

* 特徴量抽出ノードは全て 1 つの画像入力に対して 1 つの値を出力する

種類用意した．実験に使用した画像変換ノード（画像処理フィルタ）は付録 A の画像処理フィルター覧から nop，nop1，nop2 を除いた 35 種類である．特徴量抽出ノードとしては平均階調値や階調値の標準偏差を求めるものなどの典型的な統計量を算出するものを，演算ノードとしては算術演算や大小比較などのノードを用いている．本実験に用いた特徴量抽出ノードを表 4.2 に，演算ノードを表 4.3 に示す．いずれのノードについても基礎的かつ重要なノードとして筆者らが選択した．GIN-IC には，これらのノードの組み合わせによって複雑な分類器を構成することが期待される．

表 4.3: GIN-IC による画像分類実験で用いた演算ノードの一覧

名称	入力数	処理内容
Sum	2	入力 1 と入力 2 の和を返す
Difference	2	入力 1 と入力 2 の差を返す
Multiplication	2	入力 1 と入力 2 の積を返す
Division	2	入力 1 を入力 2 で割った数を返す
Greater	2	入力 1 が入力 2 より大きければ 1.0 を， そうでなければ 0.0 を返す
Less	2	入力 1 が入力 2 より小さければ 1.0 を， そうでなければ 0.0 を返す
Equal	2	入力 1 と入力 2 が等しければ 1.0 を， そうでなければ 0.0 を返す
Greater (4)	4	入力 1 が入力 2 より大きければ入力 3 を， そうでなければ入力 4 を返す
Less (4)	4	入力 1 が入力 2 より小さければ入力 3 を， そうでなければ入力 4 を返す
Equal (4)	4	入力 1 と入力 2 が等しければ入力 3 を， そうでなければ入力 4 を返す
Absolute	1	入力 1 の絶対値を返す
Threshold	1	入力 1 が 0.0 より大きければ 1.0 を， そうでなければ 0.0 を返す
Piecewise Linear	1	入力 1 が 0.0 より大きくかつ 1.0 より小さい場合は 入力 1 の値を，0.0 以下の場合は 0.0 を， 1.0 以上の場合は 1.0 を返す
Sigmoid	1	シグモイド関数に入力 1 を代入した値を返す （シグモイド関数：$f(x) = \frac{1}{1+exp(-x)}$）
Reverse	1	入力 1 の正負を反転した値を返す
Const 1	0	定数 1.0 を返す
Const 0	0	定数 0.0 を返す
Const 255	0	定数 255 を返す
Const -1	0	定数-1.0 を返す
Const Pi	0	定数 π を返す

4.3.2 実験の結果と考察

画像変換ノード数を 100 とした場合，GIN-IC は教師画像に対する正答率が 100% の分類器を獲得することができた．しかし，画像変換ノードを 0 とした場合では，教師画像に対する正答率が 100% の分類器を獲得することができなかった．これは，画像変換ノードによって原画像を分類しやすい画像に変換することができたからであると考えられ，画像変換ノードを用いることの有効性が確認できた．

次に，獲得した分類器を未知画像の分類に適用する．未知画像とは分類器の構築に使用していない教師画像に類似の画像である．未知画像の枚数は 708 枚（各クラス 118 枚）である．本実験で用いた未知画像の例を図 4.3 に示す．10 回の試行で獲得された分類器を用いて未知画像の分類を行った正答率の平均を表 4.4 に示す．表の値は該当クラスをそのクラスと正しく分類できた割合である．画像変換ノードを用いることで性能が向上しており，特に Bark と Food の正答率が大幅に向上しているのが確認できる．ただし，Metal と Stone に関しては正答率の減少してしまった．画像変換ノードを用いた場合では，未知画像に対しても平均的に高い正答率を獲得することができ，画像変換ノードを用いる有効性を示すことができた．

図 4.4 に GIN-IC によって構築された構造の一例を示す．この構造では画像変換ノードが使用されているのがわかる．図 4.4 中 A のノードでの出力画像例を図 4.5 に示す．これらの出力画像から，GIN-IC では画像変換ノードで画像を変換することによって，画像の特徴を強調することで分類の性能を高めていると考えることができる．図 4.4 中 A のノードは Food に対応する出力ノードに接続しており，A での出力画像も Food の出力画像が他のクラスと比べて特徴的であるのが確認できる．

図 4.4 の構造を用いてテスト画像を分類した正答率を表 4.5 に示す．この分類器では，9 割を超える高い分類性能を達成しているが，Bark を Metal や Stone に間違える傾向が見受けられる．今回の実験では，特徴量として典型的な統計量だけを用いたが，より問題に適した特徴量を用いることで性能の向上が望めると考えられる．

表 4.4: GIN-IC で獲得した分類器を用いて未知画像を分類した結果（10 回の試行の平均）

	画像処理フィルタ	
	あり	なし
Bark	70.7 %	53.4%
Food	86.1 %	44.1%
Grass	88.1 %	84.8%
Metal	76.7 %	92.5%
Stone	80.4 %	83.4%
Fabric	94.7 %	92.0%
Average	82.8 %	75.0%

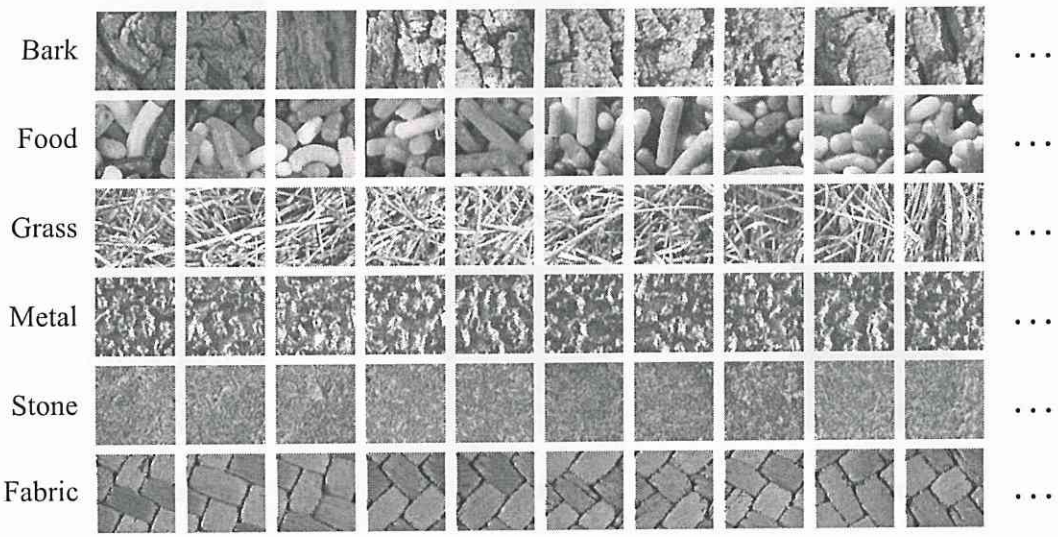


図 4.3: テクスチャ画像の分類実験に用いた未知画像の例

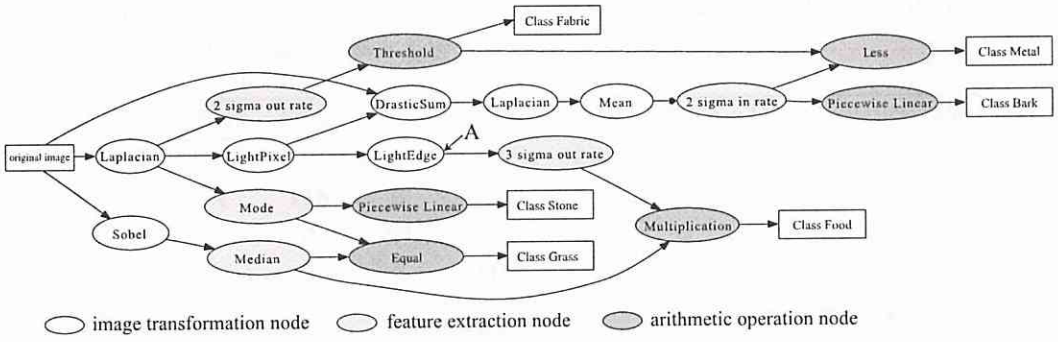


図 4.4: GIN-IC によって構築された分類器の例

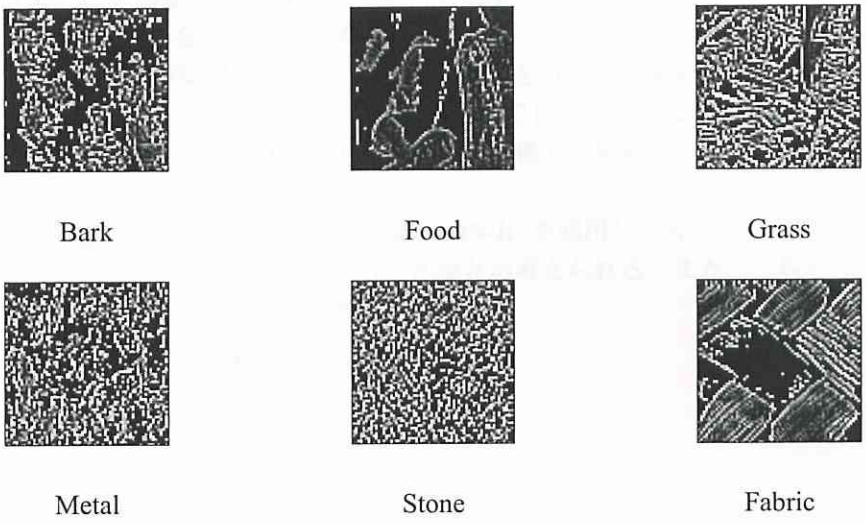


図 4.5: 図 4.4 中の A での出力画像の例

表 4.5: 図 4.4 に示した分類器を用いた場合の未知画像の分類性能（各行は GIN-IC が予測したクラス，各列は正解のクラスを表している）

	Bark	Food	Grass	Metal	Stone	Fabric	Discrimination rate
Bark	**94**	0	0	13	11	0	79.7%
Food	1	**116**	0	0	0	1	98.3%
Grass	8	0	**97**	12	1	0	82.2%
Metal	0	0	0	**110**	8	0	93.2%
Stone	0	0	0	1	**117**	0	99.2%
Fabric	0	0	0	0	13	**105**	90.0%
Average discrimination rate							90.4%

4.4 まとめ

本章では，GINを基にフィードフォワードネットワーク構造状の画像分類器の自動構築法であるGIN-ICを提案した．GIN-ICの最大の特徴は画像変換部を有することであり，画像変換によって原画像を分類し易い画像へ変換することができる．GIN-ICを多クラスのテクスチャ画像の分類問題に適用し，有効性の検証を行い，画像変換ノードを用いない場合と比べて性能が向上することを確認した．GIN-ICによって構築された分類器を確認したところ，画像変換ノードを巧みに用いて画像を変換し，特徴を強調して画像分類を行っていた．

今後の課題としては，他の画像分類問題にGIN-ICを適用して有効性を検証することや，画像の色情報を用いて性能の向上を計ることなどが考えられる．また，一般的な画像分類の手法との比較を行う必要もあると考えている．

第5章　Graph Structured Program Evolution によるプログラムの自動生成

5.1　はじめに

第2章で述べたように，これまでにGPを代表として様々な自動プログラミング手法が提案されている．しかし，ループや再帰構造を必要としたり，複数のデータ型を扱う必要がある複雑なプログラムを自動構築しようとした場合，様々な問題が存在する．一般的なGPではプログラムの表現形式として木構造を用いているが，木構造ではループや再帰構造の表現が困難である．また，複雑なプログラムでは複数のデータ型を扱う必要がある．

これらに対して本章では，より複雑なプログラムを自動生成することを目標として，グラフ構造をプログラムの表現形式とした **GRA**ph structured **P**rogram **E**volution（GRAPE）の提案を行う．GRAPEは先に述べたGPの問題点を克服する手法であり，特徴として次の3点が挙げられる．

1. グラフ構造による表現の拡大
2. 複数のデータ型の取り扱い
3. 進化計算による効率の良い自動構築

特にグラフ構造を用いることによってGPでは自動生成困難なループ構造を含むプログラムの自動生成が可能となる．本章では，提案手法をいくつかのプログラム自動生成問題へ適用し，性能の検証を行う．

5.2　Graph Structured Program Evolution（GRAPE）

5.2.1　GRAPEの構造

本章の提案手法であるGRAPEでは，複雑なプログラムの記述を可能にするためグラフ構造を採用している．GRAPEの構造例を図5.1に示す．GRAPEのプログラムは有向グラフと“データセット”から構成される．“データセット”は有向グラフ中を流れ，各ノードにおいてそのノードに応じた処理が施される．各ノードでは“データセット”に対する処理や“データセット”を用いた分岐が行われる．GPAPEのノードの例を図5.2に示す．“No.1”のノードは整数型のデータdata[0]とdata[1]を足し合わせてdata[0]に代入し，“No.2”のノードへ遷移する．“No.2”のノードは整数型のデータdata[0]とdata[1]を使って次のノードを決定する．

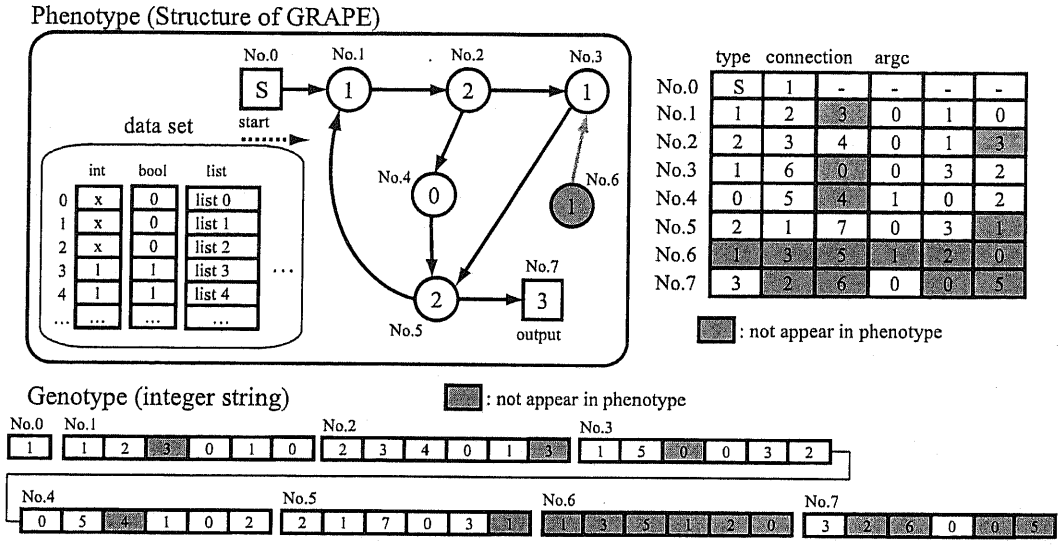


図 5.1: GRAPE の構造例（表現型）とノードの種類，接続，引数を表した遺伝子型の例

GRAPE のプログラムにはいくつかの特別なノードが存在する．図 5.1 の “No.0” のノードはスタートノードと呼ばれるもので，GRAPE のプログラムが実行される際に最初に実行される．“No.7” のノードは出力ノードと呼ばれるもので，出力ノードに達するとデータを出力しプログラムは終了する．図 5.1 の例では，“No.7” のノードは整数型の data[0] を出力する．スタートノードは GRAPE のプログラム中に 1 つしか存在しないが，出力ノードは複数存在することができる．

GRAPE の実行時には，まず “データセット” の値を初期化し入力値などをセットする．その後，スタートノードからプログラムを開始して各ノードを遷移していくことで処理が行われる．GRAPE ではグラフ構造をプログラムの表現形式としているため，分岐やループを含む複雑なプログラムの表現が可能である．さらに，グラフ中を流れる “データセット” に様々なデータ型を用意することで，複数のデータ型を 1 つのプログラムの中で扱うことができる．各ノードではあらかじめ定められたデータ型を使って処理を行う．

GRAPE では表現型のグラフ構造を遺伝子型にマッピングすることで，遺伝子型に対して遺伝操作を行う．GRAPE の遺伝子型は各ノードの種類，接続，引数を定義した一次元の整数列で表現される．遺伝子型から表現型に変換する際，ノードの種類によっては接続先や引数を指定する遺伝子が表現型に発現しない場合もある．GRAPE の総ノード数はあらかじめ指定するため，染色体の遺伝子長は固定長になる．その長さは，$N*(n_c+n_a+1)+1$ である．ここで，N は用意したノードの総数，n_c は接続の最大数，n_a は引数の最大数である．総ノード数は固定であるが，接続の状態によって使用されるノード（active node）と使用されないノード（inactive node）があるため，表現上はノード数は可変となる．図 5.1 の “No.6” のノードは inactive node である．

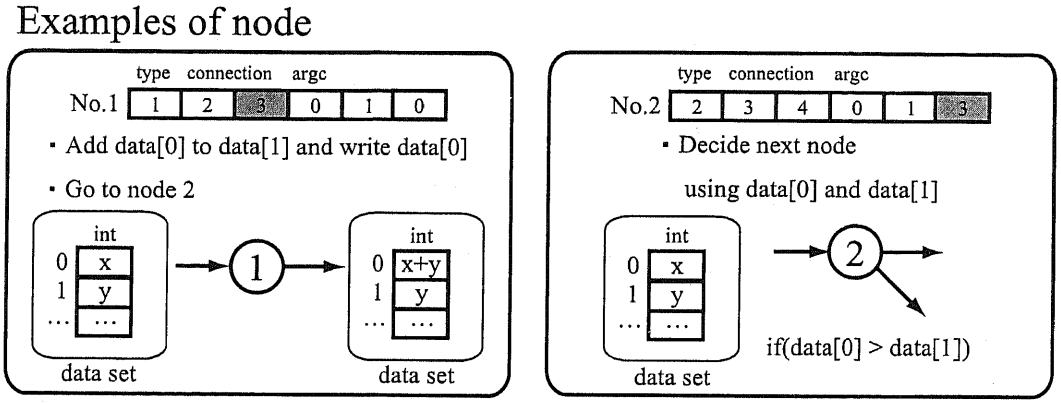


図 5.2: GRAPE のノードの例

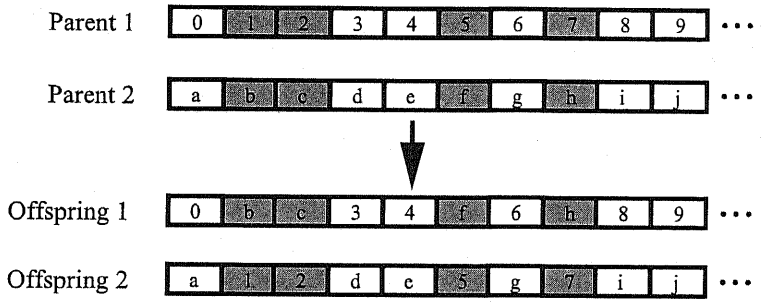


図 5.3: GRAPE における交叉の例（一様交叉）

5.2.2 GRAPE の遺伝操作

GRAPE の各個体の遺伝子型は一次元の整数列で表現されるため，一般的な GA で用いられるような比較的簡単な遺伝操作を適用することができる．5.3 節の実験では遺伝操作に一様交叉と遺伝子に対する突然変異を用いた．一様交叉では確率 P_c によってマスクパターンを生成し交叉点を決定する．GRAPE の交叉の例を図 5.3 に示す．突然変異は突然変異率 P_m によって遺伝子単位で発生するものとする．突然変異が起こるとその遺伝子の値がランダムに変更される．GRAPE の突然変異の例を図 5.4 に示す．

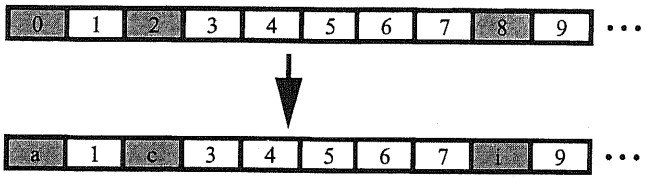


図 5.4: GRAPE における突然変異の例

5.2.3 GRAPE の特徴

ここでは，提案手法である GRAPE の特徴を述べるとともに，従来手法との相違点を考察する．GRAPE の主な特徴として次の 3 つが挙げられる．

1. グラフ構造による表現の拡大
   グラフ構造を採用することで分岐やループを容易に表現することができる．

2. 複数のデータ型の取り扱い
   “データセット” に複数のデータ型を用意することで，複数のデータ型を同時に扱うことが可能である．

3. 効率の良い自動構築
   グラフ構造の表現型から変換された整数列の遺伝子型に対して遺伝操作を行うため，特別な遺伝オペレータを設計する必要が無い．また，固定長の遺伝子型を扱うため，GP に潜在的に存在するブロートのような問題を回避することができると考えられる．

これらの特徴一つ一つについてはこれまでにも多くの研究例が存在するが，これら 3 つの特徴を全て備えている自動プログラミング手法はこれまでに提案されていない．GRAPE ではこれら 3 つの特徴全てを満足するという点がこれまでの手法との大きな違いである．

次に従来手法との相違点を述べる．グラフ構造を表現形式にした自動プログラミング手法は第 2 章で紹介したように，これまでにも提案されている．PADO はグラフ構造を扱う代表的な例であるが，PADO と GRAPE の違いとして，複数のデータ型の取り扱いが可能であることや，遺伝操作を遺伝子型に対して行う点などが挙げられる．PADO を用いて複数のデータ型を取り扱うプログラムの自動生成を行ったという報告はこれまでにされていない．GNP と GRAPE を比較した場合も同様に，複数のデータ型の取り扱いや，遺伝子型への変換などの点で異なる．また，GNP にメモリを導入することでプログラムの自動生成を行う試み [86] も行われているが，本章で扱う複数のデータ型を取り扱う問題や，ループや再帰構造が必要な問題への適用は行われていない．GNP は動的な問題に適用するために提案された手法であるため，終了ノードに対応するものがなく，プログラムの停止回数をあらかじめ決めている点も GRAPE との違いである．GRAPE は構造上の制限はしていないため，任意のグラフ構造が表現可能であり，ある程度表現を制限している PDGP，CGP などとは表現の自由度が異なる．

5.3 自動プログラミングへの適用実験

本節では提案手法であるGRAPEをいくつかの自動プログラミングの問題へ適用し，有効性の検証を行う．本実験では，階乗，フィボナッチ数列，累乗を求めるプログラムの自動生成や，リストの反転やソートを行うプログラムの自動生成を行う．これらのプログラムはループ構造もしくは再帰構造を必要とする問題であり，一般的な木構造を扱うGPでは解くことが困難な問題である．

5.3.1 階乗，フィボナッチ数列，累乗，リストの反転

実験の設定

本実験でのGRAPEのパラメータ値を表5.1に示す．GRAPEの総ノード数については10，30，50と値を変えて実験を行った．生成されたプログラムを実行する際には，ノードの遷移回数に制限を設けることで停止しないプログラムへ対応した．ノードの遷移回数が制限回数に達したプログラムは適応度として0.0が与えられる．本実験ではこの最大ノード遷移回数（execution step limits）を500とした．これは，目的のプログラムを実行するために十分な数であるといえる．

GRAPEにおけるプログラムの進化戦略として，本実験ではSimple GA（SGA），Minimal Generation Gap（MGG）[18]，Random Search（RS）の3つの方法を用いた．SGAはGAの世代交代モデルの中で最も基本的な手法である．本実験で用いたSGAは，トーナメント選択（トーナメントサイズ2）とエリート保存戦略（エリートサイズ5）を併せた世代交代を行う．MGGは多様性が保持される世代交代モデルとして知られており，多くの問題でSGAに比べて性能の高さを示している．本実験ではMGGの子個体数は50とした．RSはGAによる進化が適切に行われているかを検証するための比較対象に用いた．RSでは全ての個体はランダムに生成され，淘汰圧や遺伝操作は用いない．各戦略で世代交代時の個体の生成数が異なるため，条件を揃えるために進化の終了条件は個体を2500000個体評価するまでとした．

本章の実験に用いたGRAPEのノード関数を表5.2にまとめて示す．ノード関数は整数型のデータに対する四則演算やリストの交換，比較などの基本的なものであり，問題ごとに特別な関数を用意することはしない．

表 5.1: GRAPE によるプログラムの自動生成実験の各パラメータ値（階乗，フィボナッチ数列，累乗，リストの反転）

個体の評価回数	2500000
個体数	500
交叉率	0.9
一様交叉の交叉率 P_c	0.1
突然変異率 P_m	0.02
ノード数	10, 30, 50
最大ノード遷移回数	500

表 5.2: プログラムの自動生成実験に用いた GRAPE のノード関数の一覧

Name	# Connections	Arg(s)	Description
+	1	x, y, z	Use integer data type. Add data[x] to data[y] and substitute for data[z].
−	1	x, y, z	Use integer data type. Subtract data[x] from data[y] and substitute for data[z].
*	1	x, y, z	Use integer data type. Multiply data[x] by data[y] and substitute for data[z].
/	1	x, y, z	Use integer data type. Divide data[x] by data[y] and substitute for data[z].
=	2	x, y	Use integer data type. If data[x] is equal to data[y], choose connection 1; else, choose connection 2.
>	2	x, y	Use integer data type. If data[x] is greater than data[y], choose connection 1; else, choose connection 2.
<	2	x, y	Use integer data type. If data[x] is less than data[y], choose connection 1; else, choose connection 2.
SwapList	1	x, y	Use integer type and a list data. Swap list[data[x]] for list[data[y]].
EqualList	2	x, y	Use integer type and a list data. If list[data[x]] equals list[data[y]], choose connection 1; else, choose connection 2.
GreaterList	2	x, y	Use integer type and a list data. If list[data[x]] is greater than list[data[y]], choose connection 1; else, choose connection 2.
LessList	2	x, y	Use integer type and a list data. If list[data[x]] is less than list[data[y]], choose connection 1; else, choose connection 2.
OutputInt	0	x	Output data[x] and terminate the program.
OutputList	0	-	Output a list data and terminate the program.

Factorial（階乗を求めるプログラムの自動生成）

ここでは，階乗（$a!$）を求めるプログラムの自動生成を行う．個体の評価に用いるトレーニングデータとして，0から5の入力値を用いた．トレーニングデータの（入力，出力）のペア（a, b）は，(0, 1)，(1, 1)，(2, 2)，(3, 6)，(4, 24)，(5, 120）である．

本実験では正規化された誤差を評価関数として使用した．評価関数は次の式で表される．

$$fitness = 1.0 - \frac{\sum_{i=1}^{n} \frac{|Correct_i - Out_i|}{|Correct_i| + |Correct_i - Out_i|}}{n} \tag{5.1}$$

ここで，$Correct_i$ はトレーニングデータ i の正しい出力値，Out_i は生成されたプログラムがトレーニングデータ i に対して返した値である．n はトレーニングデータの総数である．この評価関数では適応度は [0.0, 1.0] の範囲で与えられ，大きい数値ほど良い個体であるといえる．さらに，式5.1で求めた適応度が1.0であった場合，評価関数は次の式5.2を使用する．

$$fitness = 1.0 + \frac{1}{S_{exe}} \tag{5.2}$$

ここで，S_{exe} はノードの遷移回数の総数である．この評価関数では，ノードの遷移回数が少ないプログラムほど良い個体としている．以上から各個体の評価関数としては，まず，トレーニングデータの正解値と出力値の誤差が小さい個体ほど良い評価を与え，理想の出力が得られる個体にはノードの遷移回数が少ない個体ほど良い評価とする．つまり，適応度が1.0を超えた個体（プログラム）はトレーニングデータに対しては100%正しい出力を返すプログラムであるといえる．

本実験では整数型のデータを使用し，サイズは10とした．プログラム実行時に，このデータ型の data[0]-[4] に入力値 a，data[5]-[9] に定数1をセットし，初期化する．ノード関数は Table 5.2 に示した { +, −, *, =, >, <, OutputInt } を用いた．

実験として同一のパラメータ条件で100回の試行を行った．図5.5に100回試行中の成功回数（number of successful runs）の推移を示す．今回の実験では，適応度が1.0を超える，つまりトレーニングデータに対して100%正確な出力を返す個体が現れた場合に，その試行を成功としてカウントしている．

次に，各試行において最も高い適応度を獲得した個体をテストデータに適用した．テストデータには入力値として6から12を用いた．テストデータに対する出力値を評価し，全てのテストデータに対して正しい出力値を返すプログラムである場合に，その試行を成功としてカウントする．表5.3にトレーニングデータとテストデータに対する成功回数を示す．

図5.6は本実験でGRAPEが生成した構造の一例である．このプログラムはループを含む構造となっており，任意の入力値に対して正しく階乗を求めるプログラムとなっている．

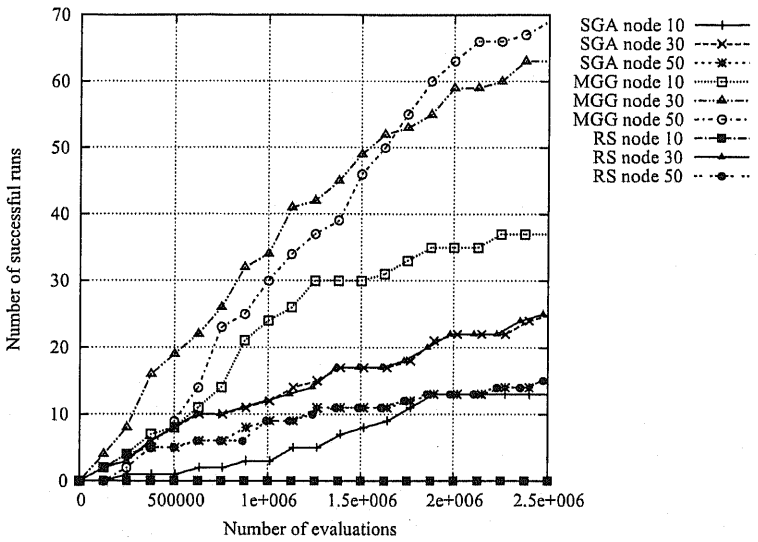


図 5.5: GRAPE によるプログラム自動生成実験の成功回数の推移（階乗）

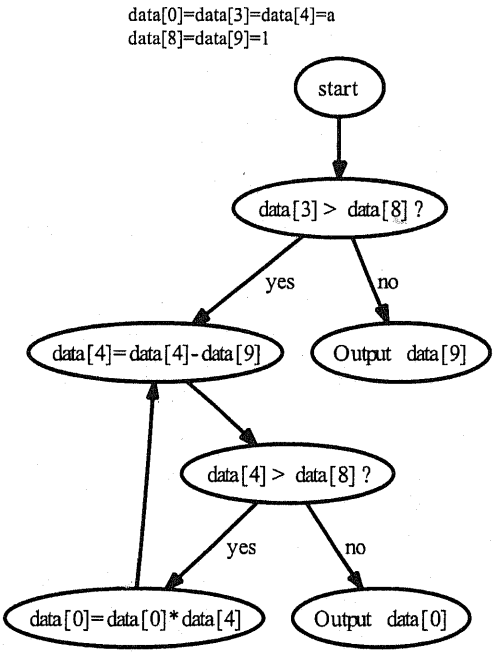


図 5.6: GRAPE によって自動生成されたプログラムの例（階乗）

Fibonacci sequence（フィボナッチ数列を求めるプログラムの自動生成）

ここでは，フィボナッチ数列の a 番目の値 b を求めるプログラムの自動生成を行う．個体の評価に用いるトレーニングデータとして，1 から 12 の入力値を用いた．トレーニングセットの（入力，出力）のペア（a, b）は，(1, 1)，(2, 1)，(3, 2)，(4, 3)，(5, 5)，(6, 8)，(7, 13)，(8, 21)，(9, 34)，(10, 55)，(11, 89)，(12, 144) である．

評価関数には式 5.1 と 5.2 を用いた．本実験では整数型のデータを使用し，サイズは 10 とした．プログラム実行時に，このデータ型の data[0]-[4] に入力値 a，data[5]-[9] に定数 1 をセットし，初期化する．ノード関数は表 5.2 に示した { +, −, *, =, >, <, OutputInt } を用いた．

実験として同一のパラメータ条件で 100 回の試行を行った．図 5.7 に 100 回試行中の成功回数の推移を示す．

次に，各試行において最も高い適応度を獲得した個体をテストデータに適用した．テストデータには入力値として 13 から 30 を用いた．表 5.3 にトレーニングデータとテストデータに対する成功回数を示す．

図 5.8 は本実験で GRAPE が生成した構造の例である．階乗の実験と同様にループを含む構造となっており，このプログラムは任意の入力値に対して正しくフィボナッチ数列を求めるプログラムとなっている．

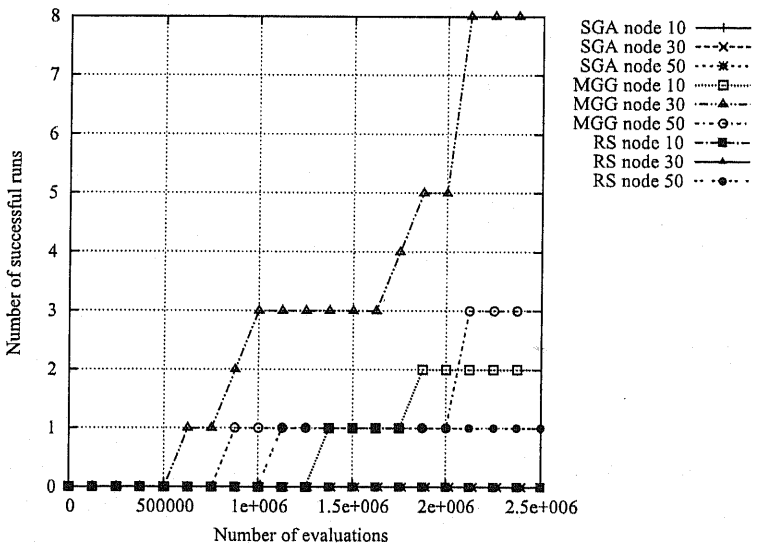


図 5.7: GRAPE によるプログラム自動生成実験の成功回数の推移（フィボナッチ数列）

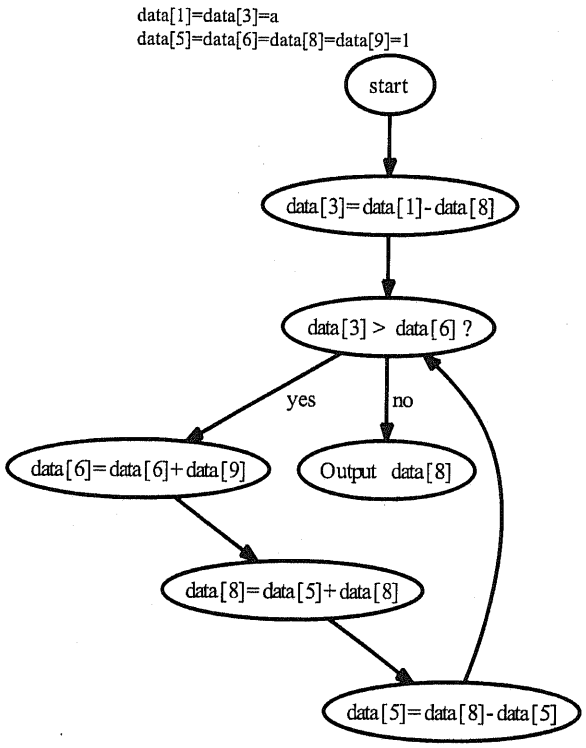


図 5.8: GRAPE によって自動生成されたプログラムの例（フィボナッチ数列）

Exponentiation（累乗を求めるプログラムの自動生成）

ここでは，累乗（a^b）を算出するプログラムの自動生成を行う．この問題は入力値が 2 つあり，a^b を求めることが目的である．個体の評価に用いるトレーニングデータ（a, b, c）として，(2, 0, 1)，(2, 1, 2)，(2, 2, 4)，(3, 3, 27)，(3, 4, 81) (3, 5, 243)，(4, 6, 4096)，(4, 7, 16384)，(4, 8, 65536) を使用した．

評価関数には式 5.1 と 5.2 を用いた．本実験では整数型のデータを使用し，サイズは 9 とした．プログラム実行時に，このデータ型の data[0]-[2] に入力値 a，data[3]-data[5] に入力値 b，data[6]-[8] に定数 1 をセットし，初期化する．ノード関数は表 5.2 に示した { +, −, *, =, >, <, OutputInt } を用いた．

実験として同一のパラメータ条件で 100 回の試行を行った．図 5.9 に 100 回試行中の成功回数の推移を示す．

次に，各試行において最も高い適応度を獲得した個体をテストデータに適用した．テストデータの入力値（a, b）には，(5, 9)，(5, 10)，(5, 11)，(4, 12)，(4, 1)，(4, 14)，(3, 15)，(3, 16)，(3, 17)，(2, 18)，(2, 19)，(2, 20) を用いた．表 5.3 にトレーニングデータセットとテストデータセットに対する成功回数を示す．

図 5.10 は本実験で GRAPE が生成した構造の例である．このプログラムも任意の入力値に対して正しく累乗を求めることができるプログラムとなっている．

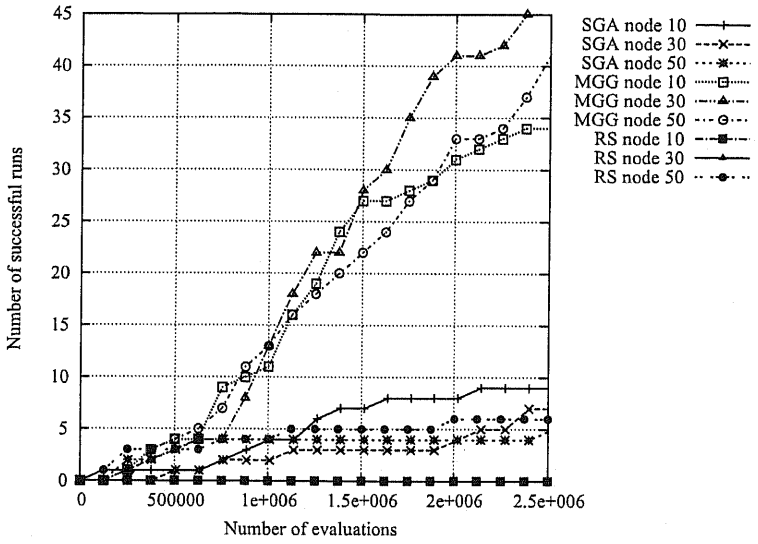


図 5.9: GRAPE によるプログラム自動生成実験の推移（累乗）

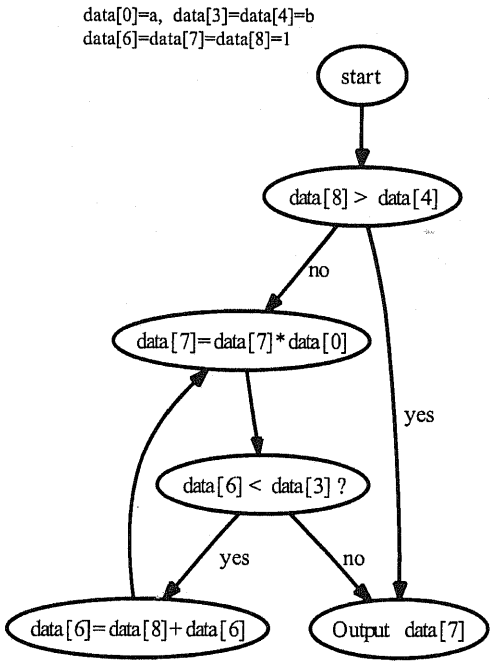


図 5.10: GRAPE によって自動生成されたプログラムの例（累乗）

Reversing a list（リストを反転させるプログラムの自動生成）

ここでは，リストを反転するプログラムの自動生成を行う．この問題は入力としてリストを使用するので，プログラム内で整数型とリストの2つのデータ型を扱う必要がある．正しいプログラムは入力されたリストを反転させたものを返す（例：入力（1 2 3 4），出力（4 3 2 1））．トレーニングデータとして，長さ5から10のリストを用いた．

評価関数は次の式5.3で算出される．

$$fitness = 1.0 - \frac{\sum_{i=1}^{n} \frac{\sum_{j=0}^{l} (1 - \frac{1}{2^{d_j^i}})}{l_i}}{n} \tag{5.3}$$

ここで，d_j^i はトレーニングデータ i の j 番目の要素の位置と，プログラムが返したリストの要素の距離である．l_i はトレーニングデータ i のリストの長さ，n はトレーニングデータの総数である．この評価関数では適応度は [0.0, 1.0] の範囲で与えられ，大きい数値ほど良い個体であるといえる．式5.3で求めた適応度が1.0であった場合，評価関数は先に示した式5.2を使用する．

本実験では入力リストと整数型のデータを使用し，整数型データのサイズは9とした．プログラム実行時に，整数データ型の data[0]-[2] に入力リストの長さ（List Length），data[3]-data[5] に0，data[6]-[8] に定数1をセットし初期化する．ノード関数は表5.2に示した { +, −, *, /, =, >, <, SwapList, OutputList } を用いた．

実験として同一のパラメータ条件で100回の試行を行った．図5.11に100回試行中の成功回数の推移を示す．

次に，各試行において最も高い適応度を獲得した個体をテストデータに適用した．テストデータには，長さ11から15のリストを用いた．表5.3にトレーニングデータとテストデータに対する成功回数を示す．

図5.12は本実験でGRAPEが生成した構造の例である．このプログラムは入力された任意の長さのリストを反転するプログラムとなっている．生成したプログラムはリストと整数型のデータ型を扱っており，GRAPEが複数のデータ型を扱うプログラムを自動生成できることが示されたといえる．

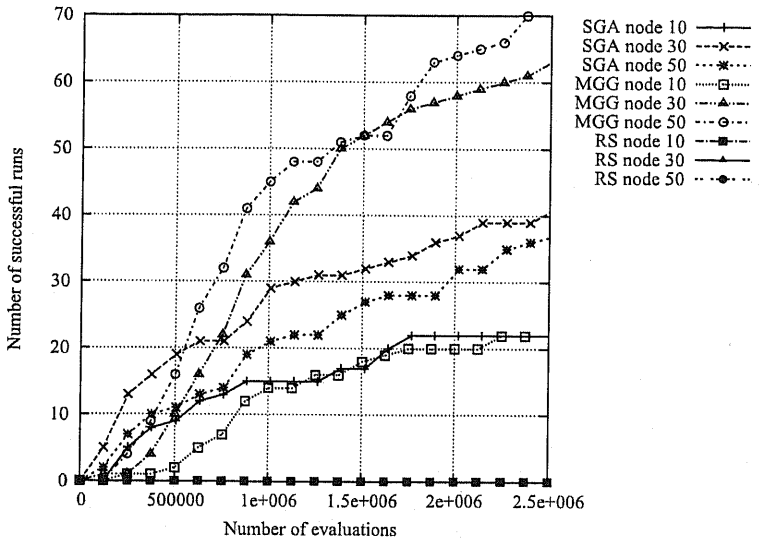


図 5.11: GRAPE によるプログラム自動生成実験の推移（リストの反転）

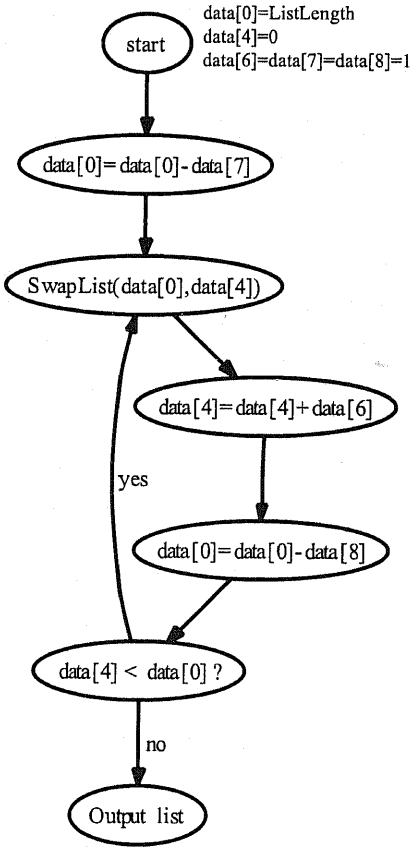


図 5.12: GRAPE によって自動生成されたプログラムの例（リストの反転）

表 5.3: トレーニングデータセットとテストデータセットに対する 100 回試行中の成功回数（階乗，フィボナッチ数列，累乗，リストの反転）

	Factorial		Fibonacci sequence		Exponentiation		Reversing a List	
	Training	Test	Training	Test	Training	Test	Training	Test
SGA (node 10)	13	12	0	0	9	9	22	20
SGA (node 30)	25	23	0	0	7	7	41	30
SGA (node 50)	16	16	0	0	5	5	37	34
MGG (node 10)	37	37	2	2	34	34	22	21
MGG (node 30)	63	57	**8**	**6**	**45**	**44**	63	56
MGG (node 50)	**69**	**59**	3	2	41	40	**71**	**65**
RS (node 10)	0	0	0	0	0	0	0	0
RS (node 30)	1	1	0	0	0	0	0	0
RS (node 50)	15	13	0	0	6	1	0	0

実験結果の考察

本実験では GRAPE の進化の戦略として，SGA，MGG，RS の 3 つの方法を用いた．図 5.5，5.7，5.9，5.11 と表 5.3 の成功回数をみると，いずれの問題においても MGG が他の戦略よりも優れていることがわかる．一方，RS ではどの問題においても解がほとんど得られていない．このことから，進化的な手法が GRAPE に対して有効に働いているということがわかる．また，SGA に比べて MGG のほうが進化の後半でも成功回数の上昇が観測でき，一般的なベンチマークに対する GA で実証されているような多様性の保持機能が GRAPE においてもうまく機能しているといえる．

次にノード数について考察を行う．総ノード数については戦略ごとに 10，30，50 としてそれぞれ実験を行った．いずれの問題においてもノード数を 30，50 とした場合のほうが，ノード数 10 の場合より高い成功回数を示した．図 5.6，5.8，5.10，5.12 に示した GRAPE が獲得した構造をみると，実際に使われているノード数は 10 以下であり，目的のプログラムをノード数 10 でも表現することは可能である．しかし，進化の過程において実際に表現型で使用されていない部分が進化し，交叉，突然変異によって有効な部分へと変化することで，目的の解を得られる可能性がある．このことから，ノード数は十分な数を用意することが有効な探索に繋がるといえる．

それぞれの問題において，各試行で得られた最も適応度の高い個体をテストデータに適用した際の成功回数が表 5.3 に示されている．表 5.3 をみると，トレーニングデータに対して成功している個体のほぼ 9 割程度は，テストデータに対しても正しい出力を返すプログラムとなっていることがわかる．このことから，提案手法で自動的に構築されたプログラムは問題に対する汎用的なプログラムとなっていると考えられる．

図 5.6 の階乗を求めるプログラムを C 言語の形式に変換すると，図 5.13 のように表現することができる．図 5.6 のプログラムでは入力値が代入された data[4] を使って，“data[4] から 1 を引く”，“入力値に data[4] を掛ける”という操作を繰り返すことで階乗を求めている．図 5.6，5.8，5.10，5.12 に示した GRAPE によって自動構築された構造をみると，いずれの構造もループを含み，目的の処理を正しく行うプログラムとなっている．つまり，

```
data[0]=data[3]=data[4]=x;
data[8]=data[9]=1;
if(x > 1) {
   while(1) {
      data[4] = data[4] - 1;
      if(data[4] > 1)
         data[0] = data[0] * data[4];
      else
         return data[0];
   }
}
else {
   return data[9];
}
```

図 5.13: 図 5.6 を C 言語形式に変換したプログラム

本手法によってトレーニングデータの入出力のペアから，図 5.13 に示すような一般的なプログラムを自動的に生成することができたといえる．

今回の実験で用いた問題は簡単な問題ではあるが，実現するためにはループまたは再帰構造が必要となり，通常の木構造を扱う GP では解くことが困難である．これらの問題に対して，GP にいくつかの改良を加えて問題を解いている例もあるが，GP のノード関数にループ用の特別なノードを用意するなど，問題に特化した設計をしなくてはならない．それに対して，本手法ではループ表現は構造的に表現可能であるためループ用に特別なノードを設計する必要が無い．そのため，共通のノードを用いて様々な問題に対して適用することが可能である．

5.3.2 ソートプログラムの自動生成（sorting a list）

ここでは，入力されたリストをソートするプログラムの自動生成を行う．先に行った4つのプログラムの自動生成の実験に比べて，プログラムの規模が大きくなるため汎用的なソートプログラムを獲得することは難しいと考えられる．本実験では世代交代モデルとしてMGGを使用し，MGGの子個体数は50とした．GRAPEの各パラメータは表5.4に示すものを用いた．評価関数にはreversing a listの実験と同様に式5.3と5.2を用いた．個体の評価に用いるトレーニングデータとして，ランダムに発生させた長さ10から20のリスト30例を用いた．

本実験ではreversing a listの実験と同様に入力リストと整数型のデータを使用し，整数型データのサイズは15とした．プログラム実行時に，整数データ型のdata[0]-[4]に入力リストの長さ（List Length），data[5]-data[9]に0，data[10]-[14]に定数1をセットし初期化する．ノード関数は表5.2に示した{ +, −, *, /, =, >, <, SwapList, EqualList, GreaterList, LessList, OutputList }を用いた．ノード関数は数値に対する四則演算とリストの交換，比較などの基本的なものである．

実験として同一のパラメータ条件で100回の試行を行った．図5.14に100回試行中の成功回数の推移を示す．また，図5.15に100回の試行の平均適応度の推移を，図5.16に100回試行中の各世代における最良個体で実際に使用されているノード（active node）の平均数の推移を示す．

次に，各試行において最も高い適応度を獲得した個体をテストデータに適用した．テストデータには，ランダムに発生させた長さ5-10，10-20，20-50，50-100のリスト4種類を用意した．各テストデータの数は500例とした．表5.5にトレーニングデータとテストデータに対する成功回数を示す．

表5.4: GRAPEによるソートプログラムの自動生成実験の各パラメータ値

個体の評価回数	5000000
個体数	500
交叉率	0.9
一様交叉の交叉率 P_c	0.1
突然変異率 P_m	0.02
ノード数	10, 30, 50
最大ノード遷移回数	3000

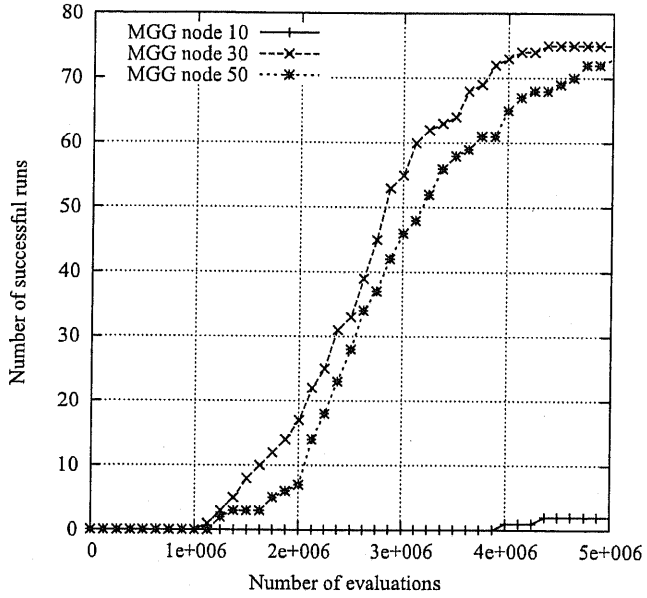


図 5.14: GRAPE によるソートプログラムの自動生成実験における成功回数の推移

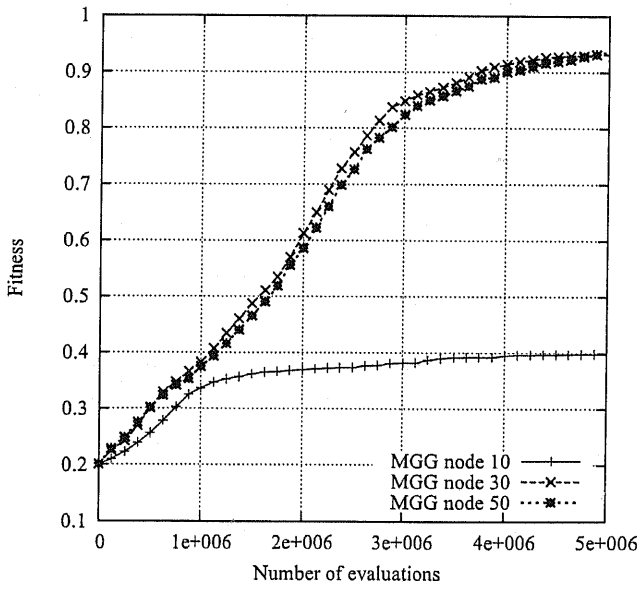


図 5.15: GRAPE によるソートプログラムの自動生成実験における適応度の推移

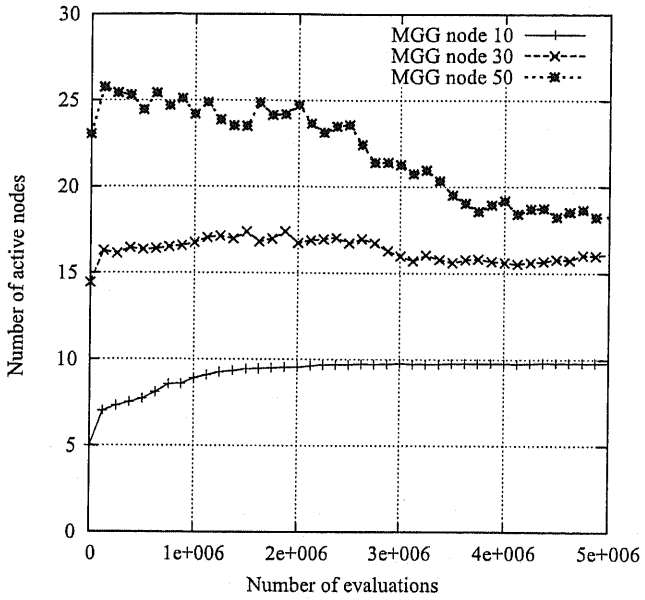


図 5.16: GRAPE によるソートプログラムの自動生成実験における使用ノード数の推移

表 5.5: ソートプログラムの自動生成実験での GRAPE のトレーニングデータセットとテストデータセットに対する 100 回試行中の成功回数

	MGG (node 10)	MGG (node 30)	MGG (node 50)
Training set	2	75	73
Test set (5-10)	1	26	30
Test set (10-20)	1	28	29
Test set (20-50)	1	25	31
Test set (50-100)	1	26	30

図 5.14 からノード数 30 と 50 と設定して実験した場合は，進化の後半にかけて成功回数の上昇が著しい．一方，ノード数が 10 の場合は最終的な成功回数は 2 回と低い結果になっている．これは，ノード数が 10 ではソートという複雑なアルゴリズムを表現することが困難であるためだと考えられる．また，進化の前半では成功回数がほぼ 0 回となっており，トレーニングデータを満足するプログラムが得られていない．このことからも，ソートプログラムの自動生成が難しいことがうかがい知れる．図 5.15 に示した平均適応度の推移をみると進化の前半にも適応度の上昇が確認できる．つまり，トレーニングデータを満足するプログラムの獲得はできていないものの，適応度を上昇させる方向に進化が働き，進化の後半で有効なプログラムを発見することができているといえる．この結果から，進化計算による効率のよい探索が行われていると考えられる．

ノード数を 30，50 とした場合の最終的な成功回数はそれぞれ 75，73 回であった．7 割

以上の試行でトレーニングデータを満足するソートプログラムが獲得されておりGRAPEの性能の高さを示しているといえる．また，表5.5からトレーニングデータに対する成功回数に比べてテストデータに対する成功回数が低下していることがわかる．これは，獲得したプログラムがトレーニングデータに特化したものになってしまったためだと考えられる．今回の実験では評価関数に式5.2を用いて，ノード遷移回数が少ないプログラムほどよいプログラムであるとした．この評価式によって，GRAPEのプログラムはトレーニングデータをより早くソートする方向へと進化していく．しかしこのことが逆に，トレーニングデータに特化したアルゴリズムを生成しやすくし，アルゴリズムの汎用性を低下させる一因となっている可能性があると考えられる．トレーニングデータの選定，評価式の設計やプログラムの構造に関する評価を導入することは今後の課題として考えられる．しかし，テストデータのリストの長さの違いによる成功回数の差を見ると，長さによって大きな成功回数の違いは見られない．このことから，自動生成したプログラムのうちいくつかは入力の長さに頑健なプログラムとなっているといえる．

ソートプログラムを自動生成しようという試みはこれまでにもなされているが，ソート用のノードを設計している手法[27, 28]や再帰を用いる手法[41, 44]がほとんどで，GRAPEのようにループ構造を使用することで解いている例はない．

GRAPEが自動生成したソートプログラムの一例を図5.17に示す．この構造にはループが2つ存在する．図5.17のソートプログラムを，C言語の形式に変換すると，図5.18のように表現することができる．このプログラムは入力された任意の長さのリストをソートするプログラムとなっている．ここで，n番目をリストの最後とする．リスト中で一番大きい値を探し，n番目の要素と交換する．次に，(n-1)番目以下のリストから一番大きい値を探し，(n-1)番目の要素と交換する．これを最後まで繰り返す．このソートアルゴリズムは選択ソート（selection sort）と呼ばれる手法に近い方法である．このプログラムはノード数が10以下で表現されているが，ノード数を10とした場合の成功回数は低い．これはノード数10とした場合，用意されたノードのほぼ全てを効率的に使ってプログラムを表現しなくてはならないのに対して，ノード数30，50の場合は不必要な部分にノードを使用してしまっていても表現ができるためだと考えられる．図5.16のプログラム中で実際に使用されているノード数の推移をみると，ノード数10の場合は全てのノードを使用してしまっているが，ノード数30，50では用意した数の半分程度を使用して進化している．このノード数の推移のグラフから，木構造表現のGPで問題となっているブロートがGRAPEにおいては生じておらず，安定した探索が行えていると考えられる．

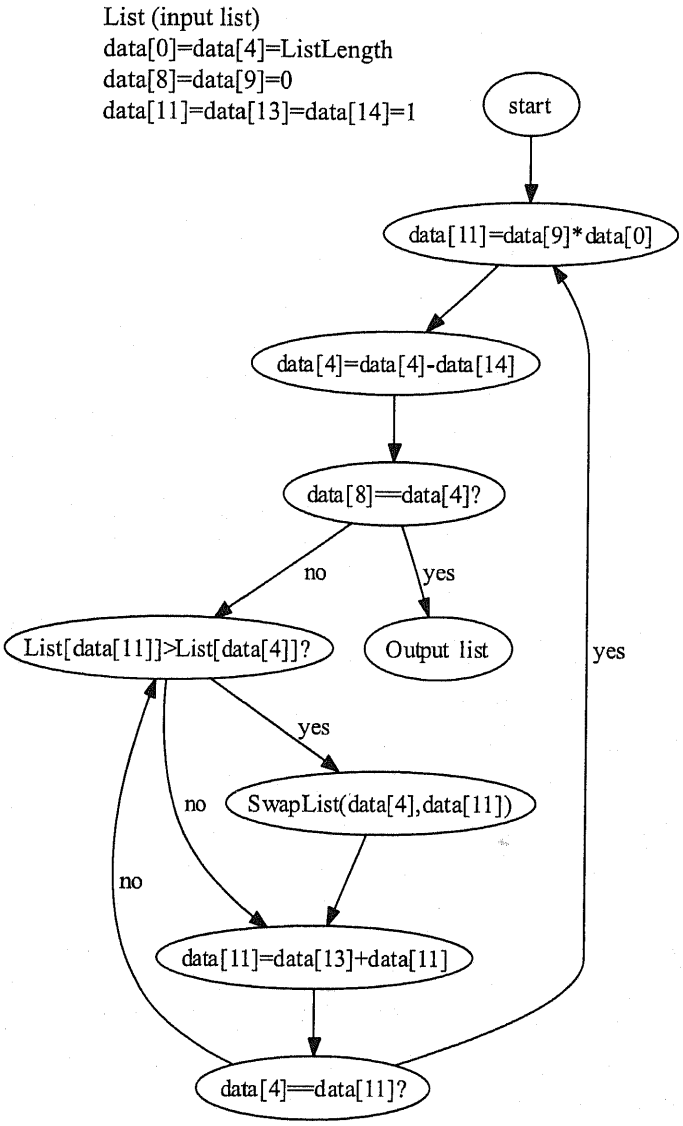


図 5.17: GRAPE によって生成されたソートプログラムの例

```
List=input list;
data[4]=ListLength;
data[11]=1;
while(1) {
  data[11] = 0;
  data[4] = data[4] - 1;
  if(data[4] == 0) {
    return List;
  }
  else {
    do {
      if(List[data[11]] > List[data[4]])
        SwapList(data[4], data[11]);
      data[11] = data[11] + 1;
    } while(data[4] == data[11]);
  }
}
```

図 5.18: 図 5.17 のプログラムを C 言語形式に変換したプログラム

5.3.3 他の手法との比較

先の実験で扱ったようなプログラムの自動生成問題を解いている手法の一つに，Object Oriented Genetic Programming （OOGP）[43, 44]がある．OOGP では再帰構造を導入することでこれらの問題を解くことを可能にしている．表 5.6 は文献 [43, 44]から抜粋した OOGP を用いた各種のプログラム自動生成の結果と，GRAPE を用いた場合の結果である．数値は 100 回の独立な試行のうちトレーニングデータに対して正しい出力を行うプログラムが獲得できた試行回数である．Fibonacci sequence では GRAPE は OOGP に劣るものの，他の問題においては同等以上の性能を示している．各種パラメータや用意している関数などが完全に同一ではないため，対等な比較を行うことはできないが，本章で扱ったプログラム自動生成問題に対して GRAPE の有効性が確認できたと考えられる．

表 5.6: 100 回の試行中の OOGP と GRAPE の成功回数の比較

	OOGP*	GRAPE
Factorial	74	69
Fibonacci sequence	25	8
Exponentiation	6	45
Sorting a list	46	75

*OOGP の値は文献 [43, 44]からの抜粋

5.4 GRAPEにおける交叉方法の比較

5.4.1 概要

5.2.2節で述べたように，GRAPEの各個体の遺伝子型は一次元の整数列で表現されるため，一般的なGAで用いられるような比較的簡単な遺伝操作を適用することができる．前節の実験では，GRAPEの遺伝操作として一様交叉と突然変異を用いた．ここでは，GRAPEを用いた階乗$n!$，組合せの数${}_nC_m$を求めるプログラム，リストのソートを行うプログラムの自動生成を例に，複数の交叉方法を比較し，その解析を行う．

5.4.2 GRAPEにおける交叉方法の比較

交叉方法の比較の実験では，GRAPEにおける交叉方法の違いによる性能を比較するために，次に示す交叉方法を用いた．また，いずれの交叉方法も突然変異と併用することとした．

- 一様交叉（Uniform Crossover; UC）
  確率P_cによって整数列にマスクパターンを生成し交叉点を決定する．実験ではP_cの値として0.1と0.5を用いる．

- 一様ノード交叉（Uniform Node Crossover; UNC）
  確率P_cによってノードごとにマスクパターンを生成し交叉点を決定する．一様交叉との違いは整数列に対する交叉ではなく，ノード単位での交叉を行う点である．実験ではP_cの値として0.1と0.5を用いる．

- 二点ノード交叉（Two Point Node Crossover; TPNC）
  一様乱数によってノード単位で交叉点を2点決定する．

- 交叉を行わない（Mutation Only; MO）
  遺伝操作として交叉を使用せず，突然変異だけを用いる．突然変異は突然変異率P_mによって遺伝子単位で発生するものとする．選択された遺伝子はランダムに変更される．実験ではP_mの値として0.02と0.1を用いる．木構造を扱うGPにおいては，交叉が問題によってはほとんど無意味であるという報告もされている[87]．

5.4.3 GRAPE の交叉方法の比較実験

ここでは GRAPE を階乗 $n!$（factorial），組合せの数 ${}_nC_m$（combination）を求めるプログラム，リストのソート（sorting a list）を行うプログラムの自動生成に適用し，各交叉方法の比較，解析を行う．

実験の設定

実験で使用した GRAPE の各パラメータ値を表 5.7 に示す．生成されたプログラムを実行する際には，ノードの遷移回数に制限を設けることで停止しないプログラムに対応した．遷移回数が制限回数に達したプログラムは適応度として 0.0 が与えられる．本実験ではこの最大ノード遷移回数（execution step limits）を 500（factorial），1000（combination），3000（sorting a list）とした．これは，目的のプログラムを実行するために十分な数であると考えられる．本実験での GRAPE のノード関数は表 5.2 に示したものを用いる．Factorial の実験では { +, −, *, =, >, <, OutputInt } を，combination の実験では { +, −, *, /, =, >, <, OutputInt } を，sorting a list の実験では { +, −, *, /, =, >, <, SwapList, EqualList, GreaterList, LessList, OutputList } を用いた．

個体の評価に用いるトレーニングデータとして，factorial の実験では 0 から 5 の入力値を，combination の実験では，入力値（n, m）の組 30 例を，sorting a list の実験では，ランダムに発生させた長さ 10 から 20 のリスト 30 例を用いた．

個体の評価関数は factorial と combination の実験では，式 5.1 を用いた．Sorting a list の実験ではプログラムの出力がリストとなるため，評価関数として式 5.3 を用いた．さらに，式 5.1，式 5.3 で求めた適応度が 1.0 であった場合，評価関数は次の式 5.4 を使用する．

$$fitness = 1.0 + \frac{1}{N_{active}} \tag{5.4}$$

ここで，N_{active} はプログラム中で使用されているノード数（number of active nodes）である．この評価関数では，active node が少ないプログラムほど良い個体としている．以上か

表 5.7: GRAPE の交叉方法の比較実験に用いた各パラメータ値

世代交代モデル	MGG
世代数	100000
個体数	500
MGG の子個体数	50
一様交叉の交叉率 P_c	0.1, 0.5
突然変異率 P_m	0.02, 0.1
ノード数	50
最大ノード遷移回数	500（factorial） 1000（combination） 3000（sorting a list）

ら各個体の評価としては，まず誤差の小さい個体ほど良い評価を与え，理想の出力が得られる個体にはactive nodeが少ない個体ほど良い評価を与える．つまり，適応度が1.0を超えた個体はトレーニングデータに対しては100%正しい出力を返すプログラムであるといえる．

Factorialの実験では，“データセット”にサイズ10の整数型のデータを用意し，プログラム実行時に，このデータ型のdata[0]-[4]に入力値n，data[5]-[9]に定数1をセットし初期値とする．Combinationの実験では，“データセット”にサイズ15のdouble型のデータを用意し，プログラム実行時に，このデータ型のdata[0]-[4]に入力値n，data[5]-[9]に入力値m，data[10]-data[14]に定数1.0をセットし初期値とする．Sorting a listの実験では，“データセット”に入力リストとサイズ15の整数型のデータを用意し，整数データ型のdata[0]-[4]に入力リストの長さ（List Length），data[5]-data[9]に0，data[10]-[14]に定数1をセットし初期値とする．

それぞれの実験について100回の独立な試行を行い評価する．

実験の結果と考察

100回の試行のうち，トレーニングデータに対して正しい出力が得られるプログラムが生成された試行回数をカウントしたもの（成功回数）を表5.8に示す．いずれの問題においても，“UC (P_c=0.1)”，“UNC (P_c=0.1)”，“MO (P_c=0.02)”は他の交叉方法と比べて良好な結果を示した．Factorialの実験では交叉を行わない“MO (P_c=0.02)”が最も成功率が高かったが，combination，sorting a listの実験では交叉の有効性が観測できる．“UC (P_c=0.5)”，“UNC (P_c=0.5)”，“MO (P_c=0.1)”では他の交叉方法と比べて，表現上で親と大きく異なる子個体が生成されやすいと考えられる．このことから，親から子へと表現上で構造の変更が少ない交叉方法を用いるほうが効率の良い進化が行えると考えられる．

図5.19はsorting a listの実験における各世代ごとの成功回数を示したものである．“UNC (P_c=0.1)”と“MO (P_c=0.02)”が早い世代から正しい出力を得る個体を発見できているのに対し，“UC (P_c=0.1)”では後半にかけての成功率の上昇が著しい．

図5.20はsorting a listの実験における各世代における最良個体のプログラム中で使用されているノード（active node）数の推移である．式5.4により，active nodeの数が少ない個体のほうが良いという評価をしているため，解が発見できた後の世代後半ではactive node数が減少する傾向にある．世代前半を見てみると，“MO (P_c=0.02)”のactive node数が大きいという特徴が分かる．これは低い確率の突然変異だけで次の世代の個体を生成するため，ノード間の接続が大きく変わることが少なく，比較的多くのノードが接続されやすいという特徴をもっていると考えられる．

最後に，combinationの実験で自動生成されたGRAPEのプログラムの例を図5.21に示す．このプログラムは任意の入力n，mに対して，${}_nC_m$を返す一般的なプログラムとなっている．

表 5.8: GRAPE の各交叉方法による成功回数の比較

Crossover method	Factorial	Combination	Sorting a list
UC (P_c=0.1)	86	**3**	**76**
UC (P_c=0.5)	16	0	0
UNC (P_c=0.1)	91	1	75
UNC (P_c=0.5)	34	0	51
TPNC	80	2	72
MO (P_m=0.02)	**96**	2	67
MO (P_m=0.1)	90	0	2

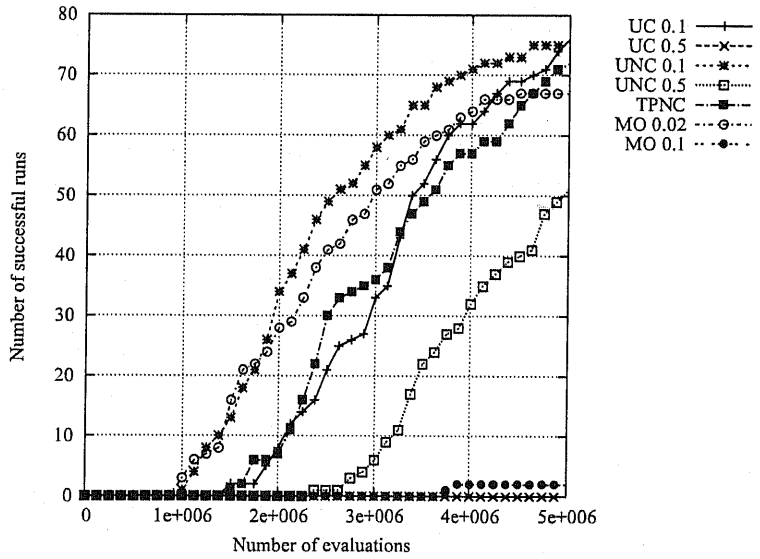


図 5.19: GRAPE における各交叉方法の成功回数の推移の比較（sorting a list）

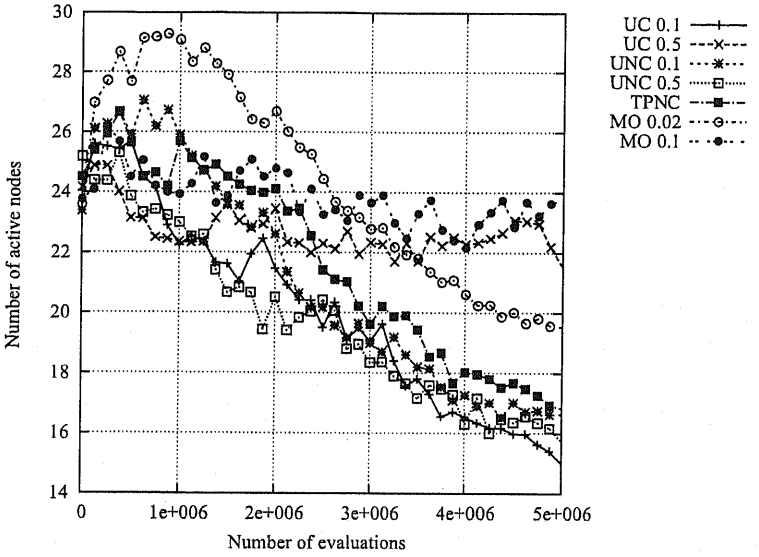


図 5.20: プログラム中で使用されているノード数の推移（sorting a list）

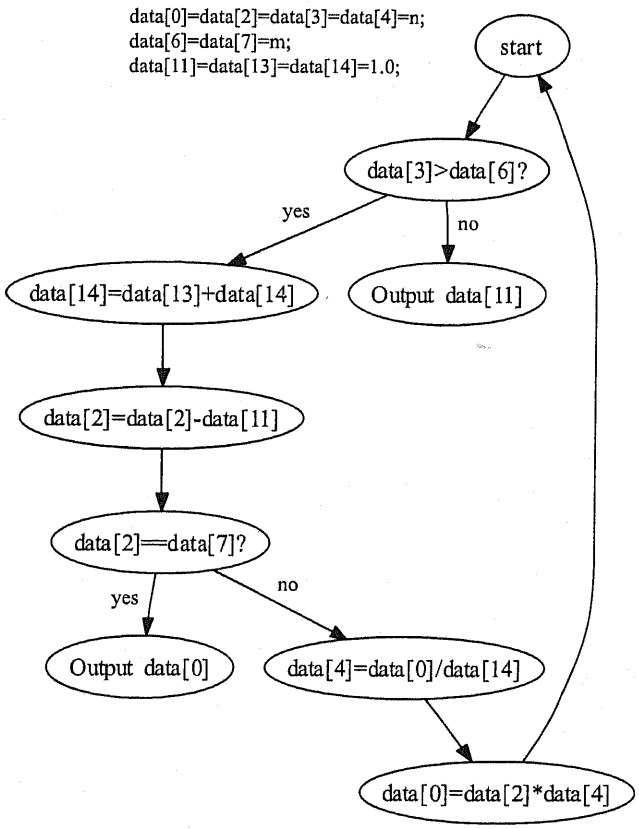


図 5.21: GRAPE が自動生成した組合せの数を求めるプログラムの例

5.5 まとめ

本章では，グラフ構造を用いた自動プログラミングの手法であるGRAPEを提案した．GRAPEはグラフ構造をプログラムの表現形式としているためループの表現が容易にできる．また，“データセット”を用いることで複数のデータ型の取り扱いが可能になる．提案手法のGRAPEをいくつかの自動プログラミングの問題に適用し，有効性の検証を行った．本章で扱った問題は，どれも一般的な木構造をプログラムの表現形式としているGPでは解くことが困難である．実験の結果，GRAPEを用いてループを含む複雑なプログラムの自動生成が可能であることが確認できた．さらに，進化的な探索も有効に働いていることが確認でき，GRAPEによって効率のよい自動生成を行うことができていた．また，GRAPEの交叉方法を変えてプログラムの自動生成実験を行い，交叉方法の影響について調べた．その結果，親から子へと表現上で構造の変更が少ない交叉方法を用いるほうが効率の良い進化が行えることが確認された．

今後の課題として，より効率的なアルゴリズムの自動生成に向けて，トレーニングデータの選定方法や，適応度に計算量やプログラムの構造を考慮したものを導入するなどの検討が必要であると考えられる．さらに，本手法のより複雑な問題への適用が考えられる．具体的には，画像処理や認識の分野や自律エージェントの行動決定などの複雑な問題や複数のデータ型を処理する必要がある問題への適用を考えている．本章では既に知られているアルゴリズムの自動生成を行ったが，GRAPEによって人間でも構築困難なアルゴリズムや考え付かないようなアルゴリズムの自動生成を達成することが期待される．

第6章　Graph Structured Program Evolution による探索アルゴリズムの進化

6.1　はじめに

進化計算に代表されるメタヒューリスティック探索法はこれまでに様々な手法やその改良法が提案され，関数最適化問題や組み合わせ最適化問題などの様々な問題領域に対して有効性を示している．一方，“すべてのブラックボックス探索において，あらゆる目的関数について他を常に上回るような探索アルゴリズムは存在しない”，という No Free Lunch（NFL）[88]定理が示されている．このことから，ある特定の問題領域に対して効率的な探索アルゴリズムを構築することが重要であると考えられる．ある問題が与えられたときにその問題に対する有効な探索法を人手で構築することが一般的であるが，この探索アルゴリズムの構築過程を自動化することは非常に重要な課題であるといえる．

これに対して，GA を用いて GA のパラメータの最適化を行うメタ GA が提案されている [5]．一般的なメタ GA では，GA のパラメータとして，集団サイズや交叉率，突然変異率などの最適化を行う．また，GP の一種である Linear Genetic Programming（LGP）を用いて世代交代の手順を進化させる方法 [89]や，GP を用いて個体の交叉方法を進化させる方法 [90]，Particle Swarm Optimization（PSO）の更新方法を進化させる方法 [91]などが提案され有効性が示されている．また，何らかの学習メカニズムを使用して，低レベルのヒューリスティクスを組合わせて新たなヒューリスティクスを作り出す概念をハイパーヒューリスティクス（Hyper Heuristics）と呼ぶ [92]．ハイパーヒューリスティクスに関する研究の例としては，GA によって bin-packing problem に対するヒューリスティクスを生成する例 [93]や，タブーサーチによってスケジューリング問題のヒューリスティクスを生成する例 [94]などがある．さらに，GP を用いたハイパーヒューリスティクスに関する研究も報告されている．Burke らは GP によって bin-packing problem のヒューリスティクスを生成することに成功している [95, 96]．Fukunaga は SAT 問題に対して人間が設計したヒューリスティクスと同等のヒューリスティクスを自動生成した [97–99]．Kumar らは GP を用いて 2 目的の 0/1 ナップザック問題に対して，ヒューリスティクスの集合が獲得できることを示した [100]．Pillay らは GP によって timetabling problem に対するヒューリスティックスの生成を行っている [101, 102]．

本章では第 5 章で提案した GRAPE を用いて探索空間を探索するエージェントの行動プログラムを自動生成することで，探索アルゴリズムの進化を行うことを目的としている．GRAPE の各ノードでは，エージェントのもつ設計変数への操作を行う．ノードを遷移することによって設計変数が更新され，評価ノードに達した際にその設計変数が評価される．このようにエージェントの設計変数の更新方法を記述したグラフ構造によって，探索アルゴリズムが記述される．本章では，提案手法を関数最適化問題のベンチマーク問題と実問

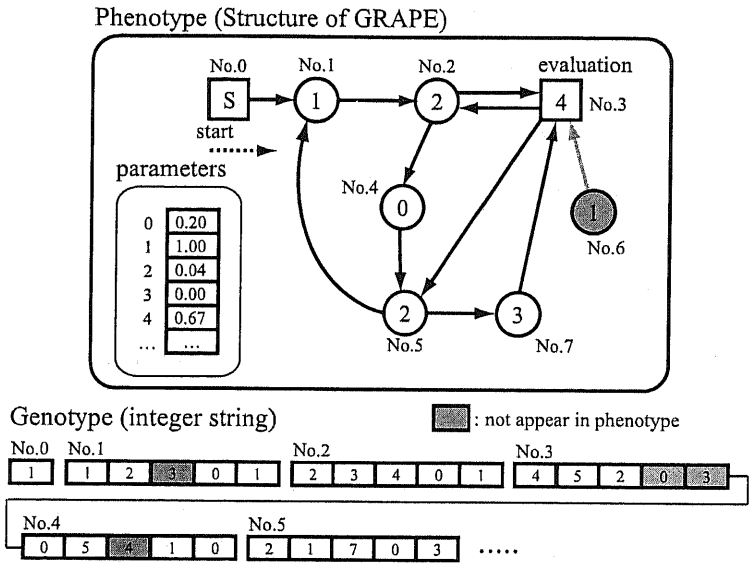


図 6.1: 探索エージェントの行動プログラムを表す GRAPE の構造とその遺伝子型の例

題としてテンプレートマッチングに適用し，探索アルゴリズムの自動生成を行うとともに有効性の検証を行う．

6.2 GRAPE による探索アルゴリズムの進化

本章では GRAPE を用いて探索空間を探索するエージェントの行動プログラムを自動生成することで，探索アルゴリズムの進化を実現する．本モデルでは探索空間中をエージェントが移動することで，次の探索点を決定する．エージェントは GRAPE を用いて表現された行動規則にしたがって探索空間を移動する．GRAPE によって表現された行動規則を最適化することで，探索空間に適応した探索アルゴリズムが獲得できると考えられる．本章での GRAPE のプログラムの構造例を図 6.1 に示す．GRAPE の “データセット” には各問題に対応した設計変数が保存される．GRAPE の各ノードを遷移していくことで自身の設計変数に対する処理が行われ，設計変数が更新される．その後，評価ノード（evaluation node）に達すると自身の設計変数が評価され，次のノードに移る．あらかじめ定められた回数の評価が行われた場合に探索を終了する．このように，エージェントの行動規則を用いて探索が行われていく．

今回の実験で用いたノード関数は，設計変数をランダムに初期化するものや，ある設計変数を代入するもの，2 つの設計変数から一様乱数や正規乱数によって新たな設計変数を生成するものなどである．今回の実験で使用したノード関数を表 6.1 に示す．なお，表中の設計変数 q は，対象エージェントのこれまでの探索過程で最も良い評価の設計変数（personal best），他のエージェントも含めた全体でこれまでに最も良い評価の設計変数（global best），ランダムに選択した他のエージェントの設計変数（random selected agent）の中から決定できるようにした．これらのノード関数は実数値 GA [103, 104]などで用いられるものを

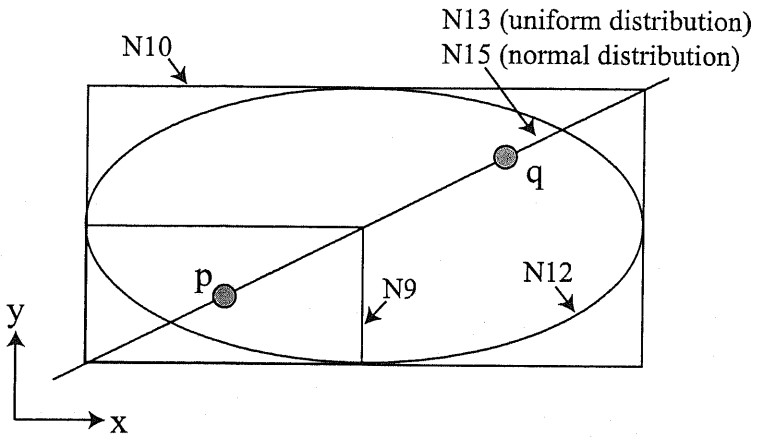


図 6.2: 表 6.1 に示したノード関数を用いた場合に生成される設計変数の範囲分布の例（2 次元の設計変数の場合）

基に用意した．例えば，N16 は実数値交叉の $BLX-\alpha$ である．用意したノード関数を用いて新たに生成される設計変数の分布範囲の例を図 6.2 に示す．この図の例では 2 次元の設計変数空間の場合を示している．また，評価ノード（EVAL）では評価値が現在の global best を更新した場合，personal best を更新した場合，その他の場合で 3 分岐することとしている．

GRAPE で表現された探索アルゴリズムを用いて探索を行った結果を基に GRAPE の各個体は評価される．例えば，複数回の探索の最良評価値の合計などである．今回の実験で用意したノード関数は比較的単純なものであるが，それらを組み合わせることで複雑な探索アルゴリズムの構築が可能となる．このように，特定の探索領域に対して有効な探索アルゴリズムが獲得されることが期待される．

表 6.1: 探索アルゴリズムの獲得実験に用いる GRAPE のノード関数

Id	# Connection	Arg	Definition
For a random selected parameter p_r			
N1	1	–	Initialize the parameter p_r with uniform random number.
N2	1	q	$p_r = q_r$
N3	1	q	$p_r = p_r + u(p_r - \alpha\|p_r - q_r\|, p_r + \alpha\|p_r - q_r\|)$
N4	1	q	$p_r = u(min(p_r, q_r) - \alpha\|p_r - q_r\|, max(p_r, q_r) + \alpha\|p_r - q_r\|)$
N5	1	q	$p_r = N(p_r, (\alpha\|p_r - q_r\|)^2)$
N6	1	q	$p_r = N(\frac{(p_r+q_r)}{2}, (\alpha\|p_r - q_r\|)^2)$
For the all parameters p_i $(i = 1, 2, 3, ...)$			
N7	1	–	Initialize the all parameters p_i with random number.
N8	1	q	$p_i = q_i$
N9	1	q	$p_i = p_i + u(p_i - \alpha\|p_i - q_i\|, p_i + \alpha\|p_i - q_i\|)$
N10	1	q	$p_i = u(min(p_i, q_i) - \alpha\|p_i - q_i\|, max(p_i, q_i) + \alpha\|p_i - q_i\|)$
N11	1	q	$p_i = N(p_i, (\alpha\|p_i - q_i\|)^2)$
N12	1	q	$p_i = N(\frac{(p_i+q_i)}{2}, (\alpha\|p_i - q_i\|)^2)$
Other types			
N13	1	q	$p_i = (p_i - \alpha de_i) + R \cdot ((q_i + \alpha de_i) - (p_i - \alpha de_i))$ d: the distance between p_i and q_i, $e_i = \frac{(q_i - p_i)}{\|q_i - p_i\|}$, $R = u(0.0, 1.0)$
N14	1	q	$p_i = p_i + ze_i$ $z = N(0, (\alpha d)^2)$, d: the distance between p_i and q_i, $e_i = \frac{(q_i - p_i)}{\|q_i - p_i\|}$
N15	1	q	$p_i = \frac{(p_i+q_i)}{2} + ze_i$ $z = N(0, (\alpha d)^2)$, d: the distance between p_i and q_i, $e_i = \frac{(q_i - p_i)}{\|q_i - p_i\|}$
N16	2	–	Decide the next node with rondom number. If $u(0.0, 1.0) < 0.5$, then connection 1 is chosen. Else connection 2 is chosen.
Evaluation node			
EVAL	3	–	Evaluate the current parameters p. And if global best is updated, then connection 1 is chosen. Else if local best is updated, then connection 2 is chosen. Else connection 3 is chosen.

p_i: ith parameter of agent p.
q_i: ith parameter of partner agent q.
(q is best global parameters ($q0$) or best local parameters ($q1$)
or parameters of other randomly selected agents ($q2$).)
$u(x, y)$: uniformly distributed random number among $[x, y]$.
$N(\mu, \sigma^2)$: normally distributed random number.
Parameter $\alpha = 0.5$.

6.3 探索アルゴリズムの自動獲得実験

ここでは関数最適化問題のベンチマーク関数と，実問題としてテンプレートマッチングを例に探索アルゴリズムの獲得実験を行う．その後，獲得した探索アルゴリズムの有効性を検証するために，進化過程に用いていない探索空間に獲得した探索アルゴリズムを適用する．

6.3.1 探索アルゴリズムの獲得実験の設定

今回の実験で使用した各パラメータ値を表 6.2 に示す．ノードの遷移回数に制限を設け，最大遷移回数（execution step limits）内に評価ノードに到達しない個体は最も悪い適応度が与えられることとした．今回の実験では最大遷移回数を 50 とした．GRAPE の遺伝操作としては一様交叉と文字列に対する突然変異を用い，世代交代モデルには MGG [18] を使用した．

表 6.2: GRAPE による探索アルゴリズムの進化の実験に用いた各パラメータ値

Parameters for GRAPE	
最大世代数	10000
個体数	100
MGG の子個体数	20
交叉率	0.6
一様交叉の交叉率 (P_c)	0.1
突然変異率 (P_m)	0.02
ノード数	50
最大ノード遷移回数	50
Parameters for search simulation	
エージェント数	1, 10, 50
関数評価回数	10000（ベンチマーク関数） 5000（テンプレートマッチング）
探索の試行回数 (N_t)	5

6.3.2 ベンチマーク関数

今回の実験では探索アルゴリズムの自動生成を行う際の関数に Rastrigin-d 関数を用いた．Rastrigin-d 関数は次の式（6.1）で表される多峰性の関数である．図 6.3 に 2 次元の場合の Rastrigin-d 関数の景観を示す．

$$f(x_1, ..., x_n) = 10n + \sum_{i=1}^{n} \{(x_i - d)^2 - 10cos(2\pi(x_i - d))\} \tag{6.1}$$

$$(-5.12 \leq x_i \leq 5.12)$$

この関数は，$(d, ..., d)$ で最小値 0 をとる．n は次元数であり，本実験では $n = 20$ とした．また，d については $d = 1.0$ を用いた．

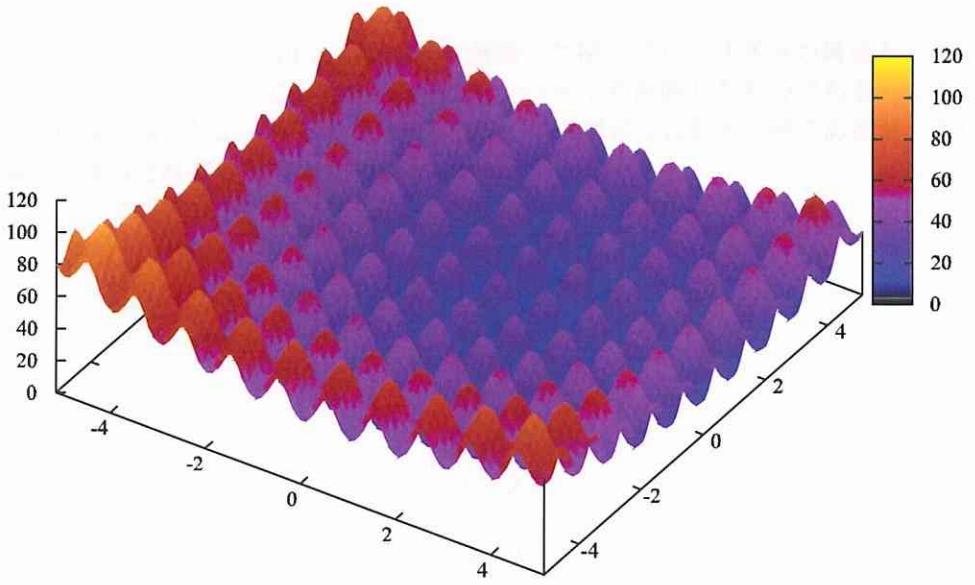


図 6.3: 2 次元の場合の Rastrigin-d 関数の景観

設計変数は定義域内でランダムに初期化され，GRAPE によって表現された探索アルゴリズムによって探索を決められた評価回数行う．エージェント数については 1, 10, 50 と変えて検証を行った．エージェント数が複数の場合はすべてのエージェントが同一の GRAPE で表現された探索アルゴリズムを参照する homogeneous モデルを用いた．また，GRAPE の各個体の評価には次の式を用いる．

$$fitness = -\sum_{i=1}^{N_{trial}} f_i \tag{6.2}$$

ここで，f_i は i 番目の探索で得られた最も良い評価値であり，N_{trial} は探索の試行回数（number of trials）である．この評価関数は値が大きいほうが良い評価である．以上の設定によって探索アルゴリズムの進化を行い，Rastrigin-d 関数に対して有効な探索アルゴリズムを獲得した．

図 6.4 にエージェント数 10 とした場合に構築した探索アルゴリズムの構造を示す．なお，図 6.4 の構造は削除しても処理に影響しないノードを削除したものである．この処理では，評価値の違いによって 2 つの分岐を行う．この構造は比較的単純であるが，より複雑な構造も獲得されていることを確認している．

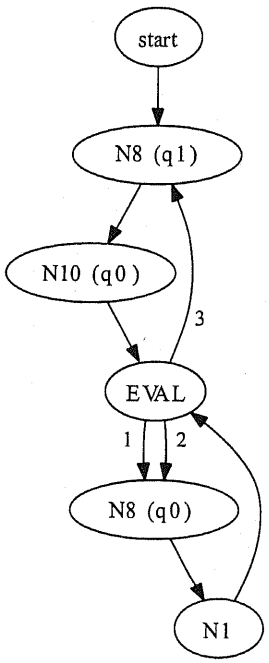


図 6.4: GRAPE によって獲得されたベンチマーク関数に対する探索アルゴリズムの例（エージェント数 10 とした場合）

次に獲得した探索アルゴリズムを様々なベンチマーク関数に適用し，検証を行う．ベンチマーク関数には次の式 6.3〜6.8 に示す 6 種類の関数を使用した．

Sphere 関数は次式で表される単峰性の関数であり，$(0, ..., 0)$ で最小値 0 をとる．

$$f(x_1, ..., x_n) = \sum_{i=1}^{n} x_i^2 \quad (-100 \leq x_i \leq 100) \tag{6.3}$$

Ridge 関数は次式で表される単峰性の関数であり，$(0, ..., 0)$ で最小値 0 をとる．

$$f(x_1, ..., x_n) = \sum_{i=1}^{n} (\sum_{j=1}^{i} x_j)^2 \quad (-100 \leq x_i \leq 100) \tag{6.4}$$

Rosenbrock 関数は次式で表される単峰性の関数であり，$(1, ..., 1)$ で最小値 0 をとる．

$$f(x_1, ..., x_n) = \sum_{i=1}^{n-1} \{100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\} \quad (-30 \leq x_i \leq 30) \tag{6.5}$$

Rastrigin 関数は次式で表される多峰性の関数であり，$(0, ..., 0)$ で最小値 0 をとる．

$$f(x_1, ..., x_n) = 10n + \sum_{i=1}^{n} \{x_i^2 - 10cos(2\pi x_i)\} \quad (-5.12 \leq x_i \leq 5.12) \tag{6.6}$$

Ackley 関数は次式で表される多峰性の関数であり，$(0, ..., 0)$ で最小値 0 をとる．

$$f(x_1, ..., x_n) = 20 - 20exp(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}) + e - exp(\frac{1}{n}\sum_{i=1}^{n} cos(2\pi x_i)) \quad (-32 \leq x_i \leq 32) \tag{6.7}$$

Schwefel 関数は次式で表される多峰性の関数であり，$(420.968750, ..., 420.968750)$ で最小値 $-418.98288727 \times n$ をとる．

$$f = \sum_{i=1}^{n} -x_i sin(\sqrt{|x_i|}) \quad (-500 \leq x_i \leq 500) \tag{6.8}$$

実験条件は次の通りである．

- ベンチマーク関数の次元数 n：$10, 20, 30$
- 試行回数：50
- 関数評価回数：5.0×10^5

関数評価回数は十分な探索を行うために，GRAPE を用いて探索アルゴリズムを構築する際よりも大きくした．

また，一般的な探索手法と性能を比較するために PSO [15] と比較を行う．PSO は動物の社会的振る舞いを模倣した多点探索手法であり，解候補である個体は現在の位置情報で

ある実数値を保持し，それを更新式に従って移動させることで最適解を探索する．各個体 i の現在の位置ベクトルを x_i，速度ベクトルを v_i とする．探索には各個体の自分自身の最良解の位置ベクトル $\hat{x}$ と現在までの全個体中における最良解の位置ベクトル $\hat{x}_g$ を用いる．次にその更新式を示す．

$$v_i = \omega v_i + c_1 r_1(\hat{x} - x_i) + c_2 r_2(\hat{x}_g - x_i) \tag{6.9}$$

$$x_i = x_i + v_i \tag{6.10}$$

ω は速度に対する重み係数で正の値をとる．c_1 と c_2 は各項に対する重み係数である．r_1 と r_2 は $[0.0, 1.0]$ の一様乱数を表している．本実験では，文献 [105] において良い性能が得られるとされた，$\omega = 0.72$，$c_1, c_2 = 1.49$ とした．なお，PSO の個体数は 50 個体として実験を行っている．

表 6.3 に探索を行った結果を示す．表中の値は 50 回の試行で得られた評価値の平均である．また，括弧内の数字は標準偏差である．GRAPE によって獲得された探索アルゴリズムは，PSO と比較しても構築時に使用した Rastrigin 関数などを中心に，多峰性の関数に対しても良好な結果を示しているのがわかる．探索アルゴリズムの構築時には 20 次元の Rastrigin-d 関数だけを用いて GRAPE の最適化を行ったが，得られた探索アルゴリズムが他の関数に対しても良好な結果を得ていることから，頑健性のある探索アルゴリズムの獲得ができたと考えられる．

表 6.3: ベンチマーク関数に対する GRAPE によって獲得された探索アルゴリズムと PSO の探索性能（50 回の試行の平均値，括弧内の数字は標準偏差）

Function	n	# agents = 1	# agents = 10	# agents = 50	PSO
Sphere	10	2.36×10^{-32} (6.58×10^{-32})	1.72×10^{-116} (8.25×10^{-116})	4.10×10^{-119} (1.38×10^{-118})	**0.0** (**0.0**)
	20	7.05×10^{-28} (1.51×10^{-27})	2.54×10^{-81} (7.96×10^{-81})	2.41×10^{-84} (7.24×10^{-84})	$\mathbf{4.94 \times 10^{-324}}$ (**0.0**)
	30	3.91×10^{-25} (7.38×10^{-25})	2.92×10^{-65} (1.29×10^{-64})	1.77×10^{-67} (9.35×10^{-67})	$\mathbf{6.32 \times 10^{-179}}$ (**0.0**)
Ridge	10	2.10×10^{-30} (1.06×10^{-29})	3.25×10^{-114} (2.17×10^{-113})	8.62×10^{-117} (3.84×10^{-116})	$\mathbf{1.22 \times 10^{-276}}$ (**0.0**)
	20	4.00×10^{-25} (2.04×10^{-24})	3.27×10^{-79} (1.11×10^{-78})	$\mathbf{1.28 \times 10^{-82}}$ ($\mathbf{3.27 \times 10^{-82}}$)	5.33×10^{2} (1.63×10^{3})
	30	1.03×10^{-22} (1.66×10^{-22})	5.64×10^{-62} (3.75×10^{-61})	$\mathbf{2.53 \times 10^{-65}}$ ($\mathbf{9.50 \times 10^{-65}}$)	4.07×10^{3} (5.05×10^{3})
Rosenbrock	10	8.22 (7.48)	**4.36** (**5.95**)	4.74 (6.30)	3.74×10^{3} (1.78×10^{4})
	20	9.75 (8.26)	**4.90** (**6.04**)	7.14 (7.88)	9.07×10^{3} (2.73×10^{4})
	30	1.20×10^{1} (1.49×10^{1})	8.65 (**7.92**)	**7.49** (1.15×10^{1})	1.11×10^{4} (2.94×10^{4})
Rastrigin	10	$\mathbf{7.39 \times 10^{-15}}$ ($\mathbf{1.07 \times 10^{-14}}$)	4.52×10^{-14} (6.14×10^{-14})	1.11×10^{-14} (1.34×10^{-14})	6.27 (3.35)
	20	$\mathbf{4.04 \times 10^{-14}}$ ($\mathbf{4.14 \times 10^{-14}}$)	3.16×10^{-13} (2.76×10^{-13})	7.22×10^{-14} (5.25×10^{-14})	3.79×10^{1} (1.18×10^{1})
	30	$\mathbf{1.51 \times 10^{-13}}$ ($\mathbf{9.69 \times 10^{-14}}$)	2.80×10^{-12} (9.27×10^{-12})	2.58×10^{-13} (1.37×10^{-13})	9.38×10^{1} (2.91×10^{1})
Ackley	10	1.68×10^{-14} ($\mathbf{5.36 \times 10^{-15}}$)	3.42×10^{-14} (2.76×10^{-14})	1.44×10^{-14} (6.27×10^{-15})	$\mathbf{4.00 \times 10^{-15}}$ (3.61×10^{-2})
	20	4.80×10^{-14} (1.58×10^{-14})	1.57×10^{-13} (2.88×10^{-13})	$\mathbf{3.42 \times 10^{-14}}$ ($\mathbf{1.11 \times 10^{-14}}$)	3.29×10^{-1} (2.23×10^{-2})
	30	2.29×10^{-13} (1.11×10^{-13})	4.23×10^{-13} (1.10×10^{-12})	$\mathbf{5.69 \times 10^{-14}}$ ($\mathbf{1.20 \times 10^{-14}}$)	1.79 (8.86)
Schwefel	10	$\mathbf{-4.19 \times 10^{3}}$ ($\mathbf{9.09 \times 10^{-13}}$)	$\mathbf{-4.19 \times 10^{3}}$ (1.66×10^{1})	$\mathbf{-4.19 \times 10^{3}}$ ($\mathbf{9.09 \times 10^{-13}}$)	-3.52×10^{3} (2.40×10^{2})
	20	$\mathbf{-8.38 \times 10^{3}}$ ($\mathbf{1.82 \times 10^{-12}}$)	$\mathbf{-8.38 \times 10^{3}}$ ($\mathbf{1.82 \times 10^{-12}}$)	$\mathbf{-8.38 \times 10^{3}}$ ($\mathbf{1.82 \times 10^{-12}}$)	-6.29×10^{3} (4.76×10^{2})
	30	$\mathbf{-1.26 \times 10^{4}}$ ($\mathbf{7.28 \times 10^{-12}}$)	$\mathbf{-1.26 \times 10^{4}}$ ($\mathbf{7.28 \times 10^{-12}}$)	$\mathbf{-1.26 \times 10^{4}}$ ($\mathbf{7.28 \times 10^{-12}}$)	-8.90×10^{3} (5.73×10^{2})

6.3.3 テンプレートマッチング

テンプレートマッチングとは対象画像中でテンプレート画像と類似した領域を見つけ出すことである．テンプレートマッチングは物体検出などに有効であり，画像処理において基礎的かつ重要な処理であるといえる．本実験ではテンプレート画像の 4 つのパラメータ（テンプレート中心の位置 (x,y)，回転角度 $rate$，拡大倍率 $angle$）を設計変数とし探索を行う．本実験で用いたテンプレートマッチングの評価関数は次の式（6.11）で与えられるテンプレート画像との誤差を正規化したものである．

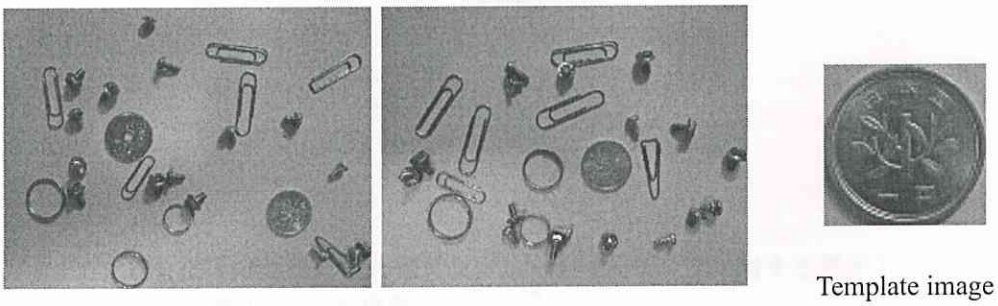
$$
\begin{aligned}
f(x,y,rate,angle) &= \frac{1}{W \cdot H \cdot V_{max}} \sum_{i=1}^{W} \sum_{j=1}^{H} |f_{i'j'} - t_{ij}| \\
i' &= rate * ((i - \frac{W}{2}) * cos(angle) + (j - \frac{H}{2}) * sin(angle)) + x \\
j' &= rate * ((i - \frac{W}{2}) * sin(angle) + (j - \frac{H}{2}) * cos(angle)) + y
\end{aligned}
\tag{6.11}
$$

ここで，t_{ij} はテンプレート画像の画素値，f_{ij} は対象画像の画素値であり，W, H はそれぞれテンプレート画像の i, j 方向の画素数，V_{max} は最大階調値である．この評価関数は値が小さいほうが良い評価である．

本実験で用いたテンプレート画像と対象画像を図 6.5 に示す．画像はグレースケール画像であり，対象画像のサイズは 640×480[pixels]，テンプレート画像のサイズは 64×64[pixels] である．各設計変数の定義域は次の通りである．

- テンプレート中心の位置：$0 \leq x \leq X$，$0 \leq y \leq Y$
  （X, Y は対象画像の x, y 方向の画素数）
- 拡大倍率：$1.0 \leq rate \leq 2.0$
- 回転角度：$-180° \leq angle \leq 180°$

実験のパラメータは表 6.2 と同様のものを用い，GRAPE の各個体の評価も式（6.2）を用いる．図 6.5 の対象画像 1 とテンプレート画像を GRAPE による探索アルゴリズムの構築に使用する．以上の設定によってテンプレートマッチングのための探索アルゴリズムの進化を行った．図 6.6 はエージェント数を 10 とした際に獲得した探索アルゴリズムの構造である．



Target image 1　　Target image 2　　Template image

図 6.5: テンプレートマッチングの実験に用いた対象画像とテンプレート画像

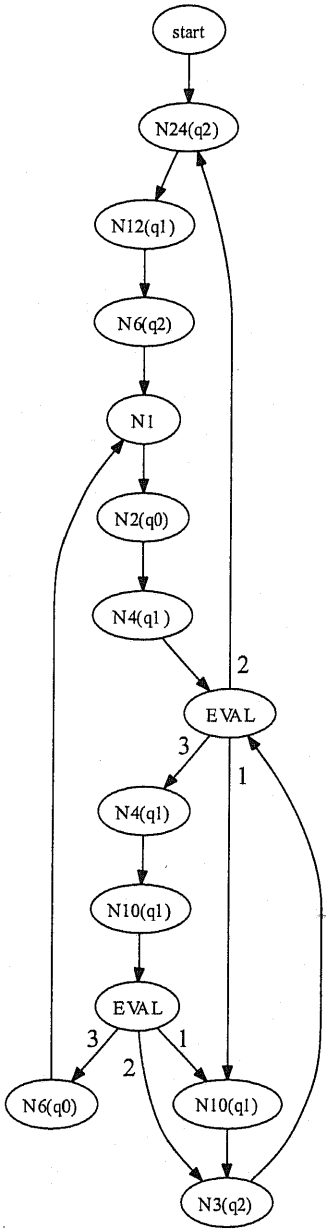


図 6.6: GRAPE によって獲得されたテンプレートマッチングに対する探索アルゴリズムの例（エージェント数 10 とした場合）

表 6.4: テンプレートマッチングに対する GRAPE によって獲得された探索アルゴリズムと PSO の探索性能（50 回の試行の平均値）

	# agents=1	# agents=10	# agents=50	PSO
Target image 1				
Average	0.0890	**0.0889**	0.0890	0.105
Stddev	0.00501	0.00578	**0.00424**	0.0137
Best	0.0846	**0.0845**	**0.0845**	0.0858
Worst	0.107	0.107	**0.103**	0.134
Target image 2				
Average	0.0735	0.0726	**0.0714**	0.0793
Stddev	0.00684	0.00328	**0.00281**	0.0113
Best	**0.0698**	**0.0698**	**0.0698**	0.070
Worst	0.116	**0.0776**	0.0790	0.132

GRAPE によって構築された探索アルゴリズムと PSO の探索性能を表 6.4 に示す．最大評価回数は 10000 とし，図 6.5 に示す対象画像 1，2 の 2 枚を用いた．どちらの対象画像に対しても GRAPE によって構築された探索アルゴリズムは PSO より高い性能を示していることが確認できる．このことからも，提案手法によってテンプレートマッチングに有効な探索アルゴリズムの獲得が行えたといえる．

6.4 まとめ

本章では GRAPE を用いて探索空間を探索するエージェントの行動プログラムを自動生成することで，探索アルゴリズムの進化を行い，関数最適化問題のベンチマーク関数とテンプレートマッチングを例に探索アルゴリズムの獲得実験を行った．提案手法によって構築された探索アルゴリズムは，PSO に比べて多峰性の関数に対して良好な結果を示すものであった．これは，探索アルゴリズムの構築時に用いた Rastrigin-d 関数が多峰性であったためと考えられる．また，テンプレートマッチングに対して適用した際にも，有効な探索アルゴリズムを獲得することができた．つまり，提案手法を用いることで，対象とする問題に対して有効な探索アルゴリズムが獲得できることを確認できた．

今後は，他の実問題に対しても有効な探索アルゴリズムを構築することができるか検証を行う予定である．さらに，今回の実験では各エージェントが均一の戦略をとる homogeneous モデルを用いたが，各エージェントが異なる戦略をもつ heterogeneous モデルについても検証を行う必要があると考えられる．

第7章　結論

7.1　本論文で得られた成果

本論文では一貫して，グラフ構造をプログラムの表現形式とする自動プログラミングに関する手法の提案を行った．各章で得られた成果を次にまとめる．

- GINによる画像変換の自動構築
  画像処理を問題対象として，画像変換を自動構築するGIN，FFGINの提案を行った．画像変換の自動構築実験の結果から，GIN，FFGINによって木構造では表現できない構造が獲得できることを確認した．また，FFGINでは構造をフィードフォワードネットワーク構造に制限することで，GINやACTITと比較して計算時間が短縮されることを示した．

- GINに基づく画像分類法
  GINを基に画像変換部，特徴量抽出部，演算部から構成される画像分類器の自動構築法であるGIN-ICを提案した．GIN-ICの大きな特徴は画像変換部での画像の補正や，特徴の強調処理が行える点である．通常の分類器とは異なり，GIN-ICでは画像分類に必要な一連の処理を一括して自動構築することが可能である．本論文では，GIN-ICを用いて多クラスのテクスチャ画像分類実験を行いその有効性の確認を行った．実験の結果から，画像変換部を含めた構造を自動構築することの有効性が示された．

- GRAPEによるプログラムの自動生成
  より任意のプログラムを自動生成することを目的として，グラフ構造を用いた自動プログラミングの手法であるGRAPEを提案した．GRAPEはグラフ構造をプログラムの表現形式としているためループの表現が容易にできる．また，“データセット”を用いることで複数のデータ型の取り扱いが可能になる．GRAPEをいくつかの自動プログラミングの問題に適用し，有効性の検証を行った．実験で扱った問題は，どれも木構造をプログラムの表現形式としている一般的なGPでは解くことができない．実験の結果，GRAPEを用いてループを含むプログラムの自動生成が可能であることが確認できた．さらに，進化的な探索も有効に働いていることが確認でき，GRAPEによって効率のよいプログラムの自動生成を行うことができていた．

- GRAPEによる探索アルゴリズムの進化
  GRAPEを用いて探索アルゴリズムの進化を行った．一般的に，解きたい問題に対して有効な探索アルゴリズムを開発するコストは非常に高く，その過程を自動化するメリットは大きい．GRAPEを用いてベンチマーク関数とテンプレートマッチングに

対する探索アルゴリズムの自動生成を行った結果，一般的な探索手法と比較しても性能の高い探索アルゴリズムを生成することができた．

以上の成果より，グラフ構造をプログラムの表現形式とすることの有効性を示すことができた．グラフ構造を採用する利点には，木構造を含む構造を表現可能であることやループ構造などの表現が容易であること，コンパクトな構造で目的の処理を表現可能であることなどが挙げられる．このような理由から，グラフ構造を用いることで効率的に複雑なプログラムの自動生成が行えたと考えられる．

7.2 今後の課題

ここでは，本研究の成果を基に自動プログラミングの研究に対しての今後の課題を述べる．

- より複雑，大規模なプログラムの自動生成
  自動プログラミングの研究は一定の成果を挙げているが，より複雑なプログラムや大規模なプログラムを自動生成することは大きな課題である．つまり，人間でも構築困難なアルゴリズムや考え付かないようなアルゴリズムの自動生成を達成することが期待される．本論文ではGRAPEによってソートプログラムの自動生成を行ったが，例えばクイックソートやそれを超える新しいソートアルゴリズムの獲得を行うことなどである．また，実システムに応用できるようなプログラムの自動生成ができれば大きな省力化が実現できる．

- プログラムの自動生成における効率的な探索
  先に述べたような，より複雑，大規模なプログラムを自動生成するためには効率的な探索アルゴリズムは欠かせない．その際，プログラム（アルゴリズム）の評価に最終的な出力の評価だけでなく，(部分）構造の評価や汎用性のあるプログラムであるかなどの評価を導入することが有効であると考えられる．また，プログラムの構造や処理内容を考慮したような遺伝操作を行うことができれば，より効率的な探索が行えるかもしれない．プログラムにおいての部分構造の定義や評価は非常に困難な問題であるが，重要かつ挑戦し甲斐のある問題である．これらを解決するような“自動プログラミングのための探索手法”を考案することができれば，大きなブレークスルーになると考えられる．

- 新たな応用分野
  本論文でも扱った画像処理の分野は自動プログラミングが成功を収めている分野の一つであるといえる．他にも自動プログラミングは回路の生成やロボット制御，エージェントの意思決定などの様々な分野で成功を収めているが，キラーアプリケーションとなりうる新たな応用分野を発見することはより大きな意味をもつ．本論文で扱った探索アルゴリズムの進化のような，新たなヒューリスティクスを自動プログラミングを用いて生成するハイパーヒューリスティクスは今後の発展に大きな可能性を秘めているかもしれない．近年では高速なハードウェアであるGPUを用いた自動プログラミングの研究が活発に行われており，高速に計算を行えることを生かして，新たな応用分野への自動プログラミングの発展が期待される．

今後は，より複雑，大規模なプログラムの自動生成を可能にする自動プログラミング手法を考案するとともに，新たな応用分野に対しての自動プログラミングの有効性を検証していきたい．

謝辞

本研究を進めるにあたり適切なご指導，ご助言をいただいたり，研究環境を整えてくださった長尾智晴教授に深く感謝申し上げます．

本論文をまとめるにあたり貴重なご指導，ご助言を頂きました有澤博教授，田村直良教授，森辰則教授，岡嶋克典准教授，富井尚志准教授に感謝申し上げます．

矢田紀子研究教員には学部生で長尾研究室に配属された時から先輩として，様々な面でお世話になりました，ありがとうございました．

また，長尾研究室の皆様，卒業生の皆様には貴重なアドバイス，討論をしていただいただけでなく，研究室生活，その他においてもお世話になりました，ありがとうございました．

他にも，横浜国立大学　工学府　濱上研究室の皆様との月に一回のゼミは研究を進めるにあたり励みになりました．

日本学術振興会には平成20年度より特別研究員（DC2）に採用していただき研究奨励金などのご支援をいただきました．

最後に，博士課程後期に進学するにあたって，経済的な支援と共に，温かく見守ってくれた両親に感謝します．

参考文献

[1] John H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[2] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison Wesley, 1989.

[3] 安居院猛, 長尾智晴. ジェネティックアルゴリズム. 昭晃堂, 1993.

[4] 北野宏明. 遺伝的アルゴリズム. 産業図書, 1993.

[5] 伊庭斉志. 遺伝的アルゴリズムの基礎. オーム社, 1994.

[6] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, 1992.

[7] John R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs.* MIT Press, 1994.

[8] Wolfgang Banzhaf, Peter Nordin, Robert E. Keller, and Frank D. Francone. *Genetic Programming – An Introduction; On the Automatic Evolution of Computer Programs and its Applications.* Morgan Kaufmann Publishers, 1998.

[9] 伊庭斉志, 新田徹（訳）, Wolfgang Banzhaf, Peter Nordi, Robert E. Keller, Frank D.Francone（著）. 遺伝的プログラミング. 科学技術出版, 2000.

[10] 伊庭斉志. 遺伝的プログラミング. 東京電機大学出版局, 1996.

[11] Lawrence J. Fogel, Alvin J. Owens, and Michael J. Walsh. *Artificial Intelligence through Simulated Evolution.* John Wiley, 1966.

[12] Thomas Back, Frank Hoffmeister, and Hans paul Schwefel. A survey of evolution strategies. In *Proceedings of the Fourth International Conference on Genetic Algorithms (ICGA '91)*, pp. 2–9, San Diego, 1991. Morgan Kaufmann Publishers.

[13] Macro Dorigo, V Maniezzo, and A Colorni. Ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, Vol. 26, No. 1, pp. 29–41, 1996.

[14] Marco Dorigo and Thomas Stutzle. *Ant Colony Optimization.* The MIT Press, 2004.

[15] James F. Kennedy and Russell C. Eberhart. Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks*, Vol. IV, pp. 1942–1948, Perth, Australia, 1995.

[16] James F. Kennedy, Russell C. Eberhart, and Yuhui Shi. *Swarm Intelligence.* Morgan Kaufmann Publishers, 2001.

[17] Hiroshi Satoh, Masayuki Yamamura, and Shigenobu Kobayashi. Minimal generation gap model for considering both exploration and exploitations. In *Proceedings of the IIZUKA '96*, pp. 494–497, 1996.

[18] 佐藤浩, 小野功, 小林重信. 遺伝的アルゴリズムにおける世代交代モデルの提案と評価. 人工知能学会誌, Vol. 12, No. 5, pp. 734–744, 1997.

[19] Hajime Kita, Isao Ono, and Shigenobu Kobayashi. Multi-parental extension of the unimodal normal distribution crossover for real-coded genetic algorithms. In *Proceedings of the 1999 Congress on Evolutionary Computation (CEC '99)*, Vol. 2, pp. 1581–1587, 1999.

[20] Shigeyoshi Tsutsui, Masayuki Yamamura, and Takahide Higuchi. Multi-parent recombination with simplex crossover in real coded genetic algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference 1999 (GECCO '99)*, pp. 657–664, 1999.

[21] Kalyanmoy Deb, Ashish Anand, and Dhiraj Joshi. A computationally efficient evolutionary algorithm for real-parameter optimization. *Evolutionary Computation*, Vol. 10, No. 4, pp. 371–395, 2002.

[22] Peter J. Angeline and Jordan B. Pollack. The evolutionary induction of subroutines. In *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society*, pp. 236–241, Bloomington, Indiana, USA, 1992. Lawrence Erlbaum.

[23] Peter J. Angeline and Jordan B. Pollack. Evolutionary module acquisition. In *Proceedings of the Second Annual Conference on Evolutionary Programming*, pp. 154–163, La Jolla, CA, USA, 25-26 1993.

[24] Justinian P. Rosca and Dana H. Ballard. Learning by adapting representations in genetic programming. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence (WCCI '94)*, Orlando, Florida, USA, 27-29 June 1994. IEEE Press.

[25] Lee Spector. Simultaneous evolution of programs and their control structures. In Peter J. Angeline and K. E. Kinnear, Jr., editors, *Advances in Genetic Programming 2*, pp. 137–154. MIT Press, 1996.

[26] Peter Nordin and Wolfgang Banzhaf. Genetic reasoning evolving proofs with genetic search. In *Proceedings of the Second Annual Conference on Genetic Programming*, pp.

255–260, Stanford University, CA, USA, 13-16 July 1997. Morgan Kaufmann Publishers.

[27] Kenneth E. Kinnear, Jr. Generality and difficulty in genetic programming: Evolving a sort. In *Proceedings of the Fifth International Conference on Genetic Algorithms (ICGA '93)*, pp. 287–294, University of Illinois at Urbana-Champaign, 17-21 July 1993. Morgan Kaufmann Publishers.

[28] Kenneth E. Kinnear, Jr. Evolving a sort: Lessons in genetic programming. In *Proceedings of the 1993 International Conference on Neural Networks*, Vol. 2, pp. 881–888, San Francisco, USA, 28 March-1 April 1993. IEEE Press.

[29] William B. Langdon. Evolving data structures using genetic programming. In *Proceedings of the Sixth International Conference on Genetic Algorithms (ICGA '95)*, pp. 295–302, Pittsburgh, PA, USA, 15-19 July 1995. Morgan Kaufmann Publishers.

[30] William B. Langdon. *Genetic Programming and Data Structures: Genetic Programming + Data Structures = Automatic Programming!*, Vol. 1 of *Genetic Programming*. Kluwer Academic Publishers, 1998.

[31] David J. Montana. Strongly typed genetic programming. *Evolutionary Computation*, Vol. 3, No. 2, pp. 199–230, 1995.

[32] Astro Teller. Learning mental models. In *Proceedings of the Fifth Workshop on Neural Networks: An International Conference on Computational Intelligence: Neural Networks, Fuzzy Systems, Evolutionary Programming, and Virtual Reality*, 1993.

[33] Astro Teller. Genetic programming, indexed memory, the halting problem, and other curiosities. In *Proceedings of the Seventh annual Florida Artificial Intelligence Research Symposium*, pp. 270–274, Pensacola, Florida, USA, May 1994. IEEE Press.

[34] Astro Teller. Turing completeness in the language of genetic programming with indexed memory. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence (WCCI '94)*, Vol. 1, pp. 136–141, Orlando, Florida, USA, 27-29 June 1994. IEEE Press.

[35] Markus Brameier and Wolfgang Banzhaf. A comparison of linear genetic programming and neural networks in medical data mining. *IEEE Transactions on Evolutionary Computation*, Vol. 5, No. 1, pp. 17–26, February 2001.

[36] Conor Ryan, J. J. Collins, and Michael O'Neill. Grammatical evolution: Evolving programs for an arbitrary language. In *Proceedings of the First European Workshop on Genetic Programming (EuroGP '98)*, Vol. 1391 of *LNCS*, pp. 83–95, Paris, 14-15 April 1998. Springer-Verlag.

[37] Michael O'Neill and Conor Ryan. Grammatical evolution. *IEEE Transactions on Evolutionary Computation*, Vol. 5, No. 4, pp. 349–358, August 2001.

[38] Michael O'Neill and Conor Ryan. *Grammatical Evolution: Evolutionary Automatic Programming in a Arbitrary Language*, Vol. 4 of *Genetic Programming*. Kluwer Academic Publishers, 2003.

[39] Lee Spector. Autoconstructive evolution: Push, pushGP, and pushpop. In *Proceedings of the Genetic and Evolutionary Computation Conference 2001 (GECCO '01)*, pp. 137–146, San Francisco, California, USA, 7-11 July 2001. Morgan Kaufmann Publishers.

[40] Lee Spector and Alan Robinson. Genetic programming and autoconstructive evolution with the push programming language. *Genetic Programming and Evolvable Machines*, Vol. 3, No. 1, pp. 7–40, March 2002.

[41] Lee Spector, Jon Klein, and Maarten Keijzer. The push3 execution stack and the evolution of control. In *Proceedings of the Genetic and Evolutionary Computation Conference 2005 (GECCO '05)*, Vol. 2, pp. 1689–1696, Washington DC, USA, 25-29 June 2005. ACM Press.

[42] Simon M. Lucas. Exploiting reflection in object oriented genetic programming. In *Proceedings of the Seventh European Conference on Genetic Programming (EuroGP '04)*, Vol. 3003 of *LNCS*, pp. 369–378, Coimbra, Portugal, 5-7 April 2004. Springer-Verlag.

[43] Alexandros Agapitos and Simon M. Lucas. Learning recursive functions with object oriented genetic programming. In *Proceedings of the Ninth European Conference on Genetic Programming (EuroGP '06)*, Vol. 3905 of *LNCS*, pp. 166–177, Budapest, Hungary, 10-12 April 2006. Springer-Verlag.

[44] Alexandros Agapitos and Simon M. Lucas. Evolving efficient recursive sorting algorithms. In *Proceedings of the 2006 IEEE Congress on Evolutionary Computation (CEC '06)*, pp. 9227–9234, Vancouver, 6-21 July 2006. IEEE Press.

[45] Astro Teller and Manuela Veloso. Program evolution for data mining. *The International Journal of Expert Systems*, Vol. 8, No. 3, pp. 216–236, 1995.

[46] Astro Teller and Manuela Veloso. Algorithm evolution for face recognition: What makes a picture difficult. In *International Conference on Evolutionary Computation*, pp. 608–613, Perth, Australia, 1-3 December 1995. IEEE Press.

[47] Astro Teller and Manuela 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.

[48] Riccardo Poli. Evolution of graph-like programs with parallel distributed genetic programming. In *Proceedings of the Seventh International Conference on Genetic Algorithms (ICGA '97)*, pp. 346–353, Michigan State University, East Lansing, MI, USA, 19-23 July 1997. Morgan Kaufmann Publishers.

[49] Julian F. Miller and Peter Thomson. Cartesian genetic programming. In *Proceedings of the Third European Conference on Genetic Programming (EuroGP '00)*, Vol. 1802 of *LNCS*, pp. 121–132, Edinburgh, 15-16 April 2000. Springer-Verlag.

[50] Julian F. Miller and Stephen L. Smith. Redundancy and computational efficiency in cartesian genetic programming. *IEEE Transactions on Evolutionary Computation*, Vol. 10, No. 2, pp. 167–174, April 2006.

[51] James Alfred Walker and Julian F. Miller. Evolution and acquisition of modules in cartesian genetic programming. In *Proceedings of the Seventh European Conference on Genetic Programming (EuroGP '04)*, Vol. 3003 of *LNCS*, pp. 187–197, Coimbra, Portugal, 5-7 April 2004. Springer-Verlag.

[52] James Alfred Walker and Julian F. Miller. The automatic acquisition, evolution and reuse of modules in cartesian genetic programming. *IEEE Transactions on Evolutionary Computation*, Vol. 12, No. 4, pp. 397–417, August 2008.

[53] 平澤宏太郎, 大久保雅文, 片桐広伸, 古月敬之, 村田純一. 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, Vol. 121, No. 6, pp. 1001–1009, 2001.

[54] Kotaro Hirasawa, Masafumi Okubo, Jinglu Hu, and Junichi Murata. Comparison between genetic network programming (GNP) and genetic programming (GP). In *Proceedings of the 2001 Congress on Evolutionary Computation (CEC '01)*, pp. 1276–1282, COEX, World Trade Center, 159 Samseong-dong, Gangnam-gu, Seoul, Korea, 27-30 May 2001. IEEE Press.

[55] Hironobu Katagiri, Kotaro Hirasawa, Jinglu Hu, and Junichi Murata. Network structure oriented evolutionary model-genetic network programming-and its comparison with genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference 2001 (GECCO '01) Late Breaking Papers*, pp. 219–226, San Francisco, California, USA, 9-11 July 2001.

[56] Toru Eguchi, Kotaro Hirasawa, Jinglu Hu, and Noriko Ota. A study of evolutionary multiagent models based on symbiosis. *IEEE Transactions on Systems, Man and Cybernetics Part B*, Vol. 36, No. 1, pp. 179–193, 2006.

[57] Wolfgang Kantschik and Wolfgang Banzhaf. Linear-graph GP—A new GP structure. In *Proceedings of the Fifth European Conference on Genetic Programming (EuroGP '02)*, Vol. 2278 of *LNCS*, pp. 83–92, Kinsale, Ireland, 3-5 April 2002. Springer-Verlag.

[58] 片岡寛明, 原章, 長尾智晴. 遺伝的オートマトンGAUGE. 情報処理学会論文誌, Vol. 44, No. 12, pp. 3232–3241, 2003.

[59] Walter Alden Tackett. Genetic programming for feature discovery and image discrimination. In *Proceedings of the Fifth International Conference on Genetic Algorithms*

(ICGA '93), pp. 303–309, University of Illinois at Urbana-Champaign, 17-21 July 1993. Morgan Kaufmann Publishers.

[60] Mengjie Zhang and Christopher Graeme Fogelberg. Genetic programming for image recognition: An LGP approach. In *Applications of Evolutionary Computing, EvoWorkshops 2007*, Vol. 4448 of *LNCS*, pp. 340–350, Valencia, Spain, 11-13 April 2007. Springer-Verlag.

[61] Brian Lam and Vic Ciesielski. Discovery of human-competitive image texture feature extraction programs using genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference 2004 (GECCO '04), Part II*, Vol. 3103 of *LNCS*, pp. 1114–1125, Seattle, WA, USA, 26-30 June 2004. Springer-Verlag.

[62] Melanie Aurnhammer. Evolving texture features by genetic programming. In *Applications of Evolutionary Computing, EvoWorkshops 2007*, Vol. 4448 of *LNCS*, pp. 351–358, Valencia, Spain, 11-13 April 2007. Springer-Verlag.

[63] 長尾智晴. 進化的画像処理. 昭晃堂, 2002.

[64] Shinya Aoki and Tomoharu Nagao. Automatic construction of tree-structural image transformation using genetic programming. In *Proceedings of the 1999 International Conference on Image Processing (ICIP '99)*, Vol. 1, pp. 529–533, Kobe, Japan, 24-28 October 1999. IEEE Press.

[65] 青木紳也, 長尾智晴. 木構造状画像変換の自動構築法 ACTIT. 映像情報メディア学会誌, Vol. 53, No. 6, pp. 888–894, 1999.

[66] Yuta Nakano and Tomoharu Nagao. 3D medical image processing using 3D-ACTIT; automatic construction of tree-structural image transformation. In *Proceedings of the International Workshop on Advanced Image Technology 2004 (IWAIT '04)*, pp. 529–533, Singapore, 12-13 January 2004.

[67] Yuta Nakano and Tomoharu Nagao. Automatic extraction of internal organs region from 3D PET image data using 3D-ACTIT. In *Proceedings of the International Workshop on Advanced Image Technology 2006 (IWAIT '06)*, Okinawa, Japan, 9-10 January 2006.

[68] 中野雄太, 長尾智晴, 竹林茂生. 3D-ACTIT による 3 次元拡散強調画像からの異常信号領域の自動抽出. 日本医用画像工学誌, Vol. 25, No. 5, pp. 362–370, 2007.

[69] Yuta Nakano and Tomoharu Nagao. Automatic construction of abnormal signal extraction processing from 3D diffusion weighted image. In *Proceedings of the International Workshop on Advanced Image Technology 2007 (IWAIT '07)*, Bangkok, Thailand, 8-9 January 2007.

[70] Yuta Nakano and Tomoharu Nagao. Automatic construction of moving object segmentation from video images using 3D-ACTIT. In *Proceedings of The 2007 IEEE Interna-*

tional Conference on Systems, Man, and Cybernetics (SMC '07), pp. 1153–1158, Montreal, Canada, 7-11 October 2007.

[71] 中野雄太, 長尾智晴. 3 次元画像処理自動構築システム 3D-ACTIT の提案と PET 画像への応用. 医用画像情報学会誌, Vol. 24, No. 4, pp. 119–125, 2008.

[72] 中野雄太, 長尾智晴. 3D-ACTIT による 3D-PET 画像からの肝臓領域自動抽出. 映像情報メディア学会誌, Vol. 62, No. 3, pp. 424–434, 2008.

[73] Wataru Fujishima and Tomoharu Nagao. PT-ACTIT; parameter tunable-automatic construction of tree-structural image transformation. In *Proceedings of the International Workshop on Advanced Image Technology 2004 (IWAIT '04)*, Singapore, 12-13 January 2004.

[74] 藤嶋航, 長尾智晴. GP による構造最適化と GA による数値最適化を併用した画像処理自動生成法 PT-ACTIT. 映像情報メディア学会誌, Vol. 59, No. 11, pp. 1689–1693, 2005.

[75] Wataru Fujishima and Tomoharu Nagao. Genetic matrix algorithm; simultaneous optimization of structure and numerical parameters. *IEEJ Transactions of Electrical & Electronic Engineering (TEEE)*, Vol. 3, No. 1, pp. 84–91, 2008.

[76] Simon Harding and Wolfgang Banzhaf. Fast genetic programming on GPUs. In *Proceedings of the Tenth European Conference on Genetic Programming (EuroGP '07)*, Vol. 4445 of *LNCS*, pp. 90–101, Valencia, Spain, 11-13 April 2007. Springer-Verlag.

[77] Simon Harding. Evolution of image filters on graphics processor units using cartesian genetic programming. In *Proceedings of the 2008 IEEE World Congress on Computational Intelligence (WCCI '08)*, Hong Kong, 1-6 June 2008. IEEE Press.

[78] Darren M. Chitty. A data parallel approach to genetic programming using programmable graphics hardware. In *Proceedings of the Genetic and Evolutionary Computation Conference 2007 (GECCO '07)*, Vol. 2, pp. 1566–1573, London, UK, 7-11 July 2007. ACM Press.

[79] Jun Ando and Tomoharu Nagao. Fast evolutionary image processing using multi-GPUs. In *Proceedings of The 2007 IEEE International Conference on Systems, Man, and Cybernetics (SMC '07)*, pp. 2927–2932, Montreal, Canada, 7–11October 2007.

[80] 長谷川純一, 久保田浩明, 鳥脇純一郎. サンプル図形呈示方法による画像処理エキスパートシステム IMPRESS. 電子情報通信学会論文誌 D, Vol. J70-D, No. 11, pp. 2147–2153, 1987.

[81] 濱田敏弘, 清水昭伸, 長谷川純一, 鳥脇純一郎. ビジョンエキスパートシステム IMPRESS における画像処理手順の逐次的集約法とその性能評価. 電子情報通信学会論文誌 D-II, Vol. J82-D-II, No. 11, pp. 1982–1989, 1999.

[82] 依田育士, 山本和彦, 山田博三. GA による構造的モルフォロジー手順の獲得. 電子情報通信学会論文誌 D-II, Vol. J78-D-II, No. 12, pp. 1758–1766, 1995.

[83] Inman Harvey and Adrian Thompson. Through the labyrinth evolution finds a way: A silicon ridge. In *Proceedings of the First International Conference on Evolvable Systems: From Biology to Hardware (ICES '96)*, Vol. 1259 of *LNCS*, pp. 406–422. Springer-Verlag, 1996.

[84] Vesselin K. Vassilev and Julian F. Miller. The advantages of landscape neutrality in digital circuit evolution. In *Proceedings of the Third International Conference on Evolvable Systems (ICES '00)*, Vol. 1801 of *LNCS*, pp. 252–263. Springer-Verlag, 2000.

[85] Tina Yu and Julian F. Miller. Neutrality and the evolvability of boolean function landscape. In *Proceedings of the Fourth European Conference on Genetic Programming (EuroGP '01)*, Vol. 2038 of *LNCS*, pp. 204–217. Springer-Verlag, 2001.

[86] Shingo Mabu, Kotaro Hirasawa, Yuko Matsuya, and Jinglu Hu. Genetic network programming for automatic program generation. *Journal of Advanced Computational Intelligence and Intelligent Informatics*, Vol. 9, No. 4, pp. 430–436, 2005.

[87] Sean Luke and Lee Spector. A comparison of crossover and mutation in genetic programming. In *Proceedings of the Second Annual Conference on Genetic Programming*, pp. 240–248. Morgan Kaufmann Publishers, 1997.

[88] David H. Wolpert and William G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, Vol. 1, No. 1, pp. 67–82, 1997.

[89] Mihai Oltean. Evolving evolutionay algorithm using linear genetic programming. *Evolutionary Computation*, Vol. 13, No. 3, pp. 387–410, 2005.

[90] Laura Dioşan and Mihai Oltean. Evolving crossover operators for function optimization. In *Proceedings of the Ninth European Conference on Genetic Programming (EuroGP '06)*, Vol. 3905 of *LNCS*, pp. 97–108, Budapest, Hungary, 10-12 April 2006. Springer-Verlag.

[91] Laura Dioşan and Mihai Oltean. Evolving the structure of the particle swarm optimization algorithms. In *Proceedings of the European Conference on Evolutionary Computation in Combinatorial Optimization (EvoCOP '06)*, Vol. 3906 of *LNCS*, pp. 25–36, Budapest, Hungary, 10-12 April 2006. Springer-Verlag.

[92] Eric Soubeiga. *Development and application of hyperheuristics to personnel scheduling*. PhD thesis, University of Nottingham, 2003.

[93] Peter Ross, Javier G. Marín-Blázquez, Sonia Schulenburg, and Emma Hart. Learning a procedure that can solve hard bin-packing problems: a new GA-based approach to hyperheuristics. In *Proceedings of the Genetic and Evolutionary Computation Conference*

2003 (GECCO '03), Vol. 2724 of *LNCS*, pp. 1295–1306, Chicago, Illinois, USA, 12-16 July 2003.

[94] Edmund K. Burke, Graham Kendall, and Eric Soubeiga. A tabu-search hyperheuristic for timetabling and rostering. *Journal of Heuristics*, Vol. 9, No. 6, pp. 451–470, 2003.

[95] Edmund K. Burke, Matthew R. Hyde, and Graham Kendall. Evolving bin packing heuristics with genetic programming. In *Parallel Problem Solving from Nature - PPSN IX*, Vol. 4193 of *LNCS*, pp. 860–869, Reykjavik, Iceland, 9-13 September 2006. Springer-Verlag.

[96] Edmund K. Burke, Matthew R. Hyde, Graham Kendall, and John Woodward. Automatic heuristic generation with genetic programming: evolving a jack-of-all-trades or a master of one. In *Proceedings of the Genetic and Evolutionary Computation Conference 2007 (GECCO '07)*, Vol. 2, pp. 1559–1565, London, UK, 7-11 July 2007. ACM Press.

[97] Alex S. Fukunaga. Automated discovery of composite SAT variable selection heuristics. In *Proceedings of the National Conference on Artificial Intelligence (AAAI '02)*, pp. 641–648, 2002.

[98] Alex S. Fukunaga. Evolving local search heuristics for SAT using genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference 2004 (GECCO '04), Part II*, Vol. 3103 of *LNCS*, pp. 483–494, Seattle, WA, USA, 26-30 June 2004. Springer-Verlag.

[99] Alex S. Fukunaga. Automated discovery of local search heuristics for satisfiability testing. *Evolutionary Computation*, Vol. 16, No. 1, pp. 31–61, Spring 2008.

[100] Rajeev Kumar, Ashwin H. Joshi, Krishna K. Banka, and Peter I. Rockett. Evolution of hyperheuristics for the biobjective 0/1 knapsack problem by multiobjective genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference 2008 (GECCO '08)*, pp. 1227–1234, Atlanta, GA, USA, 12-16 July 2008. ACM Press.

[101] Nelishia Pillay and Wolfgang Banzhaf. A genetic programming approach to the generation of hyper-heuristics for the uncapacitated examination timetabling problem. In *Proceedings of the EPIA Workshops*, Vol. 4874 of *LNCS*, pp. 223–234, Guimarães, Portugal, 2007. Springer-Verlag.

[102] Nelishia Pillay. An analysis of representations for hyper-heuristics for the uncapacitated examination timetabling problem in a genetic programming system. In *Proceedings of the 2008 annual research conference of the South African Institute of Computer Scientists and Information Technologists on IT research in developing countries (SAICSIT '08)*, pp. 188–192, Wilderness, South Africa, 2008. ACM Press.

[103] Larry J. Eshelman and J. David Schaffer. Real coded genetic algorithms and interval-schemata. In *Foundations of Genetic Algorithms 2*, pp. 187–202, San Mateo, CA, 1993. Morgan Kaufmann Publishers.

[104] Isao Ono and Shigenobu Kobayashi. A real-coded genetic algorithm for function optimization using unimodal normal distribution crossover. In *Proceedings of the Seventh International Conference on Genetic Algorithms (ICGA '97)*, pp. 246–253, Michigan State University, East Lansing, MI, USA, 19-23 July 1997. Morgan Kaufmann Publishers.

[105] Russell C. Eberhart and Yuhui Shi. Comparing inertia weights and constriction factors in particle swarm optimization. In *Proceedings of the 2000 Congress on Evolutionary Computation (CEC '00)*, pp. 84–89, 2000.

研究業績リスト

論文

1. 白川真一，長尾智晴：RFCNによる連続値空間上での自律エージェントの行動制御，電気学会論文誌C，Vol.128，No.5，pp.762–769 (2007)
2. 白川真一，荻野慎太郎，長尾智晴：Genetic Image Networkによる画像変換の自動構築，情報処理学会論文誌：数理モデル化と応用(TOM 19)，Vol.48，No.SIG 19，pp.117–126 (2007)
3. 白川真一，長尾智晴：Graph Structured Program Evolutionによるプログラムの自動生成，電気学会論文誌C，Vol.129，No.3，pp.70–80 (2008)
4. Shinichi Shirakawa, Shintaro Ogino, and Tomoharu Nagao: Dynamic Ant Programming for Automatic Construction of Programs, IEEJ Transactions on Electrical and Electronic Engineering (TEEE), Vol.3, Issue 5, pp.540–548 (2008)
5. 田澤和子，白川真一，長尾智晴：FCNによる自律エージェントの行動制御と行動解析～タルタロス問題への応用～，知能と情報（日本知能情報ファジィ学会誌），Vol.20，No.5，pp.800–809 (2008)

翻訳論文

1. Shinichi Shirakawa and Tomoharu Nagao: Action Control of Autonomous Agents in Continuous Valued Space Using RFCN, Electronics and Communications in Japan, Vol.91, No.2, pp.31–39 (2008)（論文1の翻訳）
2. Shinichi Shirakawa and Tomoharu Nagao: Automatic Generation of Programs Using Graph Structured Program Evolution, Electronics and Communications in Japan（論文3の翻訳，掲載予定）

国際会議

1. Shinichi Shirakawa and Tomoharu Nagao: Genetic Image Network (GIN): Automatically Construction of Image Processing Algorithm, Proceedings of the International Workshop on Advanced Image Technology 2007 (IWAIT '07), P3-34, pp.643–648, Bangkok, Thailand, 8-9 January (2007)

2. Shinichi Shirakawa, Shintaro Ogino, and Tomoharu Nagao: Graph Structured Program Evolution, Proceedings of the Genetic and Evolutionary Computation Conference 2007 (GECCO '07), Vol.2, pp.1686–1693, London, England, 7-11 July (2007)

3. Shinichi Shirakawa and Tomoharu Nagao: Evolution of Sorting Algorithm Using Graph Structured Program Evolution, Proceedings of the 2007 IEEE International Conference on Systems, Man and Cybernetics (SMC '07), pp.1256–1261, Montreal, Canada, 7-10 October (2007)

4. Shinichi Shirakawa and Tomoharu Nagao: Feed Forward Genetic Image Network: Toward Efficient Automatic Construction of Image Processing Algorithm, Advances in Visual Computing: Proceedings of the 3rd International Symposium on Visual Computing (ISVC '07) Part II, Vol.4842 of LNCS, pp.287–297, Lake Tahoe, Nevada, USA, 26-28 November (2007)

5. Shinichi Shirakawa and Tomoharu Nagao: Evolutionary Algorithm Considering Program Size: Efficient Program Evolution Using GRAPE, Proceedings of the Genetic and Evolutionary Computation Conference 2008 (GECCO '08), Late-Breaking Papers, pp.2217–2222, Atlanta, GA, USA, 12-16 July (2008)

6. Shinichi Shirakawa, Shiro Nakayama, and Tomoharu Nagao: Genetic Image Network for Image Classification, 11th European Workshop on Evolutionary Computation in Image Analysis and Signal Processing (EvoIASP '09), Tübingen, Germany, 15-17 April (2009) (accepted)

7. Shinichi Shirakawa and Tomoharu Nagao: Evolution of Search Algorithms Using Graph Structured Program Evolution, 12th European Conference on Genetic Programming (EuroGP '09), Tübingen, Germany, 15-17 April (2009) (accepted)

8. Shinichi Shirakawa and Tomoharu Nagao: Evolutionary Image Segmentation Based on Multiobjective Clustering, 2009 IEEE Congress on Evolutionary Computation (CEC '09), Trondheim, Norway, 18-21 May (2009) (accepted)

国内学会

1. 白川真一，長尾智晴：進化型ニューラルネットワークによる連続値環境下でのエージェントの行動制御，電子情報通信学会総合大会 ISS 特別企画学生ポスターセッション，D-SP-67 (2005)

2. 白川真一，長尾智晴：連続値空間上での進化型神経回路網による自律エージェントの行動制御，平成 17 年電気学会電子・情報・システム部門大会，GS15-6，pp.1045–1049 (2005)

3. 白川真一，長尾智晴：ネットワーク構造状画像変換の自動構築，第 5 回情報科学技術フォーラム（FIT 2006），F-016，pp.279–280 (2006)

4. 白川真一，荻野慎太郎，長尾智晴：Genetic Image Network による画像変換の自動構築，情報処理学会研究報告 第 63 回 数理モデル化と問題解決（MPS）研究会，Vol.2007，No.19，2007-MPS-63，pp.93–96 (2007)

5. 白川真一，長尾智晴：Genetic Image Network による複数出力画像変換の自動構築，電子情報通信学会総合大会，D-8-2 (2007)

6. 白川真一，長尾智晴：Graph Structured Program Evolution による複雑なプログラムの自動生成とその解析，情報処理学会研究報告 第 66 回 数理モデル化と問題解決（MPS）研究会，Vol.2007，No.86，2007-MPS-66，pp.21–24 (2007)

7. 白川真一，長尾智晴：多目的クラスタリングに基づく進化的画像領域分割法，進化計算シンポジウム 2007 講演論文集，pp.59–62 (2007)

8. 矢田紀子，白川真一，長尾智晴，内川惠二：色覚異常者のカテゴリカル色知覚モデルの構築，情報処理学会研究報告 第 68 回 数理モデル化と問題解決（MPS）研究会，Vol.2008，No.17，2008-MPS-68，pp.13–16 (2008)

9. 白川真一，長尾智晴：プログラムサイズを考慮した自動プログラミングのための進化アルゴリズムの提案，情報処理学会研究報告 第 68 回 数理モデル化と問題解決（MPS）研究会，Vol.2008，No.17，2008-MPS-68，pp.17–20 (2008)

10. 矢田紀子，白川真一，長尾智晴，内川惠二：FFFCN を用いた色覚異常者のカテゴリカル色知覚モデル，電子情報通信学会総合大会，D-2-13 (2008)

11. 白川真一，長尾智晴：Graph Structured Program Evolution を用いた自律エージェントの行動制御，電子情報通信学会総合大会，D-8-16 (2008)

12. 中野雄太，白川真一，長尾智晴：特徴考慮型 GIN による画像処理の自動構築，電子情報通信学会総合大会，D-11-29 (2008)

13. 白川真一，中山史朗，長尾智晴：Genetic Image Network に基づく画像分類法の提案，情報処理学会研究報告 第 71 回 数理モデル化と問題解決（MPS）研究会，Vol.2008，No.85，2008-MPS-71，pp.9–12 (2008)

14. 白川真一，長尾智晴：Graph Structured Program Evolution による探索アルゴリズムの進化，第 18 回インテリジェント・システム・シンポジウム（FAN 2008），D1-1, pp.209–214 (2008)

15. 白川真一，長尾智晴：Graph Structured Program Evolution への自動関数定義の導入とその評価，進化計算シンポジウム 2008 講演論文集，pp.162–166 (2008)

16. 中山史朗，白川真一，長尾智晴：弱識別器に Genetic Image Network を用いた画像分類法，第 36 回知能システムシンポジウム (2009)（発表予定）

17. 白川真一，長尾智晴：多目的クラスタリングに基づく画像領域分割とその解選択の検討，第 36 回知能システムシンポジウム (2009)（発表予定）

18. 石堂眞大，白川真一，長尾智晴：グラフ構造のプログラム自動生成手法への ADF の導入，電子情報通信学会総合大会 (2009)（発表予定）

特許

1. 特願 2006-186769 ネットワーク型画像処理装置，発明者:長尾智晴，白川真一，小川原也

2. 特願 2007-31592 進化計算システム及び進化計算法，発明者：長尾智晴，白川真一

付録A　本論文で用いた画像処理フィルター一覧

フィルタ名	記号	処理内容
1入力1出力フィルタ		
Mean	-	平均値フィルタ
Max	M	最大値フィルタ
Min	m	最小値フィルタ
Sobel	d	ゾーベルフィルタ
Light Edge	E	白エッジ強調
Dark Edge	e	黒エッジ強調
Light Pixel	T	しきい値（平均階調値）以下の画素を黒にする
Dark Pixel	t	しきい値（平均階調値）以下の画素を白にする
Large Area	S	分割領域面積平均値よりも小さい領域を白にする
Small Area	s	分割領域面積平均値よりも大きい領域を白にする
Inversion	i	反転フィルタ
Deploy	K	分散フィルタ
Gamma	G	ガンマ補正フィルタ（$\gamma = 2$）
Contraction	x	収縮フィルタ
Expansion	X	膨張フィルタ
Prewitt	z	Prewitt フィルタ
Laplacian	g	ラプラシアンフィルタ
High Area Per Box	P	外接矩形に対する充填率が高い（90% 以上）孤立領域を残す
Low Area Per Box	p	外接矩形に対する充填率が低い（90% 未満）孤立領域を残す
Square Box	R	外接矩形の縦横比が 1.0 に近い（0.9～1.1）孤立領域を残す
Rectangular Box	r	外接矩形の縦横比が 1.0 に近い（0.9～1.1）孤立領域を消す
High Circularity	C	外接矩形に対する孤立領域面積の真円度が 1.0 に近い（0.95～1.05）孤立領域を残す
Low Circularity	c	外接矩形に対する孤立領域面積の真円度が

		1.0 に近い（0.95～1.05）孤立領域を消す
Linear Transformation	H	線形変換フィルタ
Binarization	N	2 値化フィルタ（平均階調値）
Binarization Discriminant Analysis	n	2 値化フィルタ（判別分析法）
Nop	nop	nop フィルタ（何もしない）
2 入力 1 出力フィルタ（f_1：入力画像 1，f_2：入力画像 2）		
LogicalSum	L	論理和（$\max(f_1, f_2)$）
LogicalProd	l	論理積（$\min(f_1, f_2)$）
AlgebraicSum	A	代数和（$f_1 + f_2 - (f_1 \times f_2 \div V_{max})$）
AlgebraicProd	a	代数積（$f_1 \times f_2 \div V_{max}$）
BoundedSum	B	限界和（$f_1 + f_2$）
BoundedProd	b	限界積（$f_1 + f_2 - V_{max}$）
DrasticSum	u	激烈和（$f_1 = 0 \rightarrow f_2$， $f_2 = 0 \rightarrow f_1$，$f_1, f_2 \neq 0 \rightarrow V_{max}$）
DrasticProd	U	激烈積（$f_1 = V_{max} \rightarrow f_2$， $f_2 = V_{max} \rightarrow f_1, f_1, f_2 \neq V_{max} \rightarrow 0$）
Difference	D	差分フィルタ（$\mathrm{abs}(f_1, f_2)$）
OutIn1	nop1	f_1 を出力
OutIn2	nop2	f_2 を出力