複雑な構造に対する新しい進化的計算法の開発とその動画像処理への応用

１７３０００４４

平成１７年度～平成１８年度科学研究費補助金（　基盤研究（Ｂ）　）研究成果報告書

平成19年5月

研究代表者　　長尾　智晴
国立大学法人 横浜国立大学
大学院環境情報研究院　教授

＜はしがき＞

本研究は，「複雑な構造に対する新しい進化的計算法の開発とその動画像処理への応用」と題して平成 17 年度から平成 18 年度の 2 年間に渡って実施された科学研究費補助金（基盤研究（B））の研究成果をまとめたものである．本研究の研究組織，交付決定額（配分額），研究発表，研究成果による工業所有権の出願・取得状況を次に示す．

研究組織

研究代表者　：　長尾 智晴　（国立大学法人 横浜国立大学大学院環境情報研究院教授）

交付決定額（配分額）　（金額単位：円）

	直接経費	間接経費	合計
平成 17 年度	4,200,000	0	4,200,000
平成 18 年度	2,900,000	0	2,900,000
総　計	7,100,000	0	7,100,000

研 究 発 表

(1) **学会誌等**

1. 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)
2. Shumei Liu, and Tomoharu Nagao：Hierarchical XCS and Its Application to Financial Time Series Forecasting, IEEJ Transactions of Electrical & Electronic Engineering, Vol.1, No.4, pp.417-425 (2006)
3. Yuta Nakano and Tomoharu Nagao: Automatic extraction of internal organs region from 3D PET image data using 3D-ACTIT; Proc. of IWAIT-2006, Okinawa, Japan, Jan.10 (2006)
4. 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)
5. Genya Ogawa, Katsuyuki Kise, Tsuyoshi Torii and Tomoharu Nagao：Onboard Evolutionary Risk Recognition System for Automobiles - Towards the Risk Map System；IEEE Trans. on Industrial Electronics, Vol.54, No.2, pp.878-886, April 2007, (2007)
6. Yuta Nakano and Tomoharu Nagao: Automatic construction of abnormal signal extraction processing from 3D diffusion weighted image; Proc. of IWAIT-2007, Bangkok, Thailand, Jan.8-9, P5-05 (2007)
7. Jun Ando and Tomoharu Nagao: Fast tree-structural image processing using GPU; Proc. of IWAIT-2007, Bangkok, Thailand, Jan.8-9, P3-25 (2007)
8. Hiroshi Ando and Tomoharu Nagao: Particle Swarm Optimization for template matching; Proc. of IWAIT-2007, Bangkok, Thailand, Jan.8-9, P4-02 (2007)

9. Shinichi Shirakawa and Tomoharu Nagao: Genetic Image Network (GIN): Automatically construction of image processing algorithm; Proc. of IWAIT-2007, Bangkok, Thailand, Jan.8-9, P3-34, pp.643-648 (2007)
10. Shinichi Shirakawa, Shintaro Ogino and Tomoharu Nagao: Graph Structured Program Evolution; Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2007), London, England, Jul.7-11 (2007)

(2) 口頭発表

1. 中野雄太，長尾智晴：3D-ACTIT による 3 次元 PET 画像データからの自動領域抽出；映像情報メディア学会技術報告，IST2005-38，ME2005-96 (2005)
2. 白海英，長尾智晴：単眼カメラを用いた人の動作認識に関する研究；映像情報メディア学会技術報告，IST2005-39，ME2005-97 (2005)
3. 安藤　淳，長尾智晴：並列 ACTIT による画像処理自動構築，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-16 (2005)
4. 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)
5. 小川原也，喜瀬勝之，長尾智晴：遺伝的プログラミングによる動画像処理アルゴリズムの獲得，人工知能学会全国大会，3F4-02 (2005)
6. 安藤　淳，長尾智晴：画像処理自動生成システムＡＣＴＩＴの並列化，平成 17 年電気学会電子・情報・システム部門大会，GS8-4 (2005)
7. 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 (2005)
8. 村　英敏，長尾智晴：PC クラスタを用いた木構造状画像処理の高速化に関する研究，映像情報メディア学会技術報告，Vol.30, No.17, pp.87-88, ME2006-94 (2006)
9. 小川原也，喜瀬勝之，鳥居　毅，長尾智晴：車載用進化的危険認知システム～リスクマップシステムの実現に向けて，(社)自動車技術会 2006 年春季学術講演会 (2006)
10. 藤嶋　航, 長尾智晴:進化計算アルゴリズムにおける構造と数値の同時最適化, FIT2006 第 5 回情報科学技術フォーラム，F-015 (2006)
11. 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，FIT2006 第 5 回情報科学技術フォーラム，F-016 (2006)
12. 中野雄太，長尾智晴：3D-ACTIT を用いた 3 次元拡散強調画像からの異常信号領域自動抽出，FIT2006 第 5 回情報科学技術フォーラム，H-011 (2006)
13. 河西菜美子，長尾智晴，竹林茂生：3D-MR Angiography を用いた脳内血管の解析に関する研究，FIT2006 第 5 回情報科学技術フォーラム，H-012 (2006)
14. 余部治昭，長尾智晴：特徴量選択型 SVM を用いた欠陥画像分類，FIT2006 第 5 回情報科学技術フォーラム，I-033 (2006)
15. 森　喬顯, 安藤　宏, 長尾智晴：フィルタ列自動構築型の画像分類学習法の研究，FIT2006 第 5 回情報科学技術フォーラム，I-034 (2006)
16. 安藤　淳，長尾智晴：GPU を用いた木構造状画像処理の高速化に関する研究，FIT2006 第 5 回情報科学技術フォーラム，J-070 (2006)

17. 乾谷　徹，長尾智晴：車載単眼カメラによる距離計測に関する研究，FIT2006 第5回情報科学技術フォーラム，L-028 (2006)
18. 安藤　宏，長尾智晴：Particle Swarm Optimization に基づく木構造最適化法の提案，情処研報，Vol.2007, No.19, 2007-MPS-63, pp.105-108 (2007)
19. 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，情処研報，Vol.2007, No.19, 2007-MPS-63, pp.93-96 (2007)
20. 中野雄太，長尾智晴：テクスチャ特徴を用いた PET 画像解析，電子情報通信学会総合大会，(2007)
21. 安藤　淳，長尾智晴：複数の GPU を用いた木構造状画像処理の高速化に関する研究，電子情報通信学会総合大会，(2007)
22. 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，(2007)
23. 安藤　宏，長尾智晴：Particle Swarm Optimization for template matching，電子情報通信学会総合大会，(2007)
24. 張　穎，長尾智晴：木構造画像処理の追加学習に関する研究，電子情報通信学会総合大会，(2007)
25. 小川原也，笠置誠祐，泉名克郎，喜瀬勝之，長尾智晴：車載用歩行者抽出動画像処理のオンライン学習による獲得，情報処理学会全国大会，(2007)
26. 小川原也，笠置誠祐，泉名克郎，喜瀬勝之，長尾智晴：進化的危険認知のための車載自律学習システム，(社)自動車技術会 2007 年春季学術講演会，262 (2007)

(3) 出 版 物

1. 長尾智晴：進化的画像処理；「画像ラボ」日本工業出版株式会社；Vol.16, No.9, pp.64-67 (2005)

研究成果による工業所有権の出願・取得状況

計5件

工業所有権の名称	発明者	権利者	工業所有権の種類、番号	出願年月日	取得年月日
進化計算システム及び進化計算方法	長尾智晴，藤嶋　航	横浜国立大学	特願2005-274309	平成17年9月21日	

工業所有権の名称	発明者	権利者	工業所有権の種類、番号	出願年月日	取得年月日
進化的画像分類装置，フィルタ構造生成方法，及びプログラム	長尾智晴，森　番顯	横浜国立大学	特願2006-35085	平成18年2月13日	

工業所有権の名称	発明者	権利者	工業所有権の種類、番号	出願年月日	取得年月日
進化的画像処理装置	長尾智晴	横浜国立大学	特願2006-067329	平成18年3月13日	

工業所有権の名称	発明者	権利者	工業所有権の種類、番号	出願年月日	取得年月日
進化計算システム及び進化計算方法	長尾智晴，白川真一	横浜国立大学	特願2006-186769	平成18年7月6日	

工業所有権の名称	発明者	権利者	工業所有権の種類、番号	出願年月日	取得年月日
進化計算システム及び進化計算方法	長尾智晴，白川真一	横浜国立大学	特願2007-031592	平成19年2月13日	

目次

第１章　序論

１．１　本研究の目的と構成

本稿は平成１７年度から平成１８年度の２年間，基盤研究（B）として遂行された「**複雑な構造に対する新しい進化的計算法の開発とその動画像処理への応用**」の研究成果報告書である．

工学における多くの問題は最適化問題に帰結することができる．このため，現在，多点探索と生物進化の原理に基づく最適化法である進化戦略，遺伝的アルゴリズム(GA)や遺伝的プログラミング(GP)などの**進化的計算法**（Evolutionary Computation）が，その性能・利便性の良さから様々な分野で利用されている．しかしながら，例えばGAは文字列または数値列，GPは木構造の最適化法でしかなく，それらで記述することが難しい内部に複雑な複数の構造をもつ解を求める問題に対しては適用することができない．一方，工学で実際に現れる最適化問題の多くは，それらの構造が複雑に組み合わされた内部構造をもっている場合がほとんどである．そこで本研究では，図1.1に示すような**文字列・数値列・木構造・ネットワーク構造・グラフ構造・メモリなどの構造が内部に混在して存在し，複雑に相互作用するような，従来の進化的計算法では適用不能な対象に対する有効な新しい進化的計算法を開発するとともに，その有効性を実問題に適用して検証する**．

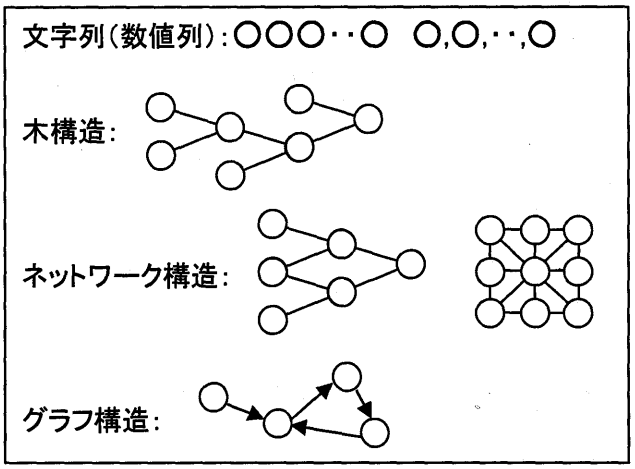


図1.1　様々な構造の例

本研究は，(1)複雑な問題の記述方法の開発，(2)記述された問題の進化的最適化のアルゴリズムの開発，(3)実問題への適用と有効性の検証，の要素から構成される．

GAやGPのような従来の進化的計算法は様々な問題に対して有効であることがこれまでに確認されているが，求める解を０や１の文字列・数値列あるいは木構造として記述しなければならず，実問題に応用する場合には制約が強過ぎることが多い．例えば本研究で適用例として扱う監視システムにおける動画像処理では，動画像からの移動物体の抽出・追跡，時間的に異なる移動物体の識別・認識，移動物体の動作解析・分類など，性質のそれぞれ異なる様々な処理を統合的に組み合わせる必要がある．このため，**従来の進化的計算法は有効ではない**．これらに対して本研究では，従来よりも問題の記述能力が格段に高く，複雑な内部構造全てを含む対象全体を最適化することができる，いわば **“知能情報処理の遺伝子解析”と呼ぶべき性能をもつ新しい進化的計算法の開発を目指す**ものであり，非常に独創的かつ新規性が高い．本手法により，進化的計算法の適用分野並びに自動構築できるアルゴリズム・方法論が飛躍的に拡張され，従来は人間が個々に多大な時間と労力をかけて考案しなければならなかった多くの知能情報処理アルゴリズムの自動構築が初めて可能になり，人工知能の発展に大いに貢献できるものと予想される．産業・社会への波及効果も非常に大きいと考えられる．

進化的計算法については，これまでに国内外の多くの研究者が研究に携わっており，様々な問題への応用，GA・GPなどの個々のアルゴリズムの性能改善，並列化，最適化原理の追及などに対して研究を行っているが，いずれも処理対象を限定し，それぞれ専用の染色体記述を適用している．これらに対して本研究では，**従来の進化的計算法を包含し，かつ，今までは扱うことができなかった構造の最適化を可能にする，より一般化された進化的計算法を目指す**ものである．これまでにまだほとんど研究が行われていないテーマであり，本研究の新規性は明らかである．

本研究では次の３つの新たな進化計算法を提案し，それらを実問題に適用した．

- GMA（Genetic Matrix Algorithm）：数値と構造を同時に最適化する進化計算法．
- GIN（Genetic Image Network）：ネットワーク構造を最適化する進化計算法．
- GRAPE（GRAph structured Program Evolution）：グラフ構造を最適化することで自動プログラミングを可能にする進化計算法．

これらはいずれも複雑な構造を最適化するもので，本研究の目的を達成するものである．本報告書ではこれらの提案する進化計算法の原理と応用例について述べる．

１．２　本報告書の構成

本報告書は全５章から構成されている．

第１章：本章であり，本研究の目的と構成，本報告書の構成について述べている．

第２章：提案手法１であるGMAについて述べている．

第３章：提案手法２であるGINについて述べている．

第４章：提案手法３であるGRAPEについて述べている．

第５章：結論であり，本研究を総括している．

第２章　提案手法１：ＧＭＡ

２．１　従来手法の問題点

進化計算法は現在，様々な観点から研究が進められている．本研究では特に構造と数値の同時最適化の問題に着目する．進化計算法の分野において，構造と数値の同時最適化が可能な新しい進化計算アルゴリズムが実現できれば，工学，産業，社会の様々な分野で有効に利用できる画期的な方法になると考えられる．

従来の手法にはいくつかの問題点が存在する．Hybrid GP/GA や GA-P では，個体に適用する遺伝操作オペレータを適切に設計することが困難である．これは，最適化する個体中に木構造と数値列が並存しているため，木構造用のオペレータは木構造にしか適用できず，数値列用のものは数値列にしか適用できないことが原因である．この２つのオペレータを交互に適用しているだけでは，結果として双方（構造と数値）の最適化が協調できず，中途半端な個体しか構築することができない．また STROGANOFF では，各ノードでの計算量を低次元に抑えることができるものの，ノードでの局所探索は多重回帰分析だけに依存しており，探索が不十分になる可能性がある．

GE では，文法を記述する BNF を遺伝子型と表現型との変換に用いることによって，比較的自由な記述と設計が可能となった．しかし，遺伝子型での小さな変化が表現型に及ぼす影響が均質でなく，時には１ビットの変化が構造を大きく変えてしまう可能性があり，進化計算上好ましくない．

ACTIT は，従来の画像処理自動構築システムの問題点を解消すべく提案されたが，いくつかの問題が残っている．その問題のひとつに，画像処理フィルタ内で参照する数値パラメータ（しきい値）の設定がある．事前に用意したフィルタの中には，数値パラメータを処理の際にしきい値として必要とするものがいくつか存在する．従来の ACTIT ではパラメータ値を固定，もしくは入力画像から算出できるような算術式を設定し半自動化していた．しかし，より汎用的かつ効果的な画像変換を構築するためには，この数値パラメータを調整する必要がある．

そこで本章では，構造と数値の同時最適化を可能にする新しい進化計算アルゴリズムであるGMA（Genetic Matrix Algorithm ）を提案する．また，その応用として画像処理の自動構築への応用例を示す．

2．2　GMA (Genetic Matrix Algorithm)

2．2．1　GMAの概要

提案手法であるGMA（Genetic Matrix Algorithm; GMA ）の概要について述べる．

本手法では，構造と数値の同時最適化を目標とし，その応用として画像処理の自動構築システムとしての実装を目指した．構造と数値の同時最適化とは具体的に，構造と数値を遺伝子のレベルで融合させることである．最適化の概略はGA，GP と同じく，個体集団に対して各種オペレータを作用させ世代交代を繰り返す形態をとった．

ここで，構造と数値の同時最適化を実現するために本手法に実装された要素の中で，特徴的な点である表現型と遺伝子型について，また，両者間の変換方式について述べる．

2．2．2　GMAの概要

本手法では最終的に構築する表現型としては木構造を想定しており，それら木構造のノードに数値パラメータが内包されている．数値パラメータをノードに内包した木構造の例を図2.1 に示す．

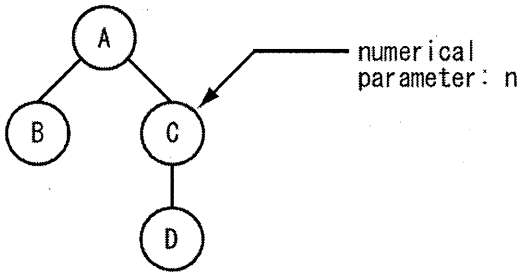


図2.1　GMAの表現型の例

この表現型では，GP と同様に，ノードにはオペレータ類を設定することができる．特に本手法では数値パラメータをそれぞれのノードに内包することができる．この数値を参照することによって，様々な処理が可能になる．数式を表現した場合の応用例を次に示す．

- **ノードでの処理の係数，パラメータを微調整する**
  例：$x^n \rightarrow x^2, x^3, \cdots$
- **ノードの種類を変更する**
  例：$f(x) \rightarrow \sin(x), \cos(x), \tan(x), \cdots$

2．2．3　遺伝子型と遺伝操作オペレータ

本手法では遺伝情報を表現する形態（遺伝子型）として，0 と1 から構成される2 次の行列を採用した．また，どの個体も同じ大きさ（行，列）をとる（homogeneous）ものとした．この形態を採用した理由を次に示す．

- **数値最適化に有利**

  0 と1 を用いて遺伝情報を構成することでGA に似た最適化が想定可能となる（本手法の着眼点のひとつである数値最適化を考慮した）
- **構造の表現を可能にする**

  1 次元（直列）の構成では，遺伝子型から表現型（木構造）への変換が困難であり，また，遺伝操作における遺伝形質の選択に偏りが発生する可能性がある（遺伝子型から表現型への変換については後述する）

ところで，このような遺伝子型が保持する遺伝情報には，最適化過程において遺伝操作オペレータが作用する．本手法では非常に単純な交叉と突然変異を実装した．ここでその詳細について述べる．

交叉

本手法では交叉の操作として，遺伝情報を表現する2 次行列から小領域を選択し，その小領域を交換する方法をとった．この操作は，GA での一点交叉の拡張である．GA では1 次元の記号列に対して1ヶ所の交叉点を選択していたものを，2 次行列中の2 点によって小領域を決定する方法に変更した．交換する小領域のサイズは2 個体間で共通にする必要があるが，位置に制限は無い（2 次行列の「左上」と「右下」の交換も可能）．操作の適用例を**図2.2** に示す．

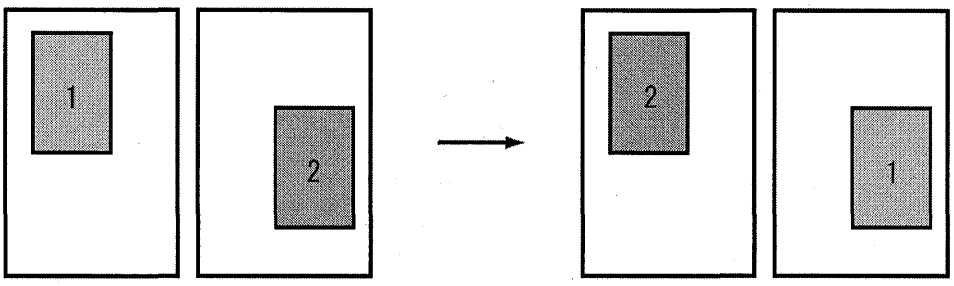


図2.2　GMAでの交叉の適用例

突然変異

本手法では突然変異の操作として，2 次行列をラスタ走査し，各ビットにおいて一定確率でビット反転させる方法をとった．この操作は，交叉と同じくGA での一様突然変異の拡張である．ラスタ走査によって，2 次行列全体に対して突然変異操作を適用する．操作の適用例を**図2.3** に示す．

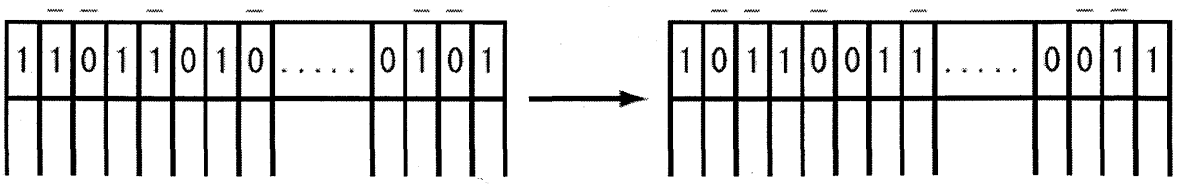


図2.3　GMAでの突然変異の適用例

2．2．4　Genotype-to-Phenotype Mapping

本手法で用いた表現型と遺伝子型の設定は，それぞれ2.2.2，2.2.3で述べたとおりである．本節では，本手法において最も肝要である，表現型と遺伝子型の対応付け（Genotype-to-Phenotype Mapping;GPM）[18]について述べる．

本手法における表現型である数値パラメータを含む木構造は，2次行列の遺伝子型を上位からラスタ走査することによって得られる．その際，表現型は根（root）から葉（leaf）へ向かう方向に遺伝子型の2次行列は，単ユニットであるレコード（行ベクトル）の集合となっている．レコードは表現型における1ノード（終端記号1つ，または，数値パラメータを含む非終端記号1つ）に相当し，非終端記号の場合には子ノードに関する情報も含まれる．レコードの概略を図2.4に示す．

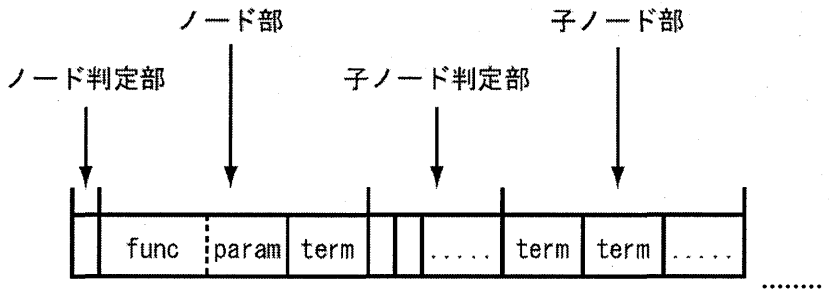


図2.4　GMAの遺伝子型コード

ここで，各部の詳細について述べる（括弧内は遺伝子長）．

- **ノード判定部（1）**
  認識中のノードを終端記号とするか非終端記号とするかを判定する部分（ノードの実体は後のノード部にある）
- **ノード部（非終端記号長＋ 数値パラメータ長＋ 終端記号長）**
  ノードの実体を保持する部分（非終端記号の場合，自身の記号（id）と数値パラメータ．終端記号の場合は，自身の記号だけ）
- **子ノード判定部（非終端記号の取りうる引数の最大数）**
  子ノードとして終端記号をとるか非終端記号をとるかを判定する部分（認識中のノードが非終端記号であった場合にだけ参照される）
- **子ノード部（引数の最大数× 終端記号長）**
  子ノードとなる終端記号の実体を保持する部分（子ノードとして終端記号をとると判定された場合にだけ参照される）

認識中のノードが非終端記号であった場合には，子ノード判定部に従って子ノードの認識に進む．その際，終端記号であれば現在のレコードにある子ノード部から認識する．しかし，子ノードとして非終端記号をとる場合には，次レコードに認識が進む．ただし，行列の最下部に到達した際に，さらに非終端記号をとると判断された場合にはレコードが不足してしまう状態が発生する．そのような場合には強制的に終端記号を採用することとし，致死遺伝子が発生しないよう考慮した．また学習の際には，1 個体が保持するレコード数（行列の行数）をあらかじめ決定する必要がある．

以上のような手順で遺伝子型から表現型への変換が行われる．この変換の具体例を図2.5 に示す．それぞれの非終端記号が必要とする子ノードの数は別途定義されているものとする．

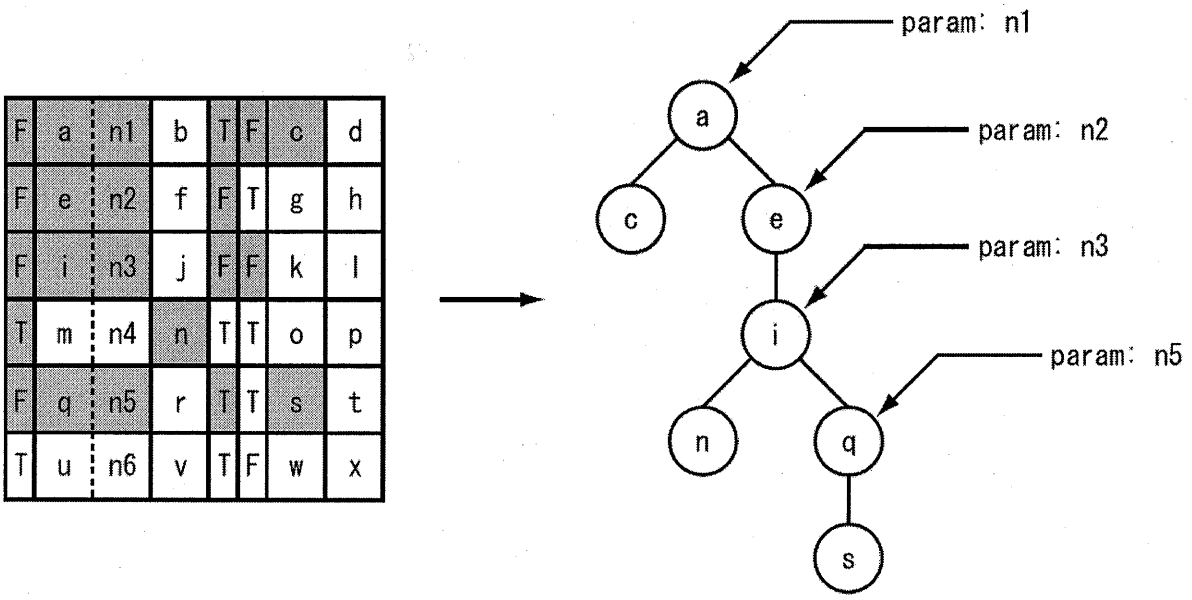


図2.5　GMAのGenotype-to-Phonotype Mappingの例

2．2．5　期待される優位性

これまでに，表現型（2.2.2）と遺伝子型（2.2.3），またそれらを対応させるGenotype-to-Phenotype Mapping（2.2.4）について述べた．ここでは，これらを実装した本手法に期待される優位性について述べる．

- **構造と数値の同時最適化**

  2.2.1 で述べたように，本手法は構造と数値の同時最適化に着目し様々な要素を実装した．同時最適化が可能になることでより効率的な最適化が可能になり，また，最適化手法の応用範囲がさらに広がり，社会の様々な分野への応用が可能になると考えられる

- **コンパクトな表現型**

  構造と同時に数値も最適化を進めることによって，同じ性能を発揮する表現型であっても，サイズの小さいコンパクトな表現型が構築可能になると考えられる．なぜなら，ある機能を表現するために，これまでは小さな処理を何回か繰り返し適用することによって表現していたが，本手法では小さな処理自身が内包する数値パラメータを調整することにより，数ステップで表現可能になると考えられるからである

- **遺伝情報の保持**

  単純なGA やGP では，個体中の遺伝情報がすべて表現型に置き換わる．これは遺伝情報を効率的に扱っているように見えるが，個体集団内で遺伝操作を行いながら進化する場合，表現型には影響を与えない部分の遺伝情報が進化に重要な役割を果たしているとされる．この「表現型に影響を与えない部分」をイントロン（intron）と呼び，「表現型の情報を保持する部分」エクソン（exon）と呼ぶ．本手法では，一定サイズの遺伝子型を使用してサ

イズの異なる表現型を構成しているため，サイズの小さな表現型の場合には使用する遺伝情報が少なく，イントロンを保持することができる．これにより本手法はより多くの遺伝情報を次世代に伝達することが可能になると考えられる

最後に，本手法に実装されたいくつかの要素について述べる．

- **世代交代モデル**

  世代交代モデルとしてMGG を採用した．

- **遺伝子のコーディング**

  効率的な遺伝形質の保持のため，遺伝子型のコーディングにはグレイコードを用いた

- **擬似乱数の生成**

  高速に良質な擬似乱数を生成する方法としてMersenne Twister[27] を使用した

2．3　画像処理の自動構築への応用

本手法は基礎的なアルゴリズムであり，様々な最適化問題への応用が考えられる．最終的に構築される構造が数値パラメータを含む木構造であることから，GP が適用できる問題に加え，有限状態オートマトンや確率過程への応用も可能であると考えられる．ここでその応用の一例として，画像処理の自動構築への応用について述べる．

本手法を用いた画像処理の自動構築として， ACTIT に似た，画像変換の自動構築システムを提案する．具体的には，処理対象である原画像と結果となる目標画像が入力されたとき，あらかじめ用意された画像処理フィルタを用いて，画像変換の過程を自動的に構築するシステムを提案する．提案するシステムの基本的な趣旨はACTIT とほぼ同様であるが，いくつかの相違点が存在する．提案手法とACTIT を比較して，いくつかの共通点と相違点についてその詳細を次に述べる．

- **共通点**

  ➢ **最適化の評価関数**

  個体の評価（解の精度）である適応度は，個体が保持する画像処理フィルタによる処理結果画像と目標画像との差分で表される．具体的な数値としては，式(2.1)を用いて算出される

$$fitness = \frac{1}{K}\sum_{k=1}^{K}\left\{1 - \frac{\sum_{i=1}^{W}\sum_{j=1}^{H} w_k^{ij}|o_k^{ij} - t_k^{ij}|}{V_{max}\sum_{i=1}^{W}\sum_{j=1}^{H} w_k^{ij}}\right\} \tag{2.1}$$

  K ： 教師画像セット数
  W ： 入力画像の幅方向の画素数
  H ： 入力画像の高さ方向の画素数
  o_k ： 原画像画素値
  t_k ： 目標画像画素値
  w_k ： 重み画像画素値（特定の画素の重要度を決める画像）
  V_{max} ： 最大画素値

  ➢ **システムの入力と出力**

  入力は原画像と処理後の目標画像であり，出力は画像変換を表現した木構造状の画像処理フ

ィルタである

- **相違点**
  - **最適化アルゴリズム（GPとGMA）**

    ACTIT ではGP を用いて最適化を進めるのに対し，提案手法ではGMA を用いて構造（フィルタの個数，種類，並び）と数値パラメータの同時最適化を進める
  - **使用する画像処理フィルタ**

    木構造中のノードとなる個々の画像処理フィルタとして，一般的によく知られた基礎的なアルゴリズムを利用する点は両手法に共通している．しかし，ACTITでは個々のフィルタは固定された操作に限定されるのに対し，提案手法では数値パラメータを参照することで操作の内容を調整することができる（提案手法とACTITに実装されている画像処理フィルタの一覧を表2.1～2.4 に示す）

表2.1　ACTITに実装されている１入力１出力フィルタ

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	ソーベルフィルタ
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）以上の画素を白にする
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 Pass	F	低周波数成分カット
Low Pass	f	高周波数成分カット
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）

表2.2　ACTITに実装されている２入力１出力フィルタ

フィルタ名	記号	処理内容
２入力１出力フィルタ（f_1：入力画像 1，f_2：入力画像 2）		
Logical Sum	L	論理和（$\max(f_1, f_2)$）
Logical Prod	l	論理積（$\min(f_1, f_2)$）
Algebraic Sum	A	代数和（$f_1 + f_2 - (f_1 \times f_2 \div V_{max})$）
Algebraic Prod	a	代数積（$f_1 \times f_2 \div V_{max}$）
Bounded Sum	B	限界和（$f_1 + f_2$）
Bounded Prod	b	限界積（$f_1 + f_2 - V_{max}$）
Drastic Sum	u	激烈和（$f_1 = 0 \rightarrow f_2$，$f_2 = 0 \rightarrow f_1$，$f_1, f_2 \neq 0 \rightarrow V_{max}$）
Drastic Prod	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（$\mathrm{abs}(f_1, f_2)$）

表2.3　提案手法に実装されている１入力１出力フィルタ

P：数値パラメータを使用するフィルタ

フィルタ名	記号	処理内容	
１入力１出力フィルタ			
Mean	1-	平均値フィルタ	
Max	1M	最大値フィルタ	
Min	1m	最小値フィルタ	
Sobel	1d	ゾーベルフィルタ	
Light Edge	1E	白エッジ強調	
Dark Edge	1e	黒エッジ強調	
Light Pixel	1T	しきい値以下の画素を黒にする（$V_{max} \times [0.0, 1.0]$）	P
Dark Pixel	1t	しきい値以上の画素を白にする（$V_{max} \times [0.0, 1.0]$）	P
Light Pixel Auto	aT	しきい値（平均階調値）以下の画素を黒にする	
Dark Pixel Auto	at	しきい値（平均階調値）以上の画素を白にする	
Large Area	1S	面積しきい値よりも小さい領域を白にする （面積しきい値＝面積平均値 $\times [0.8, 1.2]$）	P
Small Area	1s	面積しきい値よりも大きい領域を白にする （面積しきい値＝面積平均値 $\times [0.8, 1.2]$）	P
Large Area Auto	aS	分割領域面積平均値よりも小さい領域を白にする	
Small Area Auto	as	分割領域面積平均値よりも大きい領域を白にする	
Inversion	1i	反転フィルタ	
Deploy	1K	分散フィルタ	
Gamma	1G	ガンマ補正フィルタ（$\gamma = [0.2, 2.2]$）	P
Gamma Auto	aG	ガンマ補正フィルタ（$\gamma = 2$）	
Contraction	1x	収縮フィルタ	
Expansion	1X	膨張フィルタ	
Prewitt	1z	Prewitt フィルタ	
Laplacian	1g	ラプラシアンフィルタ	

表2.4　提案手法に実装されている２入力１出力フィルタ

フィルタ名	記号	処理内容	
High Area Per Box	1P	外接矩形に対する充填率が高い孤立領域を残す （充填率しきい値 $= [0.0, 1.0]$）	P
Low Area Per Box	1p	外接矩形に対する充填率が低い孤立領域を残す （充填率しきい値 $= [0.0, 1.0]$）	P
High Area Per Box Auto	aP	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 Auto	ap	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	1R	外接矩形の縦横比が 1.0 に近い孤立領域を残す （縦横比 $= 1.0 \pm [0.0, 0.3]$）	P
Rectangular Box	1r	外接矩形の縦横比が 1.0 に近い孤立領域を消す （縦横比 $= 1.0 \pm [0.0, 0.3]$）	P
Square Box Auto	aR	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 Auto	ar	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	1C	外接矩形に対する孤立領域面積の真円度が 1.0 に近い孤立領域を残す （真円度 $= 1.0 \pm [0.0, 0.2]$）	P
Low Circularity	1c	外接矩形に対する孤立領域面積の真円度が 1.0 に近い孤立領域を消す （真円度 $= 1.0 \pm [0.0, 0.2]$）	P
High Circularity Auto	aC	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 Auto	ac	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	1H	線形変換フィルタ	
Binarization	1N	２値化フィルタ（$V_{max} \times [0.0, 1.0]$）	P
Binarization Discriminant Analysis	1n	２値化フィルタ（判別分析法）	
Binarization Auto	aN	２値化フィルタ（平均階調値）	
２入力１出力フィルタ（f_1：入力画像 1，f_2：入力画像 2）			
Logical Sum	2L	論理和（$\max(f_1, f_2)$）	
Logical Prod	2l	論理積（$\min(f_1, f_2)$）	
Algebraic Sum	2A	代数和（$f_1 + f_2 - (f_1 \times f_2 \div V_{max})$）	
Algebraic Prod	2a	代数積（$f_1 \times f_2 \div V_{max}$）	
Bounded Sum	2B	限界和（$f_1 + f_2$）	
Bounded Prod	2b	限界積（$f_1 + f_2 - V_{max}$）	
Drastic Sum	2u	激烈和（$f_1 = 0 \rightarrow f_2$，$f_2 = 0 \rightarrow f_1$，$f_1, f_2 \neq 0 \rightarrow V_{max}$）	
Drastic Prod	2U	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	2D	差分フィルタ（$\mathrm{abs}(f_1, f_2)$）	

２．４　実験結果とその考察

本章では，提案手法であるGMA を用いて構築した画像処理自動構築システムの実験結果について述べる．また性能比較実験として，従来手法であるACTITによる実験結果について述べ，両者を比較する．

２．４．１　実験結果

ここでは，3 種類の画像を用いて行った実験の詳細と結果について述べる．学習時のパラメー

タは全ての実験において共通の設定を使用した．この設定を表2.5，2.6 に示す．使用したパラメータは，どのような画像に対してもある程度の性能が期待できると考えられる一般的な数値を採用した．なお本節で示す実験結果の画像は，同条件で行った3 回の実験のうちの1 例である．

表2.5　パラメータ設定（GMA，提案手法）

最適化世代数	20000	個体数	100
レコード数	100	MGG の子個体数	50
交叉率	1.0	突然変異率	0.05
数値パラメータ長	3bit	引数の最大数	2

表2.6　パラメータ設定（ACTIT，従来手法）

最適化世代数	20000
個体数	100
木構造中に含まれるノード数の最小数，期待値，最大数	1，15，40
MGG の子個体数	50
トーナメントサイズ	2
交叉率	1.0
突然変異率	0.05
引数の最大数	2

２．４．１．１　細胞壁の抽出

学習に使用した画像は，図2.6に示す人間の視細胞画像であり，画像処理の目的は細胞壁の抽出とした．画像サイズは64×64pixel である．この画像処理は，原画像中に濃淡の差（偏り）があり，一度に全体を2値化するといった単純な処理では完了できない複雑な問題である．学習では，2例の画像処理を同時に満足する画像処理フィルタの構築を目指す．

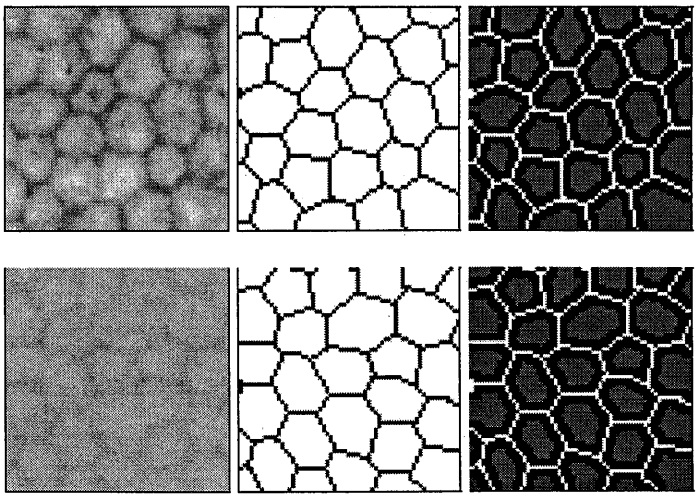


図2.6　教師画像（左から，原画像，目標画像，重み画像）

提案手法（GMA）

学習結果として得られた出力画像を**図2.7**に示す．**図2.6**の目標画像と比較すると，細胞の内包物がノイズとして残っていたり，細胞壁自体が消失していたりするものの，ある程度の抽出ができている．

従来手法（ACTIT）

学習結果として得られた出力画像を**図2.8**に示す．細胞の内包物の残留はみられないものの，細胞壁の消失部分が多く不十分である．ところで，**図2.7**では細胞壁がいくらか太く抽出され，**図2.8**では細く抽出されているが，これは重み画像において細胞壁の膨張を黙認する設定となっているためである．

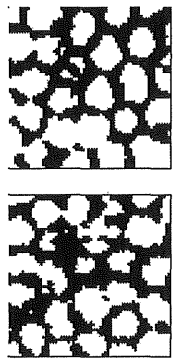


図2.7　出力画像（提案手法）

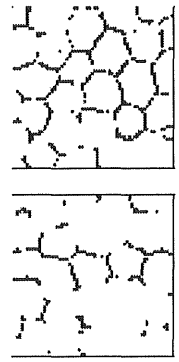


図2.8　出力画像（従来手法）

２．４．１．２　肺領域の抽出

学習に使用した画像は，**図2.9**に示す肺のレントゲン画像であり，画像処理の目的は肺領域の抽出とした．画像サイズは128×128pixelである．この画像処理は，画素階調値を厳密に調整する作業が必要となる処理であり，専門家の試行錯誤による設計であっても困難である．また原画像自体が，撮影条件の違い，人物の違いなどから均質な画像が得られにくく，複数枚の画像変換を同時に達成するとなれば，非常に困難な処理となる．

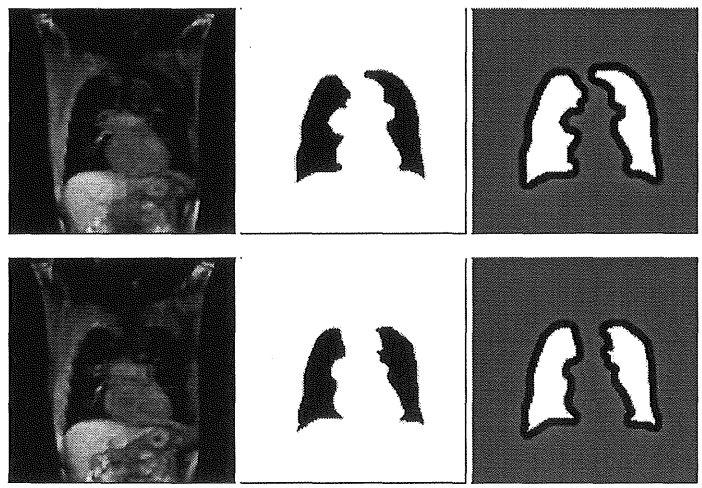


図2.9　教師画像（左から，原画像，目標画像，重み画像）

提案手法（GMA）

学習結果として得られた出力画像を**図2.10** に示す．出力画像の一方は肺の片方が完全に消失しており，処理が不完全である．しかしもう一方の出力画像は，気管支の部分に残留物があるものの良好な処理結果であるといえる．

従来手法（ACTIT）

学習結果として得られた出力画像を**図2.11** に示す．残留物が少なく良好な検出結果といえるが，肺の外周が丸みを帯びており抽出が中途半端である．

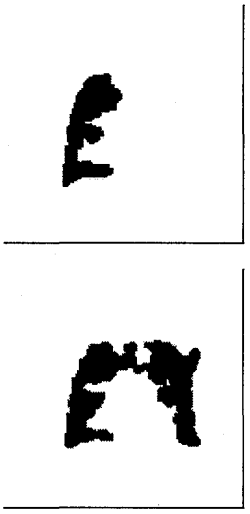


図2.10　出力画像（提案手法）

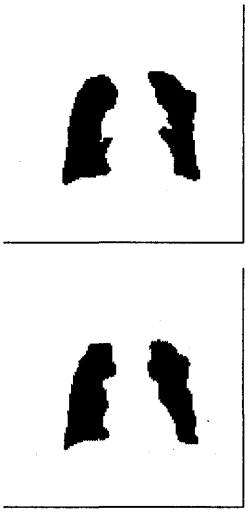


図2.11　出力画像（従来手法）

２．４．１．３　神経細胞本体の抽出

学習に使用した画像は，**図2.12**に示す神経細胞画像であり，画像処理の目的は細胞本体の抽出とした．画像サイズは658×517pixel である．この画像処理は，単純な2 値化だけで達成できるように見えるが，神経細胞本体から伸びる軸策部分の除去や，階調値にむらのある本体の抽出等の問題があり困難である．なお，この実験では最適化世代数を10000とした．

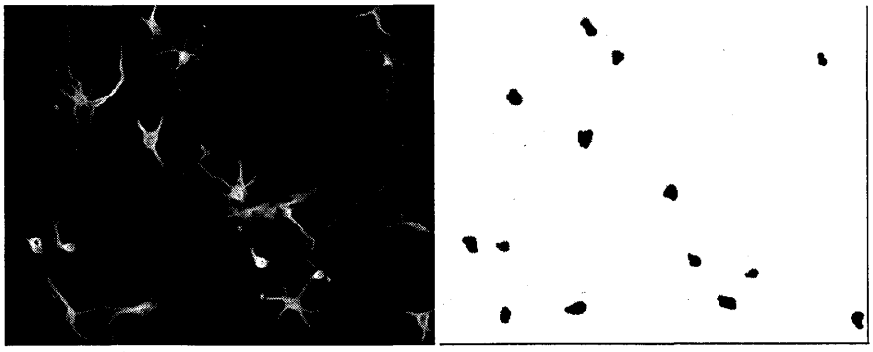


図2.12　教師画像（左から，原画像，目標画像）

提案手法（GMA）

学習結果として得られた出力画像を**図2.13**に示す．軸策が多少残留しているが，目標画像で指定した神経細胞は抽出されており，良好な結果である．

従来手法（ACTIT）

学習結果として得られた出力画像を**図2.14**に示す．残留物が少なく良好な結果である．

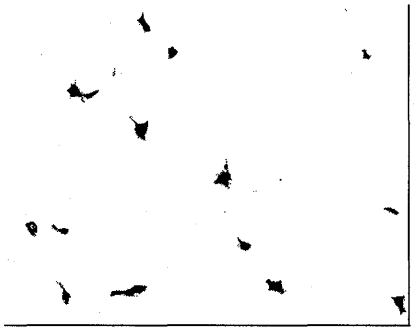


図2.13　出力画像（提案手法）

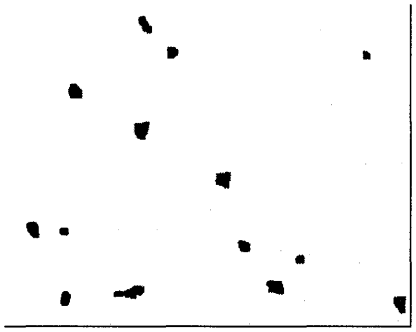


図2.14　出力画像（従来手法）

２．４．２　考察

ここでは，提案手法と従来手法の性能比較について述べる．

２．４．２．１　比較検討

画像処理自動構築システムとして提案された2 手法の性能を比較する場合，様々な要素が考えられる．ここでは，次の要素を比較検討要素として選出した．

- **学習に対する評価値（適応度）**
- **未知画像に対する汎化性**
- **学習の成功率**
- **生成される木のサイズ**
- **生成される木の統一性**
- **学習速度**

個々の要素の詳細について順に述べる．

学習に対する評価値（適応度）

学習（最適化）が終了した時点で精度のよい（評価値の高い）解が得られているか，また，評価値が学習過程でどのように推移しているかを検討する．まず，最適化の終了時点での評価値を検討する．実験結果を**図2.15**，**表2.7**に示す．なおこの実験結果は，同条件で行った3 回の実験の平均値である．

表2.7　終了時の評価値（平均値）

	GMA	ACTIT
細胞壁	0.924	0.898
肺領域	0.884	0.985
神経細胞	0.992	0.997

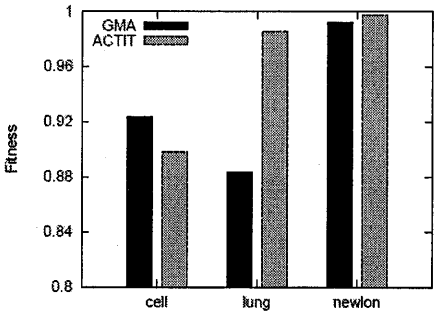


図2.15　最適化終了時点での評価値

両手法とも，90% あるいはそれを上回る100% に近い評価値を獲得している（表2.7）．これは，最適化によって有効な画像処理が構築されたことを意味しており，GP を使用した従来手法（ACTIT）だけでなくGMA を利用した提案手法が，画像処理の自動構築法として有効であることを示している．両手法の評価値を比較すると，提案手法はACTIT に多少劣る場合もあるが同等程度の性能を発揮していることが分かる．ところで，肺領域抽出の実験では提案手法が良好な結果であるのに対し，提案手法では失敗している．従来手法が良好な結果である原因として，ノードとなる画像処理フィルタで使用されるしきい値の半自動決定方法が教師として使用した肺画像について有効に作用しているものと考えられる．この特徴は，明るさ調整フィルタや孤立領域に関連するフィルタ等にみられる．

次に，学習過程における評価値の推移について検証する．実験結果を図2.16，2.17，2.18に示す．細胞壁抽出と肺領域抽出の実験結果に着目すると，ACTIT は初期段階で高い評価値の個体を発見するとそのまま収束しているが，GMA を用いた提案手法では評価値が比較的低い場合でも，後半に評価値が上昇している．これは，探索の後半になっても探索能力が衰えていないことを示している．

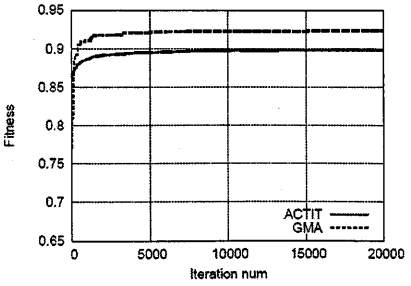


図2.16　評価値の推移(細胞壁)

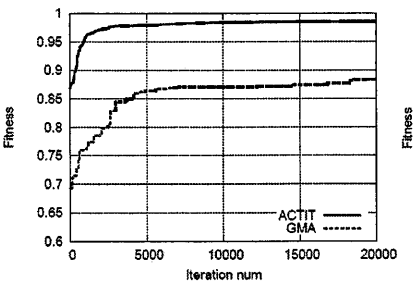


図2.17　評価値の推移(肺領域)

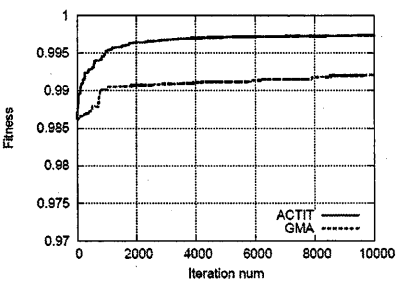


図2.18　評価値の推移(神経細胞)

未知画像に対する汎化性

学習の結果得られた画像処理フィルタの性能は評価値として定量的に表現されるが，未知画像に対する汎化性についても別途検討する必要がある．ここでいう未知画像とは学習に用いた画像に類似した画像のことをいい，学習時には使用せず汎化性の検証だけに用いる．

まず，細胞壁抽出の問題について検証する．未知画像とそれに対する処理結果を図2.19，2.20，2.21に示す．未知画像は3 種類あり，それらのサイズは64×64pixel と128×128pixel である．未知画像は教師画像に比べて画像中の濃淡の差が大きく，困難な処理であると考えられる．提案手法では抽出対象である細胞壁が太く，また細胞の内包物が残留し細胞の大半が潰れてしまっている．また，従来手法では細胞壁が細線で抽出されているものの消失部分が多く，処理が不完全である．

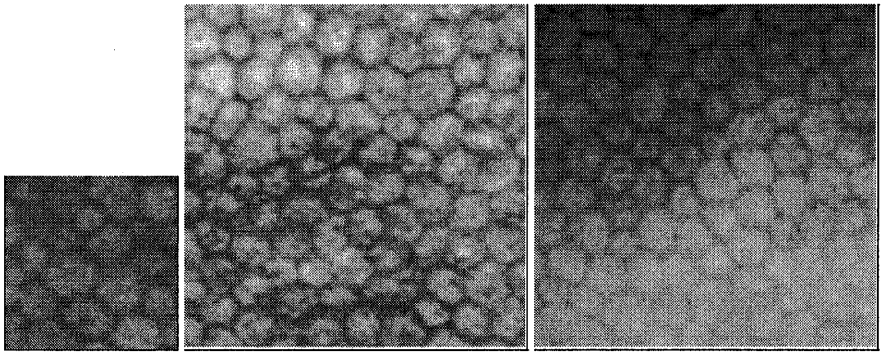


図2.19　未知画像（細胞壁）

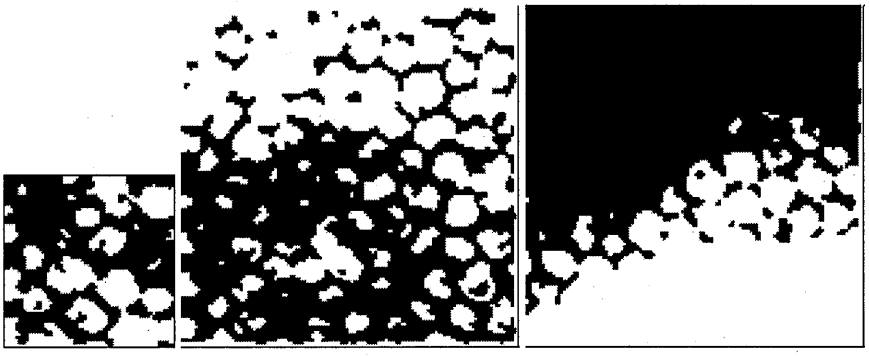


図2.20　未知画像に対する処理結果（提案手法，細胞壁）

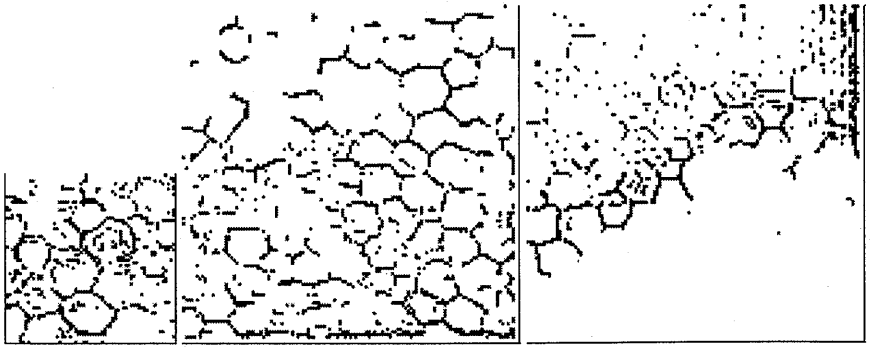


図2.21　未知画像に対する処理結果（従来手法，細胞壁）

次に，肺領域抽出の問題について検証する．未知画像とそれに対する処理結果を図2.22，2.23，2.24に示す．未知画像は1 種類あり，そのサイズは128×128pixel である．未知画像は教師画像に酷似しており，教師画像によって構築された画像処理フィルタはある程度有効に作用することが予想される．

処理結果によると両手法とも，いくらかの欠損はあるものの肺領域の抽出ができていることが分かる．しかし提案手法では肺内部や下端での欠損が多く，処理が不完全である．従来手法では良好な抽出結果となっており，この問題では提案手法の優位性が薄れている．この原因としては学習に対する評価値の検討でも述べたように，処理対象である肺画像に対してフィルタ処理で参照されるしきい値の半自動設定が特に有効に作用した点にある．

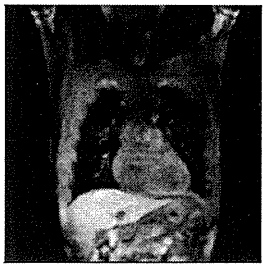


図2.22　未知画像（肺領域）

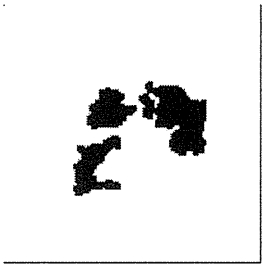


図2.23　未知画像に対する処理結果（提案手法，肺領域）

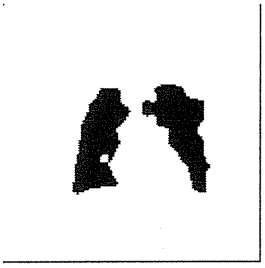


図2.24　未知画像に対する処理結果（従来手法，肺領域）

最後に，神経細胞抽出の問題について検証する．未知画像とそれに対する処理結果を図2.25，2.26，2.27に示す．未知画像は2種類あり，そのサイズは658×517pixel である．未知画像は教師画像に比べいくらか軸策部分が多いものの，類似した画像といえる．

未知画像の1 例目に対する処理結果では，従来手法と提案手法との間に大きな差があることが分かる．従来手法では神経細胞を大きく抽出した上に，軸策を巻き込んで抽出しており処理に失敗している，それに対し提案手法では，いくつかの神経細胞を捕えていないものの軸策の抽出が

なく，抽出に成功した神経細胞の形がはっきりしている．2 例目の処理結果は，両手法ともある程度良好な処理結果であるといえる．しかし，それぞれに問題点が存在する．従来手法ではどの抽出領域も多少大きく，抽出した神経細胞の形がはっきりしていない．一方，提案手法では抽出した神経細胞の形がはっきりとしているが，微小領域の残存があり，また，抽出できていない神経細胞があり，完全な処理とはいえない．

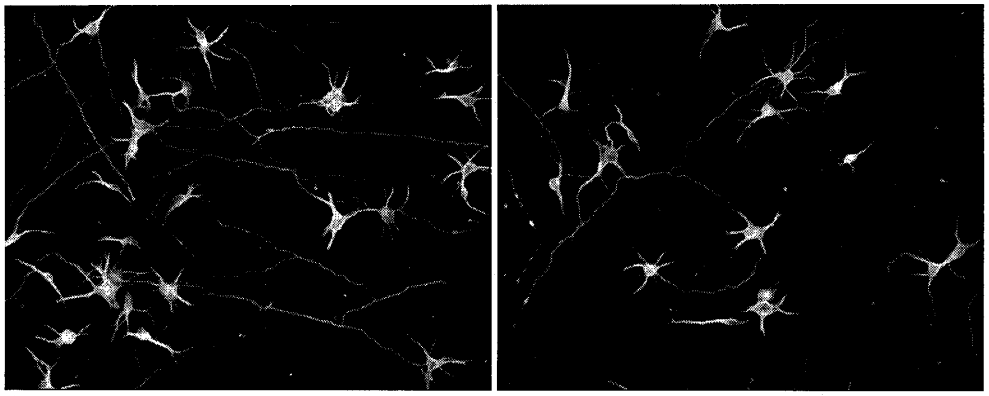


図2.25　未知画像（神経細胞）

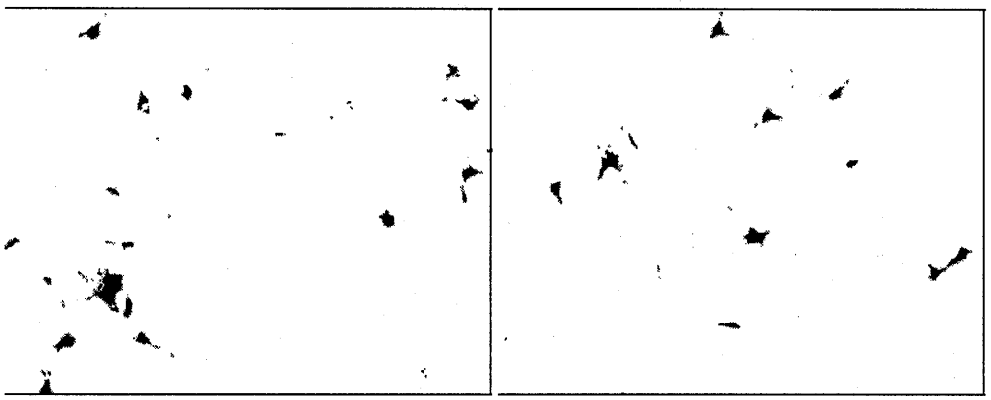


図2.26　未知画像に対する処理結果（提案手法，神経細胞）

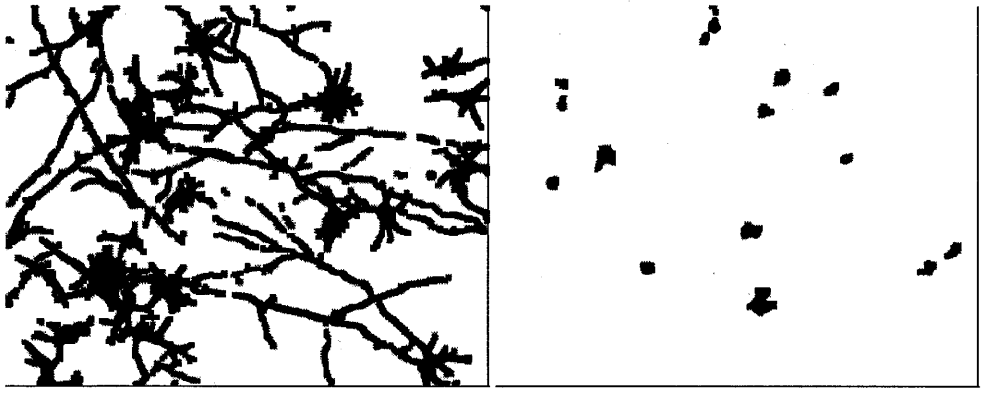


図2.27　未知画像に対する処理結果（従来手法，神経細胞）

学習の成功率

学習のアルゴリズムとしてランダム性のある確率的な手法を用いているため，必ずしも最良（もしくはそれに近い）解が得られるとは限らない．その成功率が低い場合には試行錯誤的な要因が強く，進化計算法として望ましくない．

今回行った3種類の実験（各3回）の評価値の詳細を**表2.8**に示す．

表2.8　終了時の評価値（全試行）

		GMA	ACTIT
細胞壁	1	0.924	0.899
	2	0.923	0.896
	3	0.924	0.899
肺領域	1	0.893	0.986
	2	0.873	0.984
	3	0.885	0.985
神経細胞	1	0.992	0.997
	2	0.993	0.997
	3	0.991	0.997

両手法とも，すべての試行において90% から100% に近い評価値を獲得している．統計数としては少ない試行ではあるが，全試行で良好な個体を構築していることから，両手法の学習の成功率は高いといえる．

生成される木の統一性

同条件で複数回学習を行い，同程度の評価値の画像処理フィルタを得た場合に，それらの画像処理フィルタの統一性について検討する．学習の後に得られる個体（解）に統一性が無い場合には，個体を分解し共通要素を選出して主幹を形成する必要がある．これは確率的な手法によって得られる解の精度に関する問題であり，学習の成功率と同じく非常に重要な要素である．

まず，今回の実験で構築された木構造状の画像処理フィルタをそれぞれ，

- **細胞壁（図2.28，2.29）**
- **肺領域（図2.30，2.31）**
- **神経細胞（図2.32，2.33）**

に示す．

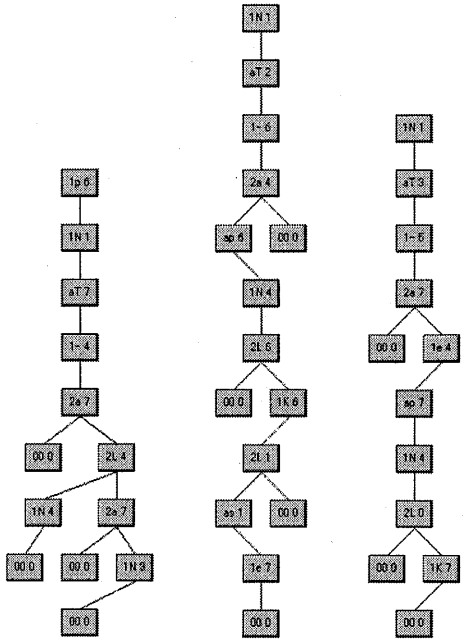


図2.28　構築された画像処理フィルタ（提案手法，細胞壁）

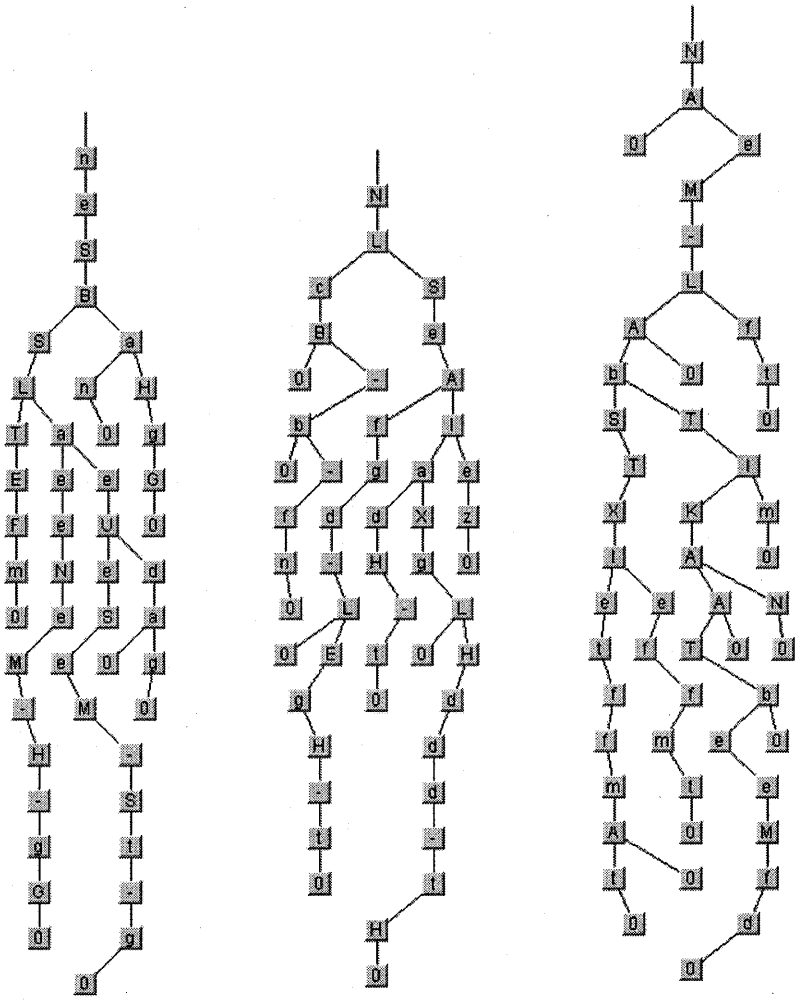


図2.29　構築された画像処理フィルタ（従来手法，細胞壁）

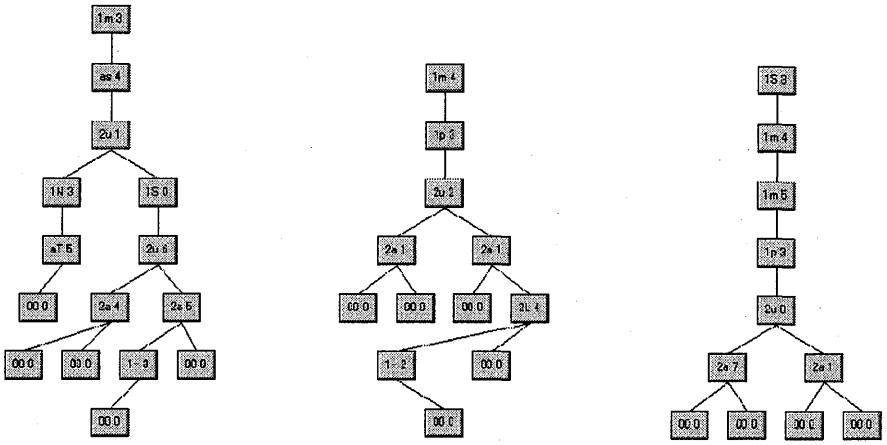


図2.30　構築された画像処理フィルタ（提案手法，肺領域）

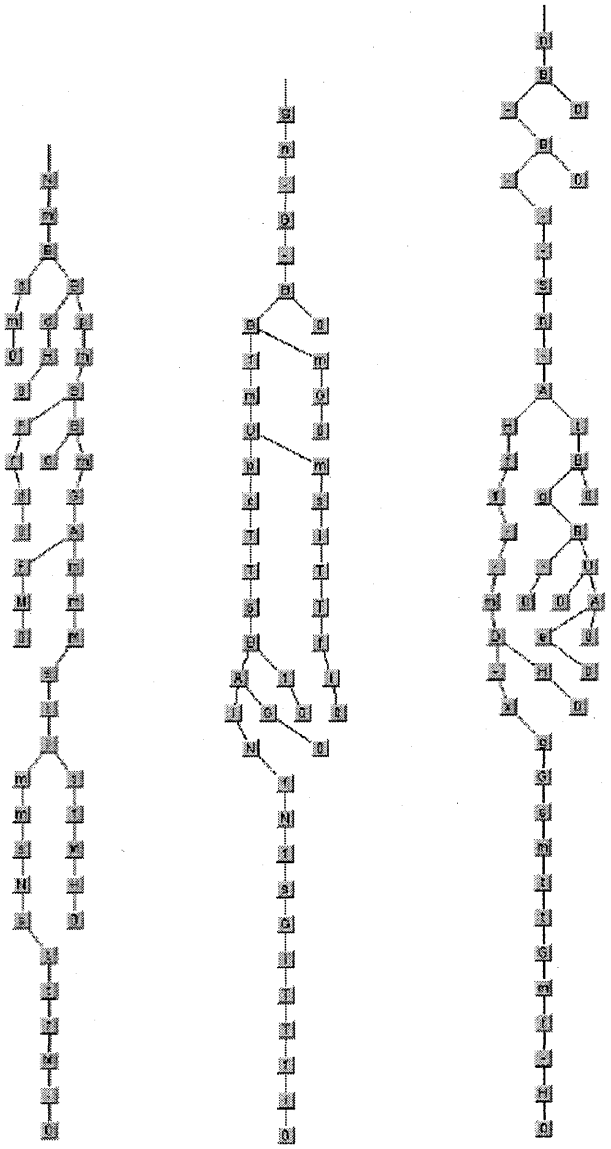


図2.31　構築された画像処理フィルタ（従来手法，肺領域）

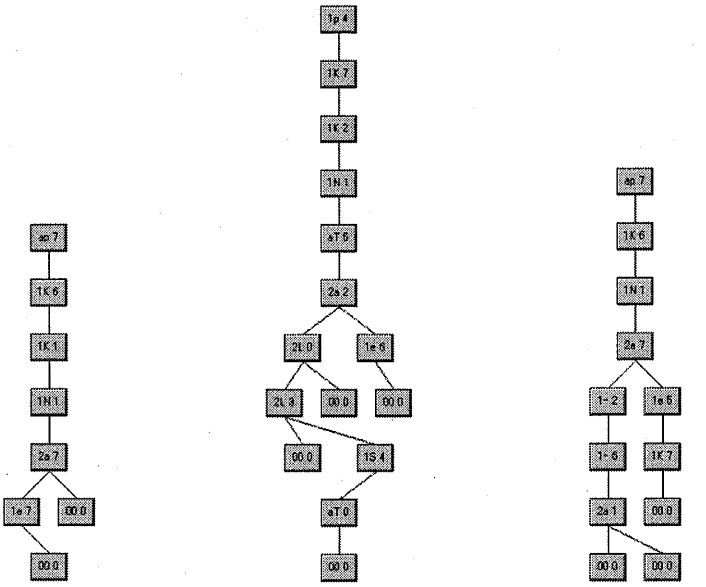


図2.32　構築された画像処理フィルタ（提案手法，神経細胞）

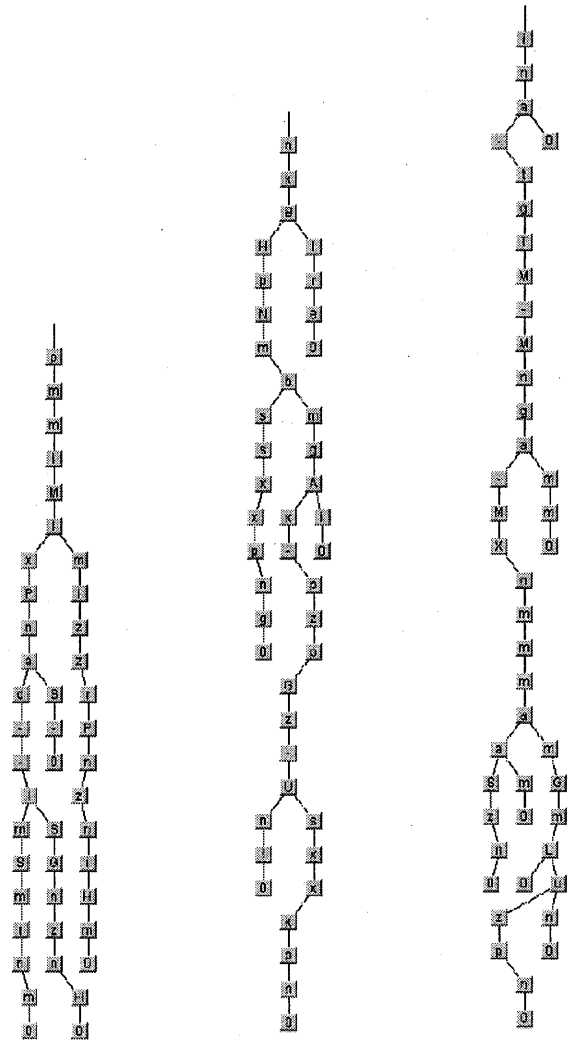


図2.33　構築された画像処理フィルタ（従来手法，神経細胞）

実験によって構築された画像処理フィルタは，それぞれに特徴のある木構造として表現された．細胞壁抽出の実験結果（**図2.28，2.29**）では，木の中段付近でいくつかの分岐があるが，全体としては直線的な構造である．また，肺領域抽出の実験結果（**図2.30，2.31**）では，木の葉に近い部分で2 分岐があり，末端が広い構造である．そして，神経細胞抽出の実験結果（**図2.32，2.33**）では，1 入力1 出力の画像処理フィルタが多用され，直線的なフィルタ構造となっている．

しかし従来手法によって構築された木構造は，全体的な形状には統一性がみられるものの，形状があまり安定していない．同条件で実験を行ったにも関らず，画像処理の最後である上端付近での処理が直線的であったり分岐形状を成していたり，細部での統一がみられない．木構造の上端は画像処理の最終調整部分であり，画像処理を考える上で非常に重要な部分である．それに対し提案手法によって構築された木構造は，全体的な形状の統一性に加え，上端付近の構造，ノードで使用している画像処理フィルタの種類にも統一性がみられる．

生成される木のサイズ

構築された木構造状の画像処理フィルタは，可能な限り簡素な構成であることが望ましい．簡素な画像処理フィルタであれば画像処理実行時間の短縮だけでなく，画像処理をチップ化しハードウェアに実装する場合にも有利である．

実験によって構築された画像処理フィルタのノード数を**図2.34** に示し，これらを比較する．なおこの実験結果は，同条件で行った3回の実験の平均値である．

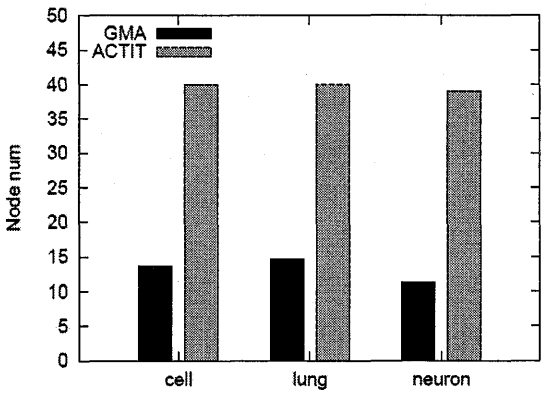


図2.34　ノード数の比較

従来手法ではノード数が40に達するまで木が成長している．これは，設定パラメータである「木構造中に含まれるノード数の最大数」と同一であり，実際にはノード数が40を超えていたことが考えられる（ACTITでは強制的に遺伝操作オペレータのやり直しを行い，ノード数が最大数を超過する事態を防いでいる）．それに対し提案手法による実験結果ではノード数が低く抑えられており，シンプルな画像処理アルゴリズムが構築されていることが分かる．教師画像に対する処理を満たしつつ，未知画像に対する汎用性も保持する画像処理アルゴリズムを構築できる点は，提案手法の大きな優位性のひとつである．

学習速度

進化計算法の一種として，学習に必要とされる計算時間は重要な検討要素である．近年の計算機の性能向上に伴い，大型の計算機を用いた最適化も可能であるが，画像処理は非常に広範な分野で必要とされており，小型の計算機（PC等）での活用を考慮すべきである．

学習に要する時間を計測するために，これまでに示した実験結果とは別に実験を行った．使用した画像は細胞壁抽出で使用した画像のうちのの1例のみを使用した．パラメータ設定は**表2.5, 2.6**と同一であるが，最適化世代数を10000とした．5回の実験結果の平均値を**表2.9** に示す．

表2.9　学習に要する時間

GMA			ACTIT		
real	user	sys	real	user	sys
4m49.717s	4m47.461	0m2.159	71m5.052	71m1.764	0m2.526

- **計算機スペック**

  DELL Dimension8250（CPU:Pentium4 2.40GHz，Memory:1GB）

- **コンパイラとコンパイルオプション**

  gcc version 3.3.1

  g++ *.cpp -lm -lgcc -O3

提案手法は従来手法に比べて大幅に計算時間が短縮されている．原因としては次の2点が考えられる．

- **ノード数の減少**

従来手法，提案手法ともに，計算のボトルネックは画像処理の部分である．提案手法ではノード数の減少（細胞壁抽出の実験では1/3程度）によって，この画像処理の計算量が大幅に減少したと考えられる

- **遺伝操作オペレータの速度**

従来手法ではGPを利用して木構造を直接扱っていたが，提案手法では木構造を扱う場面が限定され，遺伝操作オペレータは2次行列に作用している．また，提案手法では木が無制限に成長することがないため，遺伝操作オペレータのやり直しによる計算量の増大を防ぐことができる．これらも計算量の減少の要因であると考えられる

2．4．2．2　まとめ

従来手法と提案手法の比較として，様々な要素について検証を行った．これら検証結果の総括を表2.10 に示す．提案手法は，従来手法（ACTIT）と同程度の評価値，汎用性をもつ画像処理フィルタを，より小さなサイズの木構造として，より短時間に構築できることが確認できた．

表2.10　要素比較

	GMA	ACTIT
学習に対する評価値	○	◎
未知画像に対する汎用性	○	◎
学習の成功率	○	○
生成される木のサイズ	○	×
生成される木の統一性	◎	△
学習速度	◎	×

2．5　本章のまとめ

2．5．1　GMA から得られた結果

本章では，構造と数値の同時最適化アルゴリズムであるGMA を提案した．また，GMA を画像処理の自動構築に適用し，構築された画像処理フィルタについての検証を行った．さらに，画像処理自動構築法の従来手法であるACTIT との性能比較を行った．

実験の結果，GMA を用いて有効な画像処理の自動構築が可能であることが確認された．特に従来手法であるACTIT と比較した場合，より単純なフィルタで同程度の性能をもつ画像処理フィルタをより高速に構築できることが確認された．

2．5．2　GMA に関する今後の課題

本研究を発展させるためには，さらに考慮すべき要因がいくつか存在する．それらを次に示す．

- **最適化手法としての性能評価**

本章では提案手法の性能評価として画像処理自動構築の問題を適用したが，より一般的な手法として論じるには最適化手法としてのベンチマーク問題が必要となる．関数同定等の問題が適当であると考えられる

- **解空間の解析**

構造と数値を同時に最適化可能になったことで，解候補が散在する解空間の状態（fitness landscape ）に注意する必要がある．特に，構造と数値がそれぞれどのように解空間中で作用する要因となるかを考慮する必要があると考えられる．これには遺伝操作オペレータの設計に関る問題であると考えられる

第３章　提案手法２：ＧＩＮ

３．１　はじめに

本章では，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組み上げることで，目的の画像変換を自動構築する手法である Genetic Image Network (GIN)の提案を行う．これまでに，遺伝的プログラミング（GP）を用いて木構造状に画像処理フィルタを自動構築する ACTIT システムが提案されており，画像変換の自動構築に関して有効性が示されている．ACTIT は木構造を扱うのに対して，GIN ではネットワーク構造を扱うため木構造を含めた表現が可能であり，構造的に表現能力が高いと考えられる．そのため，GIN では ACTIT で表現することができないフィードバックや処理画像の再利用，複数出力といった構造を表現することができるようになる．提案手法を画像変換の自動構築に適用し，ACTIT では表現することができない構造の獲得実験を行い，提案手法の有効性を示す．

遺伝的アルゴリズム（Genetic Algorithm; GA）や遺伝的プログラミング（Genetic Programming;GP）1),2)に代表される進化計算(Evolutionary Computation;EC)は現在，様々な最適化問題に適用され，優れた成果を挙げている．GA を用いてコンピュータプログラムを自動的に構築する GP では，一般的にプログラムの表現方法として木構造が用いられ，遺伝操作では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．

これまでにプログラムの表現形式として木構造ではなく，ネットワーク構造を扱うことでプログラムの自動生成を行う手法も研究されている．Parallel Al-gorithm Discovery and Orchestration (PADO)3)-5)はネットワーク構造であり，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．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のプログラムの実行は，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．他のネットワーク構造を扱う手法としてParallel Distributed Genetic Programming (PDGP)6)やCartesian Genetic Programming (CGP)7),8)が挙げられる．これらの手法では，フィードフォワード型などに制限されたネットワーク構造を扱う．近年ではネットワーク構造をプログラムの表現形式として，自律エージェントの構造決定を行う研究が盛んである．Genetic Network Programming (GNP)9),10)や遺伝的オートマトンGAUGE11)などが代表的な例である．ネットワーク構造を採用することで，ノードの再利用や自律エージェントの過去の情報を利用することが可能になる．これらの手法のようにプログラムの表現形式としてネットワーク構造を扱うメリットとして，木構造に比べて複雑な表現が可能であることや

ネットワーク上での時系列情報の保持が可能であることが挙げられる．

一方，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，IMPRESS12),13)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14)．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15)~19)．画像処理フィルタの組合せを木構造として，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 を用いて自動構築を行うAutomatic Construction of Tree-structural Image Transformation (ACTIT)16),17)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．

木構造を用いる場合，葉ノードから画像を入力し，各ノード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．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，有効性が示されている18),19)．しかし，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，ネットワーク構造は木構造よりも高い表現能力をもっているといえる．そこで本章では，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，Genetic Image Network (GIN)を提案する．

本章では，3.2で提案手法であるGenetic Image Network (GIN) について述べる．3.3では提案手法を画像変換の自動構築に適用し従来手法との比較，解析を行う．

3．2　GIN (Genetic Image Network)

3．2．1　GINの構造

本章で提案するGenetic Image Network (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．本章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．

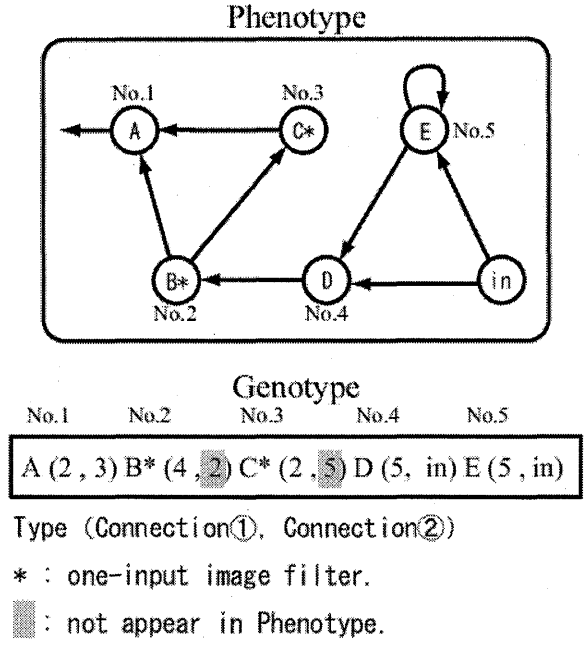


図3.1 Genetic Image Network(GIN)の構造例

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．本章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可能である．本章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．

3．2．2 GIN における遺伝操作

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．本章では，遺伝操作として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Pcによってマスクパターンを生成し交叉点を決定する．GINの交叉の例を**図3.2**に示す．網掛けの部分が確率Pc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．

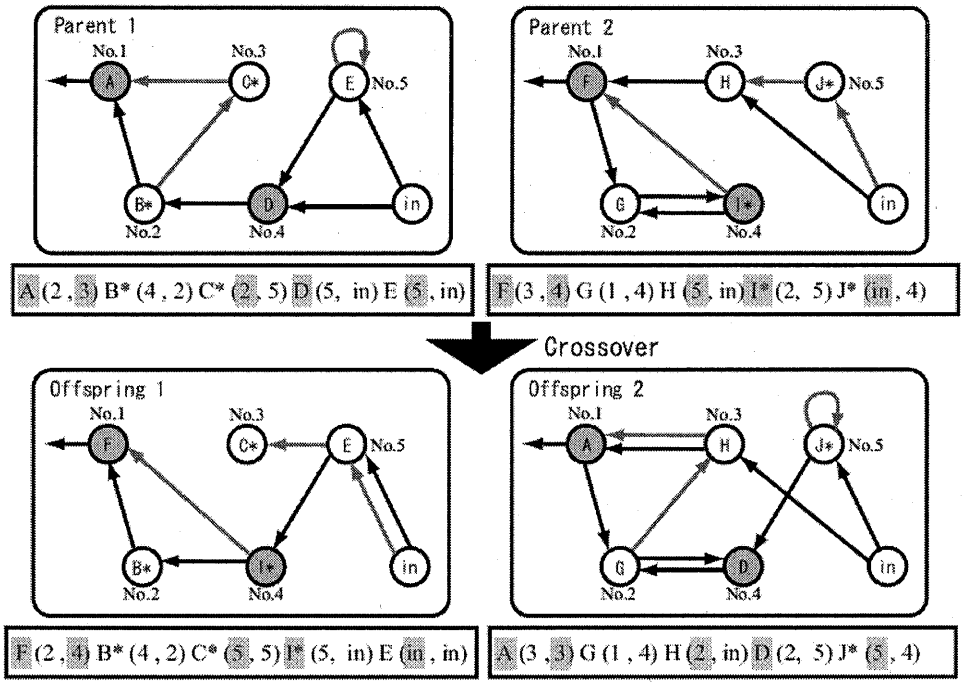


図3.2 Genetic Image Network(GIN)における交叉の例

- **突然変異** ：突然変異は突然変異率*Pm*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の突然変異の例を図3.3に示す．網掛けの部分が突然変異率*Pm*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．

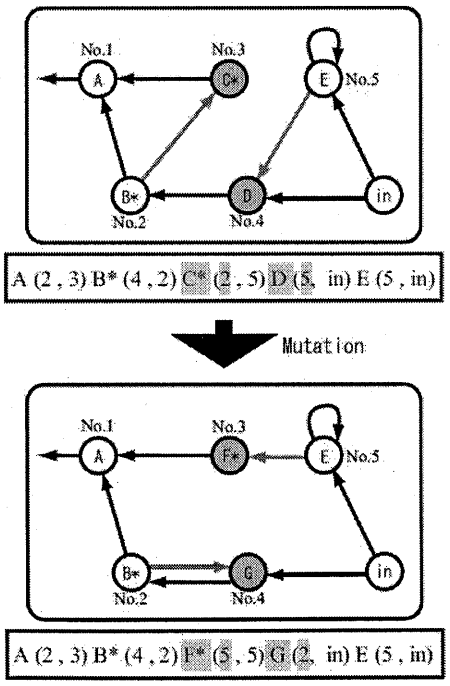


図3.3 Genetic Image Network(GIN)における突然変異の例

3．2．3　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．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．ここで，ブロートとはGPにおいて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である．

3．3　画像変換の自動構築への適用

3．3．1　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表3.2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．

表3.1　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子個体数	50
交叉率（P_c）	0.9
突然変異率（P_m）	0.03
ノード数	20
ステップ数	5,10,15,20

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

ここで，o^n_{ij}は出力画像の画素値，t^n_{ij}は目標画像の画素値，w^n_{ij}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（$w_{ij} = 1$）．

表3.2　実験に用いた画像処理フィルタ

記号	処理内容
1 入力 1 出力フィルタ	
-	平均値フィルタ
M	最大値フィルタ
m	最小値フィルタ
d	ゾーベルフィルタ
E	白エッジ強調
e	黒エッジ強調
T	しきい値（平均階調値）以下の画素を黒にする
t	しきい値（平均階調値）以下の画素を白にする
S	分割領域面積平均値よりも小さい領域を白にする
s	分割領域面積平均値よりも大きい領域を白にする
i	反転フィルタ
K	分散フィルタ
G	ガンマ補正フィルタ（$\gamma = 2$）
x	収縮フィルタ
X	膨張フィルタ
z	Prewitt フィルタ
g	ラプラシアンフィルタ
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
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
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
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
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
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
H	線形変換フィルタ
N	2 値化フィルタ（平均階調値）
n	2 値化フィルタ（判別分析法）
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）	
L	論理和（$\max(f_1, f_2)$）
l	論理積（$\min(f_1, f_2)$）
A	代数和（$f_1 + f_2 - (f_1 \times f_2 \div V_{max})$）
a	代数積（$f_1 \times f_2 \div V_{max}$）
B	限界和（$f_1 + f_2$）
b	限界積（$f_1 + f_2 - V_{max}$）
u	激烈和 （$f_1 = 0 \rightarrow f_2$，$f_2 = 0 \rightarrow f_1$，$f_1, f_2 \neq 0 \rightarrow V_{max}$）
U	激烈積 （$f_1 = V_{max} \rightarrow f_2$，$f_2 = V_{max} \rightarrow f_1$， $f_1, f_2 \neq V_{max} \rightarrow 0$）
D	差分フィルタ（$\mathrm{abs}(f_1, f_2)$）
nop1	f_1 を出力
nop2	f_2 を出力

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．

3．3．2　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った．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回ずつ行った．画素数は128 × 128pixel，$V_{max} = 255$ である．

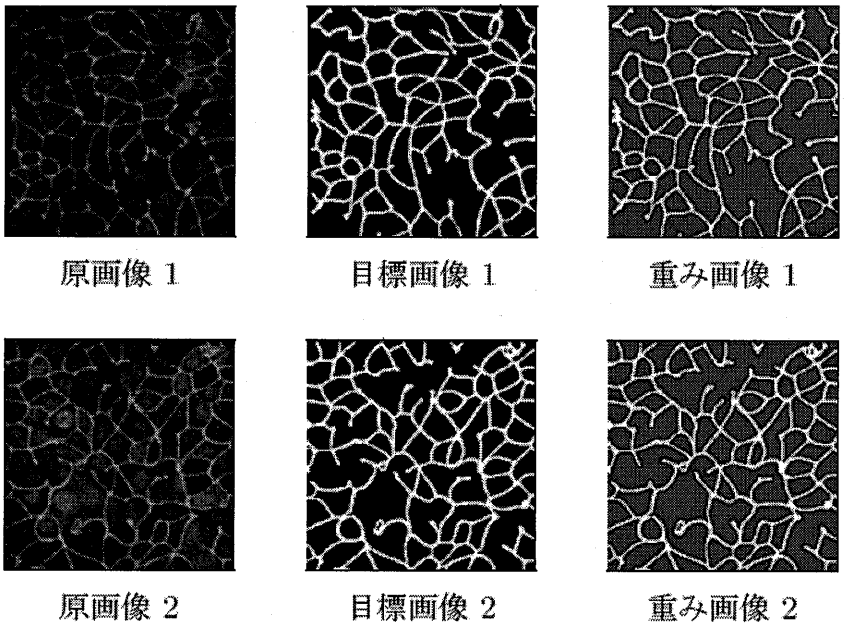


図3.4　実験で用いた教師画像セット（細胞画像）

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．

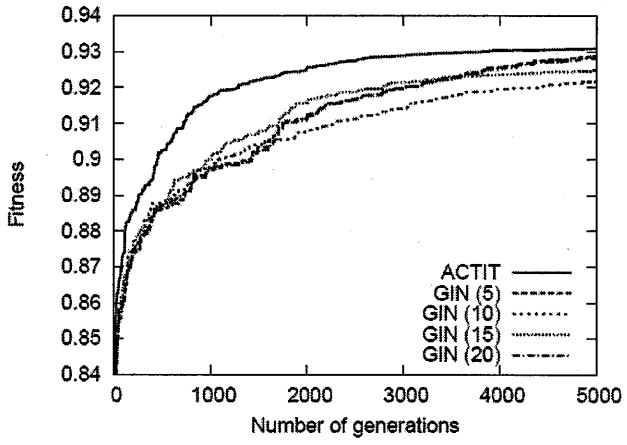


図3.5　適応度の推移（細胞画像）

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．

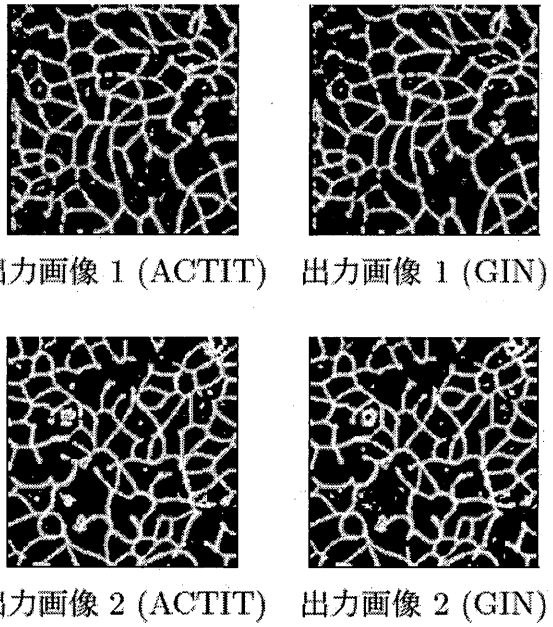
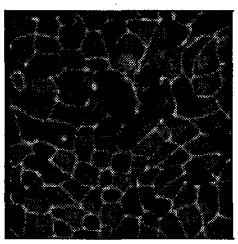


図3.6　ACTITとGINによる出力画像

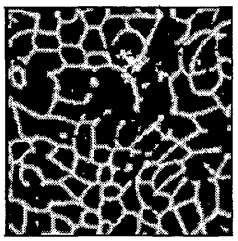
次に実験によって得られた木構造状あるいはネットワーク構造状の画像処理フィルタを，教師画像に類似した未知画像に対して適用した結果を検証する．未知画像とそれに対するACTITとGINの処理結果を図3.7に示す．ACTIT，GINともに良好な画像変換が行われていることがわかる．つまり，両手法とも細胞壁の抽出処理という画像変換を自動的に構築することができたといえる．

3．3．3　複数出力画像変換の自動構築

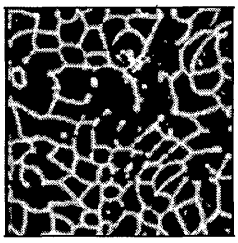
ここでは提案手法であるGINを用いて複数出力画像変換の自動構築を行う．GINではネットワーク構造を表現形式としているため複数出力の表現が可能である．複数出力表現は木構造を扱うACTITでは表現することはできないため，GINの大きな利点の1つである．本実験で用いた教師画像セットを図3.8に示す．画素数は64 × 64pixel，$Vmax = 255$である．画像処理の目的は，手書き文字と印刷文字からなる画像から，“手書き文字除去” という処理と“印刷文字除去”という2つの処理を1つのネットワークで同時に自動獲得することである．この画像では文字除去の部分が比較的明瞭であるため，重み画像は用いなかった．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



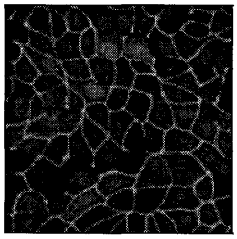
未知画像 1



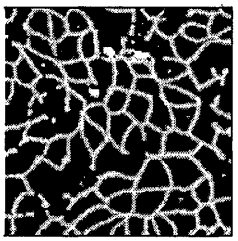
出力画像 1 (ACTIT)



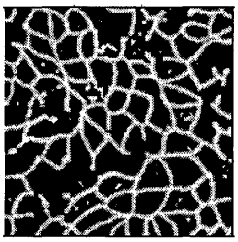
出力画像 1 (GIN)



未知画像 2

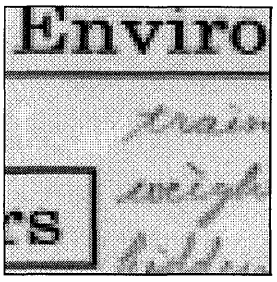


出力画像 2 (ACTIT)

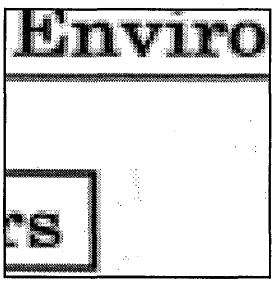


出力画像 2 (GIN)

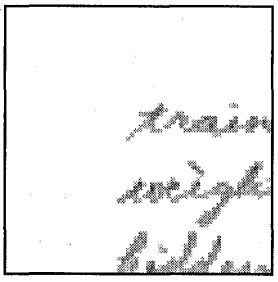
図3.7　未知画像に対するACTIT とGIN の出力画像（細胞画像）



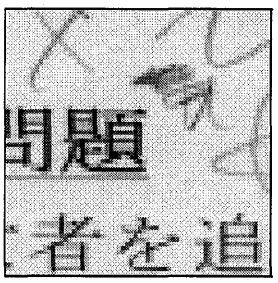
原画像 1



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　実験で用いた教師画像セット（文字除去）

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.

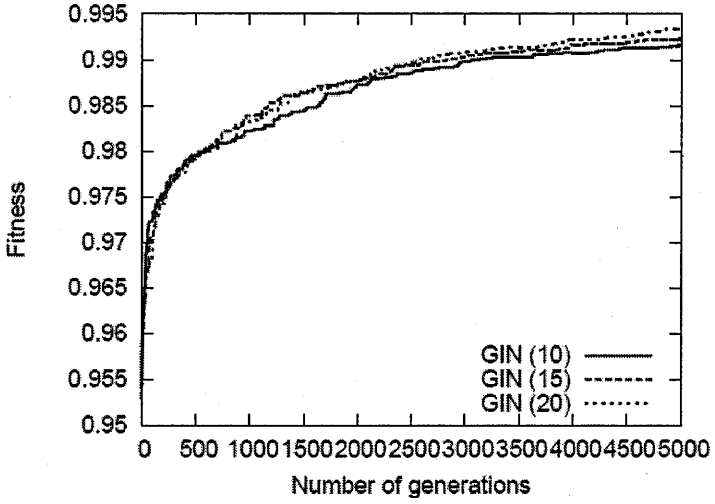
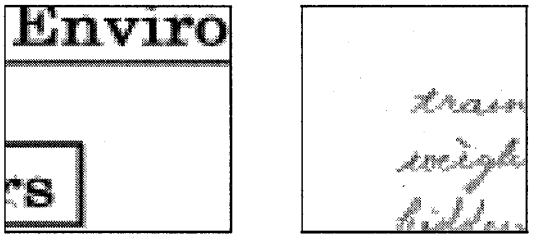
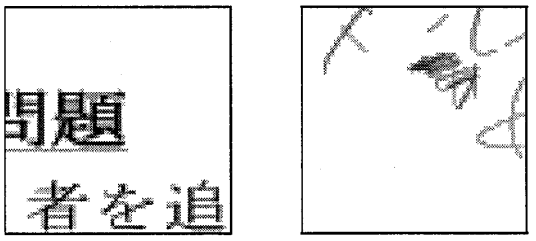


図3.9　適応度の推移（文字除去）



出力画像 1 (a)　　出力画像 1 (b)

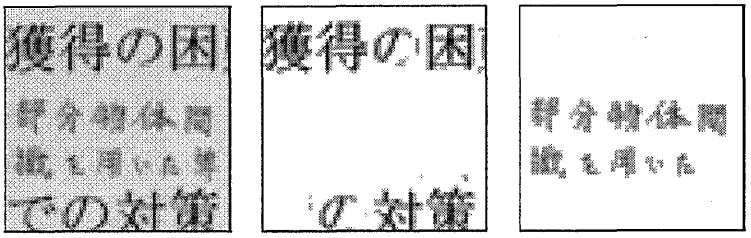


出力画像 2 (a)　　出力画像 2 (b)

図3.10　適応度の推移（文字除去）

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.



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（文字除去）

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．各ノードの記号は表3.2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.

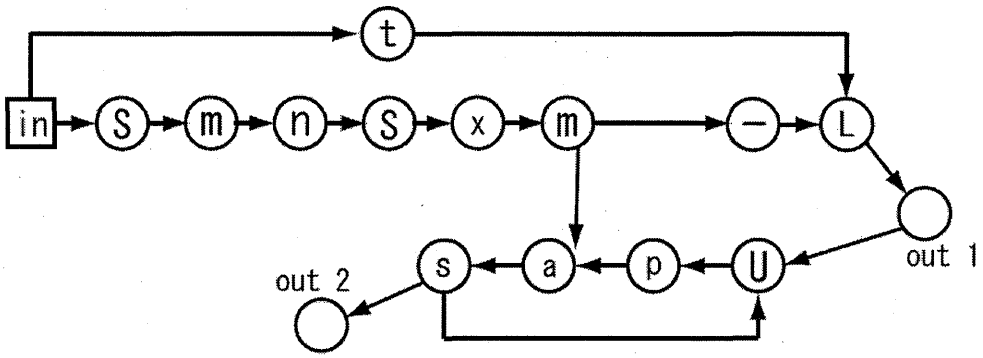


図3.12　GIN によって自動的に構築された構造

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.

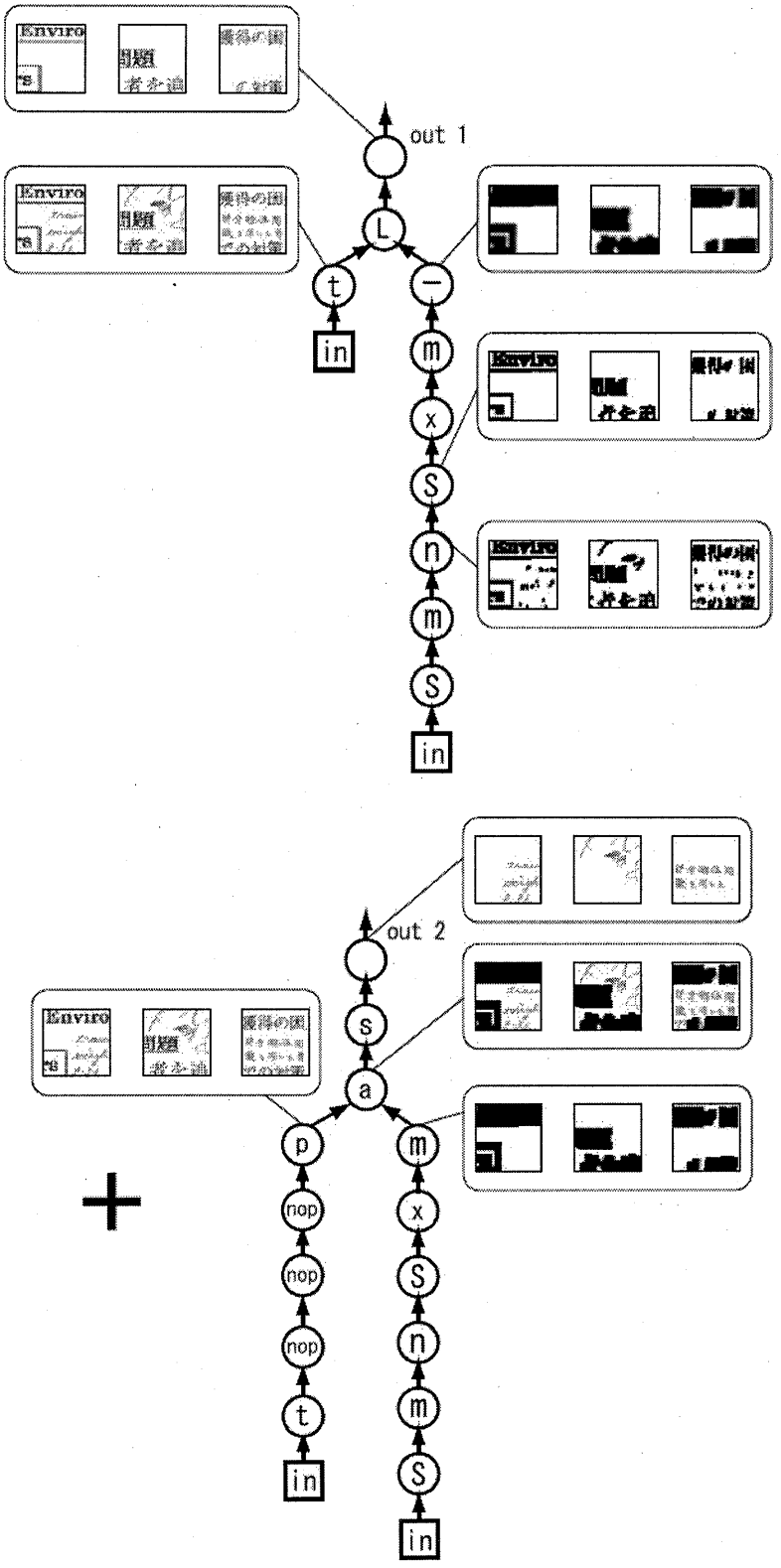


図3.13　図3.12の木構造による表現

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．

3．4　本章のまとめ

本章では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．

また，本章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．今後はより複雑な画像変換の自動構築への適用を行う予定である．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．

第４章　提案手法３：ＧＲＡＰＥ

４．１　はじめに

４．１．１　本章の目的と背景

近年，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．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．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．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．

本章では，グラフ構造を利用したプログラムの自動生成を行う Graph Structured Program Evolution (GRAPE)の提案を行う．GRAPE は先に述べた GP の問題点を克服する新しい手法であり，特徴として次の 3 点が挙げられる．

1. ループ構造によるモジュール性の確保（Modularity）
2. 複数のデータ型の取り扱い（Multiple data types）
3. 進化計算による効率の良い自動構築（Efficient automatic construction）

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．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，性能の検証を行った．

４．１．２　本章の構成

本章の構成は次のとおりである．まず 4.2 では，本研究と関連の深い従来研究について述べる．次に 4.3 では，本章の提案手法である Graph Structured Program Evolution (GRAPE)について説明を行う．4.4 では提案手法をいくつかのプログラムの自動生成の問題に適用した実験の詳細とその結果と考察を述べ，提案手法の有効性を示す．最後に 4.5 で総括する．

4．2　関連研究

ここでは，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，グラフ表現を用いた自動プログラミングの従来研究について述べる．

4．2．1　進化計算（Evolutionary Computation; EC）

始めに，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）[26,27]，遺伝的プログラミング（Genetic Programming; GP）[4,5]，進化的プログラミング（Evolutionary Programming; EP）[28]，進化戦略（Evolution Strategy; ES）などが挙げられる．ここでは，現在活発に研究が行われている GA と GP について述べる．

4．2．1．1　遺伝的アルゴリズム（Genetic Algorithm; GA）[1-3]

遺伝的アルゴリズム（Genetic Algorithm; GA）[26, 27]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 4.1 のようになる．

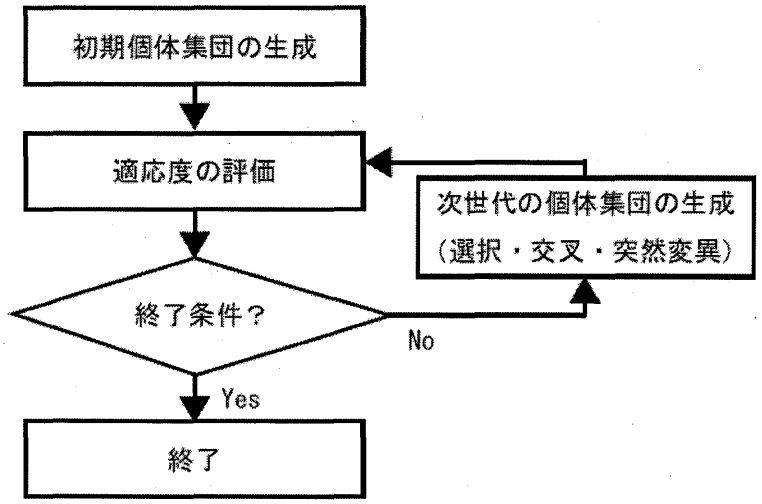


図4.1　GAの処理の流れ

GAでは，初めに初期個体集団として決められた数の個体数を作り出す．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．

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SimpleGA（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，選択した集団から2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）[29]がある．MGG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出する（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[29-31].
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回数を決める．

- **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.

4．2．1．2　遺伝的プログラミング（Genetic Programming; GP）[4-8]

遺伝的プログラミング（Genetic Programming; GP）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 4.2** に遺伝操作の例を示す.

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

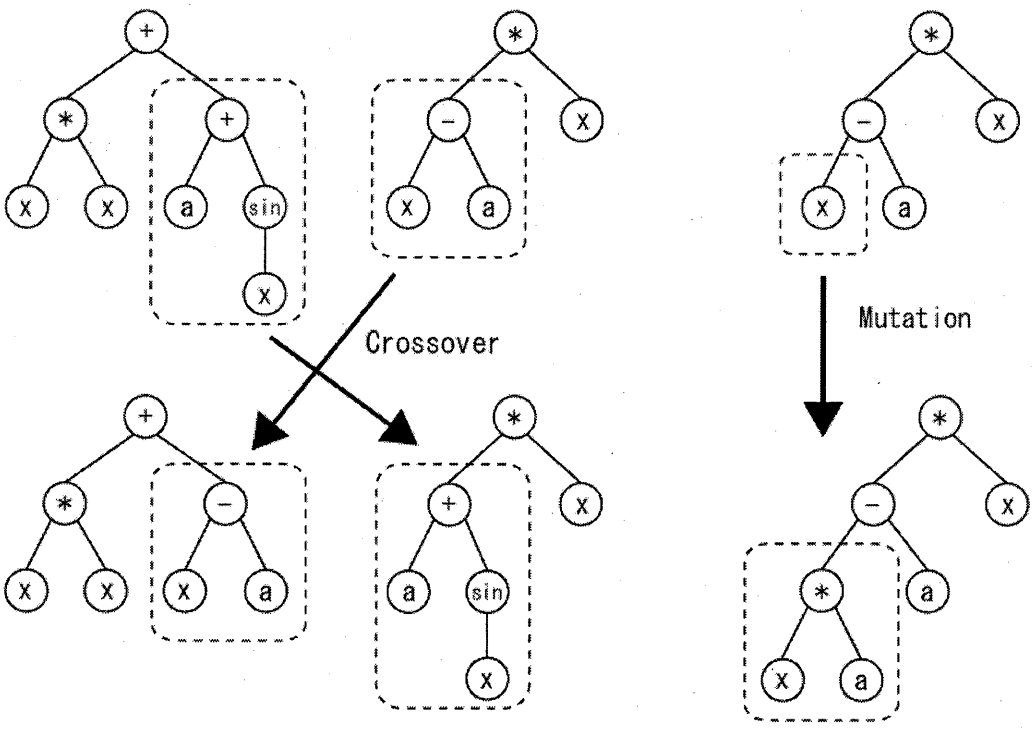


図 4.2　GP の遺伝操作の例

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．

４．２．１．３　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的生成が 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）[5]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）[32,33]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 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）[34]や自動的マクロ定義（Automatically Defined Macros; ADM）[35]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[36]．また，リストのソートアルゴリズムの進化などにも GP は適用されている[37,38]．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[39, 40]．

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）[41]は Montana により提案された．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，容易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[38]．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てフィボナッチ数列を生成する実験を行った[4]．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．

4．2．2　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，4.2.1.2 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，進化計算を用いた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[6,7].

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，パラメータ可能なサブルーチン，階層的に呼ばれるサブルーチン，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. **人間のプログラマー，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．**

本章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．ここでは，いくつかの自動プログラミングの手法について述べる．

4．2．2．1　GP with index memory （インデックスメモリ付き GP）[9-11]

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[11]．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．メモリに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する．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．

4．2．2．2 Linear Genetic Programming（LGP）[12]

Linear Genetic Programming（LGP）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 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 3.3 に示す．LGP では図 4.3 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 4.3　LGP のプログラムの例

4．2．2．3 Grammatical Evolution（GE）[13-15]

Grammatical Evolution（GE）は任意の言語を使用できる進化計算である．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 4.4 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 4.4　GE の文法ルールの例

4．2．2．4　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 [42-44]と Object Oriented Genetic Programming（OOGP）[45-47]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，ソーティングなど）に適用され，再帰構造を利用してプログラムの自動生成に成功している．

4．2．3　グラフ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構築する手法について述べる．

4．2．3．1　Parallel Algorithm Discovery and Orchestration（PADO）[16-18]

Parallel Algorithm Discovery and Orchestration（PADO）はグラフ構造であり，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 4.5 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 のプログラムの実行は，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．

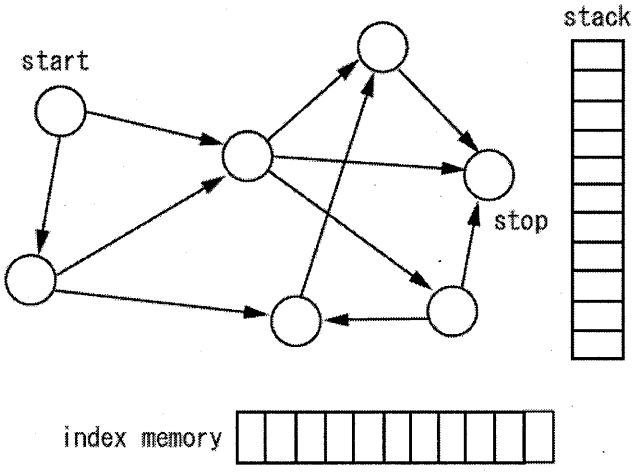


図 4.5　Parallel Algorithm Discovery and Orchestration（PADO ）の構造例

4．2．3．2 Parallel Distributed Genetic Programming（PDGP）[19] と Cartesian Genetic Programming（CGP）[20, 21]

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

4．2．3．3 遺伝的ネットワークプログラミング（Genetic Network Programming; GNP）[22-25]

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

4．2．3．4 その他のグラフ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[48]は LGP をグラフ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 GAUGE [49]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り問題解決を行う．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．

4．2．4 まとめ

本項では，本研究の関連研究として進化計算によるいくつかの自動プログラミング手法を紹介した．次章ではこれらの従来研究を踏まえて，複雑なプログラムの自動生成を可能にする新たな手法の提案を行う．

4．3　GRAPE (GRAph structured Program Evolution)

本項では本章の提案手法である GRAph structured Program Evolution（GRAPE）について述べる．

4．3．1　概要

これまでに GP をはじめ数多くの自動プログラミング手法が提案されている．しかし，現在 GP の分野において扱われている問題よりも，もっと複雑なプログラムを自動的に生成しようとした場合，性能向上が必要であると考えられる．一般的な GP では木構造によるプログラムの表現をしているため，ループ構造や再帰構造は許されていない．複雑なプログラムの記述をする場合に，ループや再帰，関数はモジュール性の面からも非常に重要である．また，数値，文字列，画像などの異なるデータ型を一つのプログラムで扱う必要がある．さらに，木構造を扱う GP にはブロートという潜在的な問題が存在する．プログラムを自動生成する上で効率の良い探索は必須条件である．

これらを踏まえて，本章では次の 3 点が提案手法の満たすべき要件であると考える．

1．**モジュール性の確保**（Modularity）

2．**複数のデータ型の取り扱い**（Multiple data types）

3．**効率の良い自動構築**（Efficient automatic construction）

これらの観点において，それぞれ別個に GP でも様々な研究が行われてきているが，本章ではこの 3 点すべてを満たすシステムの提案を行う．

4．3．2　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 4.6** 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 4.7** 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．

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 4.6** 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 4.6** 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．

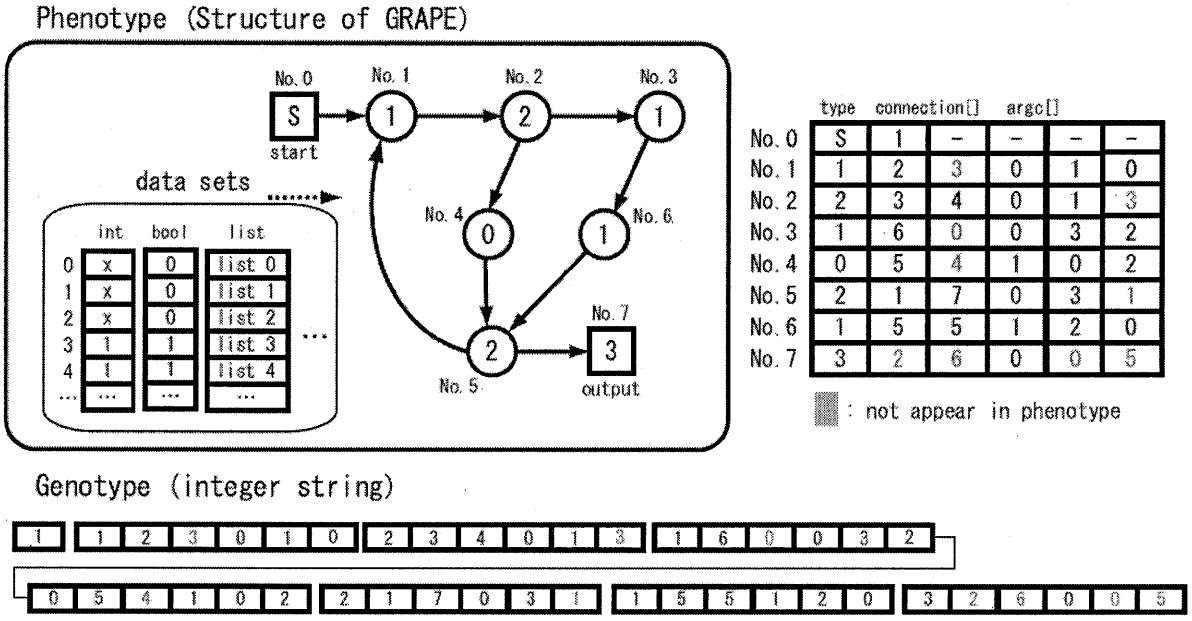


図 4.6　GRAPE の構造例

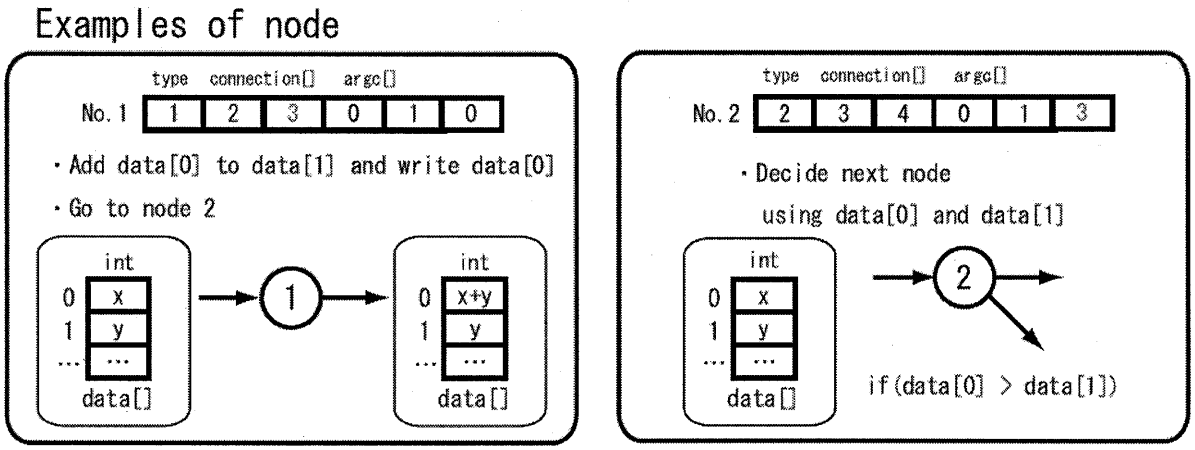


図 4.7　GRAPE のノード例

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．各ノードでは指定さ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 のプログラムを構成するノードの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 はノードの数，n_c は接続の最大数，n_a は引数の最大数である．

4．3．3　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．本章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 Pc 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 4.8 に示す．突然変異は突然変異率 Pm 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 4.9 に示す．

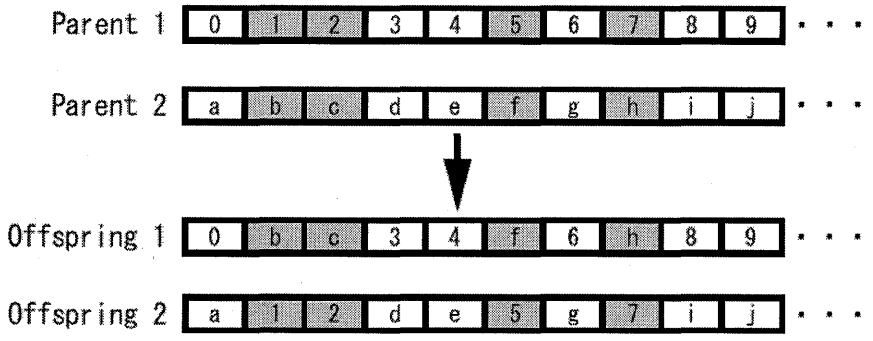


図 4.8　GRAPE の交叉の例

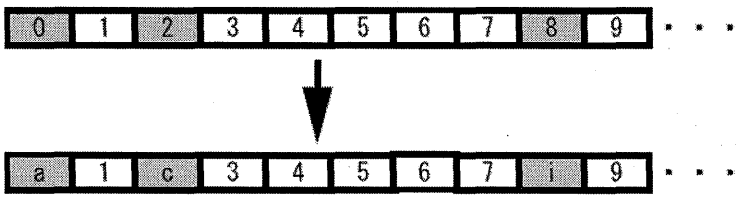


図 4.9　GRAPE の突然変異の例

4．3．4　GRAPE の特徴

ここでは 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．

まず 4.3.1 で挙げた，提案手法が満たすべき 3 つの要件と GRAPE の特徴を比較する．

1. **モジュール性の確保（Modularity）**
GRAPE では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．そのためモジュール性の高いプログラムを生成することができると考えられる．

2. **複数のデータ型の取り扱い（Multiple data types）**
GRAPE では“データセット” に複数のデータ型を用意することで，複数のデータ型を扱うことが可能である．

3. **効率の良い自動構築（Efficient automatic construction）**
GRAPE では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，4.2.1.2 で述べた 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．

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．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．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る．そのため，ある程度表現を制限している PDGP，CGP，Linear-Graph GP などとは表現の自由度が異なる．

4．3．5 まとめ

ここではグラフ構造を採用することで，複雑なプログラムの自動生成を行う提案手法 Graph Structured Program Evolution（GRAPE）について述べた．次項では GRAPE をいくつかのプログラム自動生成の問題に適用し，その結果を考察する．

４．４　自動プログラミングへの適用実験

本項では提案手法 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．

４．４．１　基本的なプログラムの自動生成

ここでは GRAPE の性能を検証するため，階乗，フィボナッチ数列，累乗，リストを反転させるプログラムの自動生成の実験を行う．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 では解くことができない．

４．４．１．１　実験の設定

本実験での GRAPE のパラメータ値を**表 4.1** に示す．GRAPE の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．

表4.1　実験に用いた各パラメータ値

umber of evaluations	2500000
Population size	500
Crossover rate (P_c)	0.9
Mutation rate (P_m)	0.02
The number of nodes	10, 30, 50
Execution step limits	500

GRAPE のプログラムの進化の戦略として，本実験では SimpleGA (SGA), Minimal Generation Gap (MGG), 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比較するために用いた．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 2,500,000 個体生成するまでとした．

本実験で用いた GRAPE のノード関数を**表 4.2** に示す．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などの基本的なものであり，問題ごとに設計は行わない．

表 4.2 GRAPE のノード関数（factorial, Fibonacci sequence, exponentiation and reversing a list）

Name	# Connections	# Args.	Argument(s)	Description
+	1	3	x, y, z	Use integer data type. Add data[x] to data[y] and substitute for data[z].
−	1	3	x, y, z	Use integer data type. Subtract data[x] from data[y] and substitute for data[z].
*	1	3	x, y, z	Use integer data type. Multiply data[x] by data[y] and substitute for data[z].
/	1	3	x, y, z	Use integer data type. Divide data[x] by data[y] and substitute for data[z].
=	2	2	x, y	If data[x] is equal data[y] connection 1 is chosen else connection 2 is chosen.
>	2	2	x, y	If data[x] is greater than data[y] connection 1 is chosen else connection 2 is chosen.
<	2	2	x, y	If data[x] is less than data[y] connection 1 is chosen else connection 2 is chosen.
SwapList	1	2	x, y	Use integer type and a list data. Swap list[data[x]] for list[data[y]].
OutputInt	0	1	x	Output integer data[x] and then the program halts.
OutputList	0	0	-	Output a list data and then the program halts.

4．4．1．2 factorial（階乗を求めるプログラムの自動生成）

ここでは階乗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．トレーニングセットの（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．

$$fitness = 1.0 - \frac{\sum_{i=1}^{n} \frac{|Correct_i - Estimate_i|}{|Correct_i| + |Correct_i - Estimate_i|}}{n} \tag{4.1}$$

ここで，$Correct_i$ はトレーニングデータ i の正しい出力値，$Estimate_i$ 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（4.1）で求めた適応度が 1.0 であった場合，評価関数は次の式（4.2）を使用する．

$$fitness = 1.0 + \frac{1}{S_{exe}} \tag{4.2}$$

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，少ないノードの遷移回数の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，評価値として 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．

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 4.2** に示した{+, -, *, =, >, <, OutputInt}を用いた．

実験は同一のパラメータ条件で 100 回の試行を行った．**図 4.10** に成功率（success rate）の推移を示す．成功率とは次の式で算出される値である．

$$\text{Success rate} = \frac{\text{Number of successful runs}}{\text{Total number of nuns}} \tag{4.3}$$

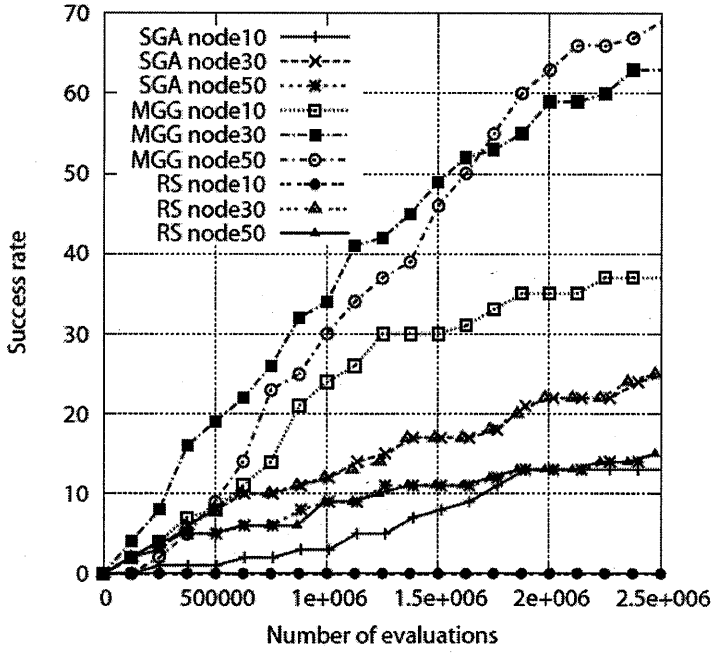


図 4.10 階乗における成功率の比較

今回の実験では，評価値が 1.0 を超える個体が現れた場合，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成功としている．

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．**表 4.3** にトレーニングデータとテストデータに対する成功率を示す．

表4.3　それぞれの問題に対する成功率の比較

	Factorial		Fibonacci Sequence		Exponentiation		Reversing a List	
	Training	Test	Training	Test	Training	Test	Training	Test
SGA node10	13%	12%	0%	0%	9%	9%	22%	20%
SGA node30	25%	23%	0%	0%	7%	7%	41%	30%
SGA node50	16%	16%	0%	0%	5%	5%	37%	34%
MGG node10	37%	37%	2%	2%	34%	34%	22%	21%
MGG node30	63%	57%	**8%**	**6%**	**45%**	**44%**	63%	56%
MGG node50	**69%**	**59%**	3%	2%	41%	40%	**71%**	**65%**
RS node10	0%	0%	0%	0%	0%	0%	0%	0%
RS node30	1%	1%	0%	0%	0%	0%	0%	0%
RS node50	15%	13%	0%	0%	6%	1%	0%	0%

図 4.11 は本実験で GRAPE が生成した構造の例である．ループを含む構造となっており，このプログラムは完全に階乗を求めるプログラムとなっている．

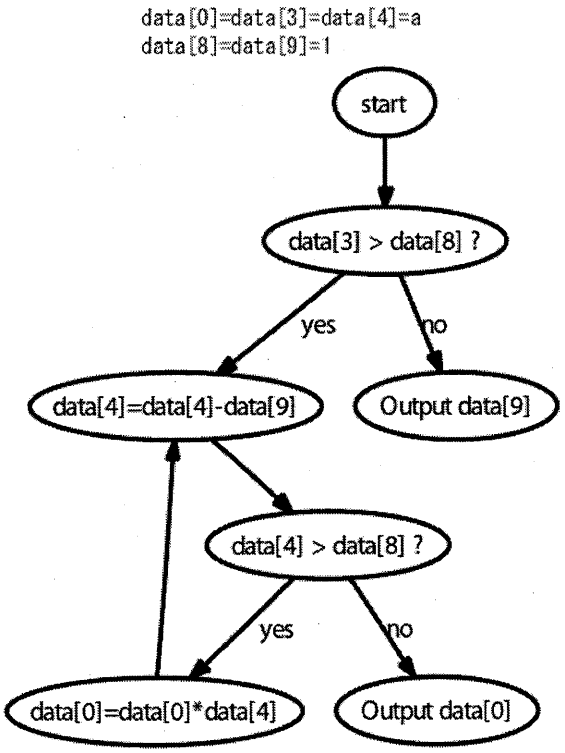


図 4.11　GRAPE が獲得した構造の例（階乗）

4. 4. 1. 3 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を算出す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（4.1）と（4.2）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 4.2** に示した{+, -, *, =, >, <, OutputInt}を用いた．

実験は同一のパラメータ条件で 100 回の試行を行った．**図 4.12** に成功率（success rate）の推移を示す．

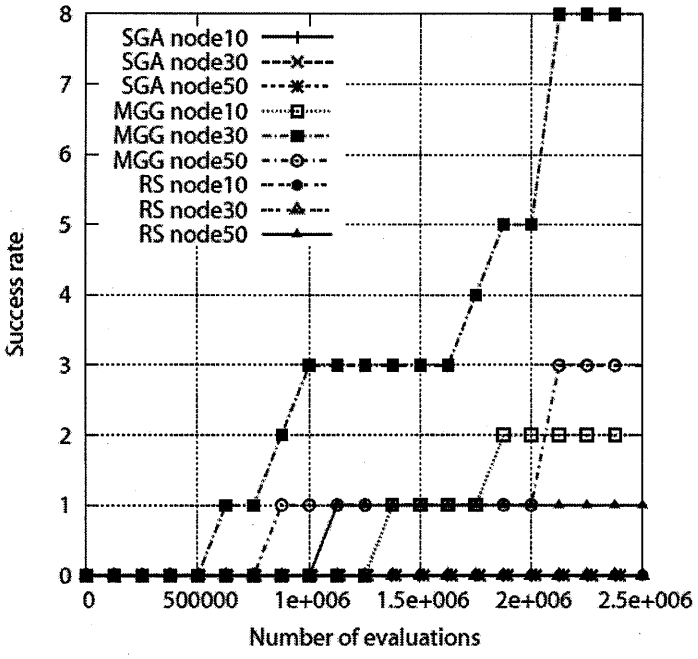


図 4.12 フィボナッチ数列における成功率の比較

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 4.3** にトレーニングデータとテストデータに対する成功率を示す．

図 4.13 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，このプログラムは完全にフィボナッチ数列を求めるプログラムとなっている．

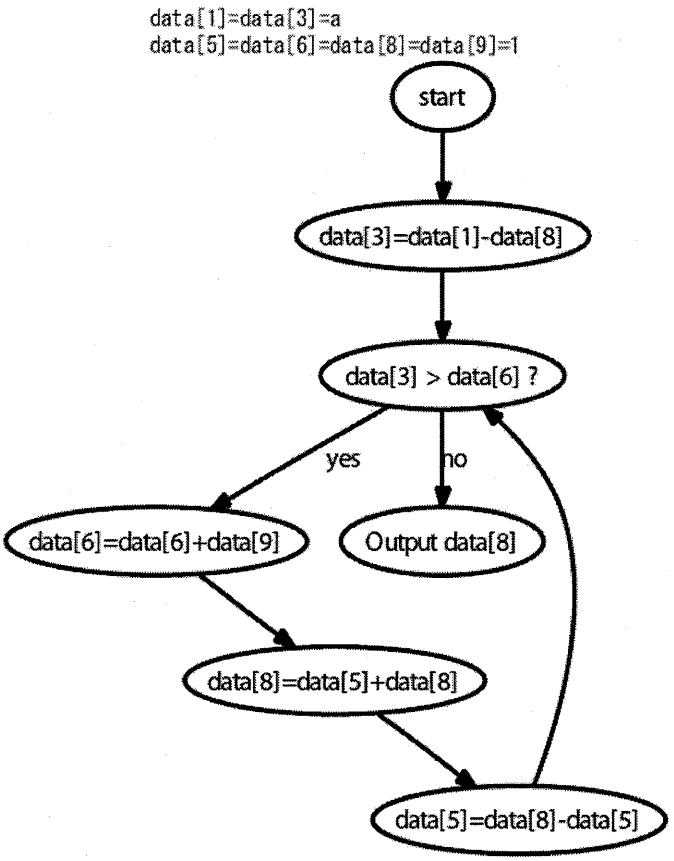


図 4.13 GRAPE が獲得した構造の例（フィボナッチ数列）

4. 4. 1. 4 Exponentiation（累乗を求めるプログラムの自動生成）

ここでは累乗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, 9）（3, 4, 27）（3, 5, 81）（4, 6, 4096）（4, 7, 16384）（4, 8, 65536 ）を使用した．

評価関数には式（4.1）と（4.2）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 4.2 に示した{+, -, *, =, >, <, OutputInt}を用いた．

実験は同一のパラメータ条件で 100 回の試行を行った．図 4.14 に成功率（success rate）の推移を示す．

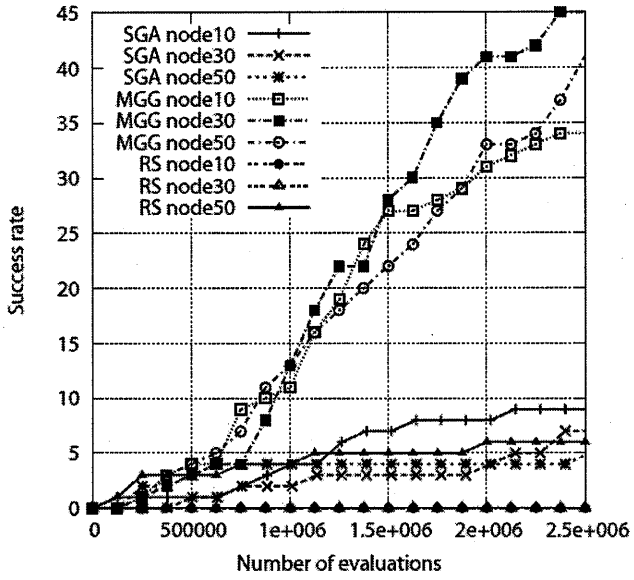


図 4.14 累乗における成功率の比較

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 4.3 にトレーニングデータとテストデータに対する成功率を示す．

図 4.2 (c)は本実験で GRAPE が生成した構造の例である．このプログラムも完全に累乗を求めることができるプログラムとなっている．

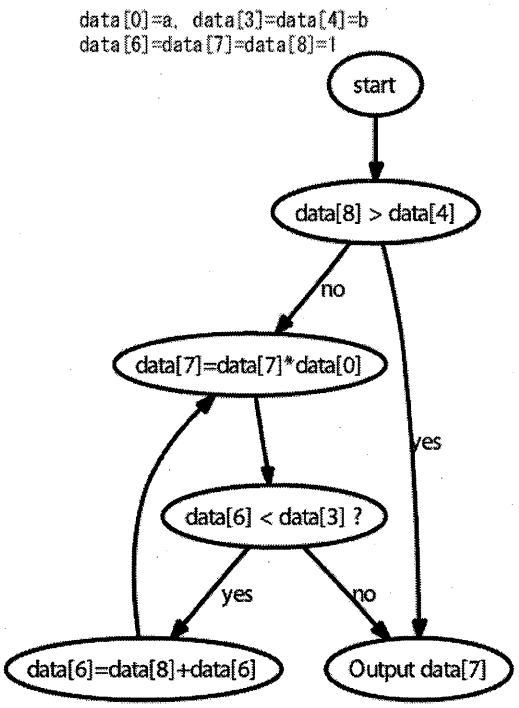


図 4.15 GRAPE が獲得した構造の例（累乗）

4．4．1．5　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．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（1234）出力（4321））．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（4.4）で算出される．

$$fitness = 1.0 - \frac{\sum_{i=1}^{n} \frac{\sum_{j=0}^{l} \left(1 - \frac{1}{2^{d_j^i}}\right)}{l_i}}{n} \tag{4.4}$$

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(4.4)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(4.2)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4.2**に示した{+, -, *, /, =, >, <, SwapList, OutputInt}を用いた．

実験は同一のパラメータ条件で100回の試行を行った．**図4.16**に成功率（success rate）の推移を示す．

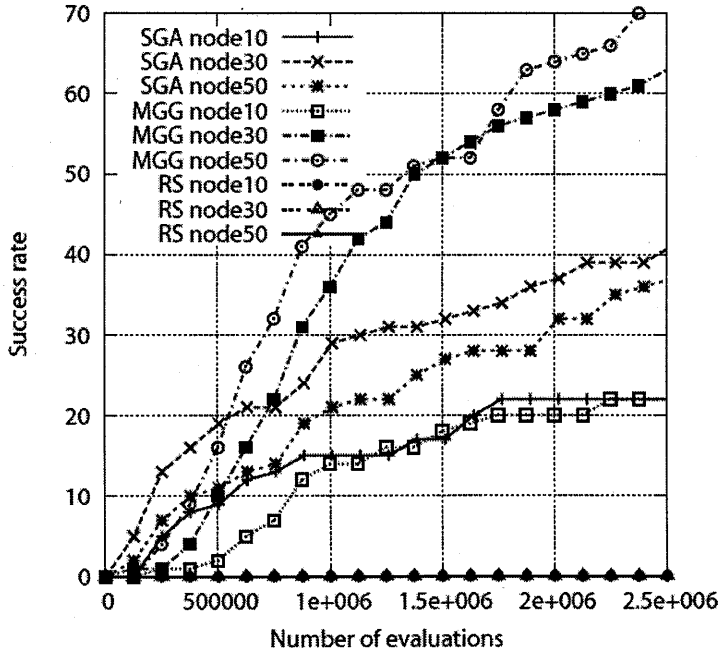


図4.16　リストの反転における成功率の比較

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 4.3** にトレーニングデータとテストデータに対する成功率を示す．

図 4.17 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．

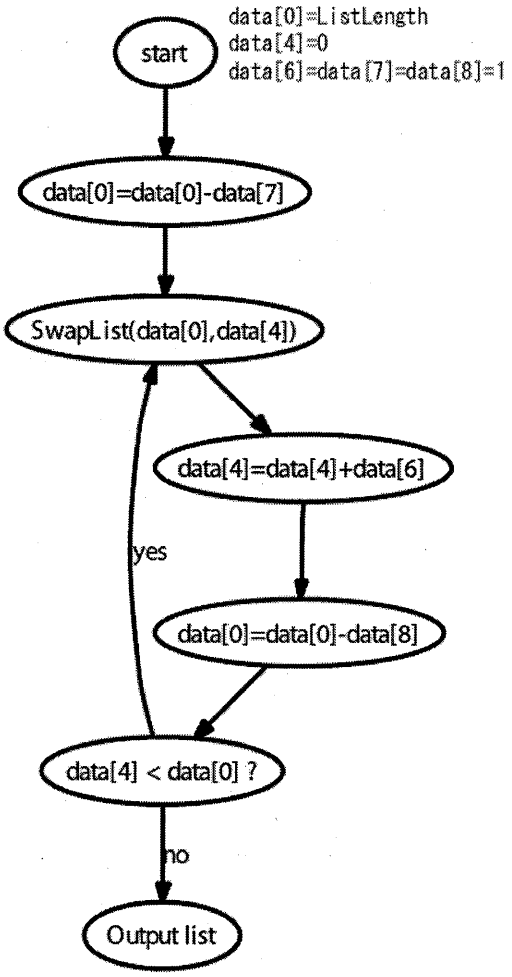


図 4.17 GRAPE が獲得した構造の例(リストの反転)

4．4．1．6　考察

本実験では進化の戦略として，SGA，MGG，RS の 3 つの方法を用いた．**図 4.10, 12, 14, 16** と**表 4.3** の成功率（success rate）を見ると，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率の上昇が観測でき，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．ノード数については戦略ごとに 10，30，50 としてそれぞれ 100 回の試行を行った．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率を示した．**図 4.11, 13, 15, 17** 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（これを進化計算において一般にイントロンと呼ぶ）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評価値の高い個体をテストデータに適用した際の成功率が**表 4.3** に示されている．テストデータに対して完全に正しい出力を返すということは，各問題において求めるアルゴリズムを完全に自動構築したといえる．**表 4.3** 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いえる．

次に，**図 4.11, 13, 15, 17** に示した GRAPE によって自動構築された構造について考察を行う．いずれの構造もループを含み，完全に目的の処理を行うプログラムとなっている．**図 4.11** 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 4.18** のように表現することができる．つまり，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 4.18** に示すような一般的なプログラムの獲得ができたといえる．こ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．

今回の実験で用いた 4 つの問題（階乗，フィボナッチ数列，累乗，リストの反転）は簡単な問題ではあるが，実現するためにはループもしくは再帰構造が必要となり，通常の木構造を扱う GP では解くことができない．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．

```
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 4.18　図 4.11 を C 言語形式に変換したプログラム

本実験で扱った階乗，フィボナッチ数列，累乗などの問題は従来手法を用いての自動生成の試みも行われている[44, 46]．しかしほとんどの場合，再帰構造を導入することで解決を図るもので，GRAPE のようにループを使って解いた例はなく，今回の実験結果は世界初の成果であるといえる．文献[46]によると階乗，フィボナッチ数列，累乗に対する成功率はそれぞれ 75%，25%，6%である．実験設定が異なるため一概に比較はできないが，GRAPE の各問題に対する成功率が従来手法と比べて遜色がないことがわかる．

今回扱った 4 つの問題は再帰構造を用いて表現することもできる．GRAPE を再帰構造が表現可能な形に拡張することで性能の向上が望めるかも知れない．これについては今後の課題で詳しく述べる．

4．4．2　ソートアルゴリズムの自動生成

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．これまでの 4 つの実験に比べて，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 4.4** に示すものを用いた．

表 4.4　GRAPE の各種パラメタ値（ソーティング）

Parameter	Value
The number of evaluations	5000000
Population size	500
Crossover rate P_c	0.9
Mutation rate P_m	0.02
The number of nodes	10, 30, 50
Execution step limits	3000

GRAPE のノード数については 10，30，50 と値を変えて実験を行った．最大ノード遷移回数（Execution step limits）には十分大きい数として 3000 を設定した．評価関数には Reversing a list の実験と同様に式（4.4）と（4.2）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 のリストを 30 例用いた．

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．本実験で用いた GRAPE のノード関数を**表 4.5** に示す．ノード関数は数値に対する四則演算とリストの比較，交換などの基本的なものである．

実験は同一のパラメータ条件で 100 回の試行を行った．**図 4.19** に成功率（success rate）の推移を示す．また，**図 4.20** に 100 回の試行の平均適応度の推移を示す．

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 10 から 50 のリストを 100 例用いた．テストデータ適用時の最大ノード遷移回数（Execution step limits）には十分大きい数として 10000 を設定した．**表 4.6** にトレーニングデータとテストデータに対する成功率を示す．

表 4.5 GRAPE のノード関数（ソーティング）

Name	# Connections	# Args.	Argument(s)	Description
+	1	3	x, y, z	Use integer data type. Add data[x] to data[y] and substitute for data[z].
−	1	3	x, y, z	Use integer data type. Subtract data[x] from data[y] and substitute for data[z].
*	1	3	x, y, z	Use integer data type. Multiply data[x] by data[y] and substitute for data[z].
/	1	3	x, y, z	Use integer data type. Divide data[x] by data[y] and substitute for data[z].
=	2	2	x, y	If data[x] is equal data[y] connection 1 is chosen else connection 2 is chosen.
>	2	2	x, y	If data[x] is greater than data[y] connection 1 is chosen else connection 2 is chosen.
<	2	2	x, y	If data[x] is less than data[y] connection 1 is chosen else connection 2 is chosen.
SwapList	1	2	x, y	Use integer type and a list data. Swap list[data[x]] for list[data[y]].
EqualList	1	2	x, y	Use integer type and a list data. If list[data[x]] is equal list[data[y]] connection 1 is chosen else connection 2 is chosen.
GreaterList	1	2	x, y	Use integer type and a list data. If list[data[x]] is greater than list[data[y]] connection 1 is chosen else connection 2 is chosen.
LessList	1	2	x, y	Use integer type and a list data. If list[data[x]] is less than list[data[y]] connection 1 is chosen else connection 2 is chosen.
OutputList	0	0	-	Output a list data and then the program halts.

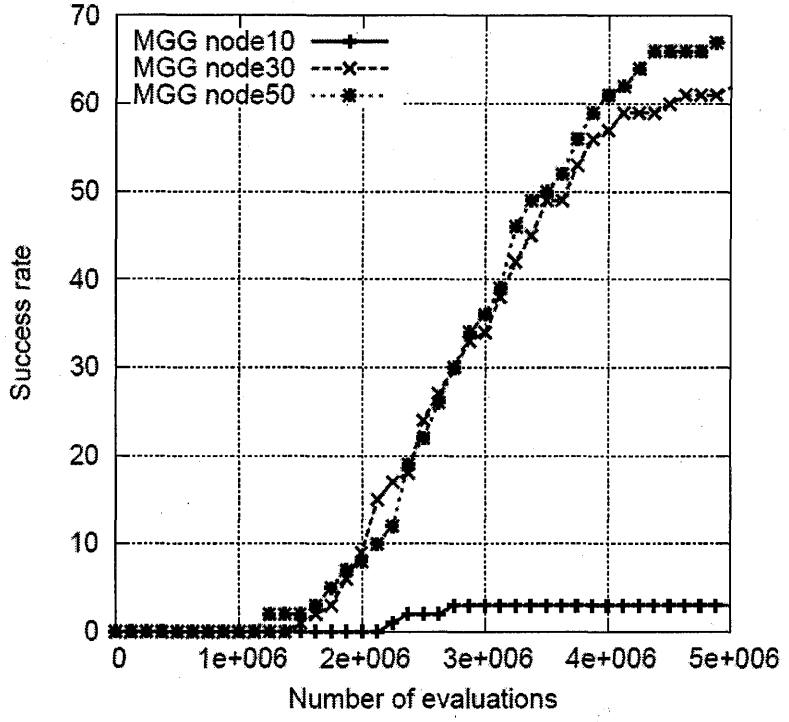


図 4.19　成功率の推移（Sorting a list）

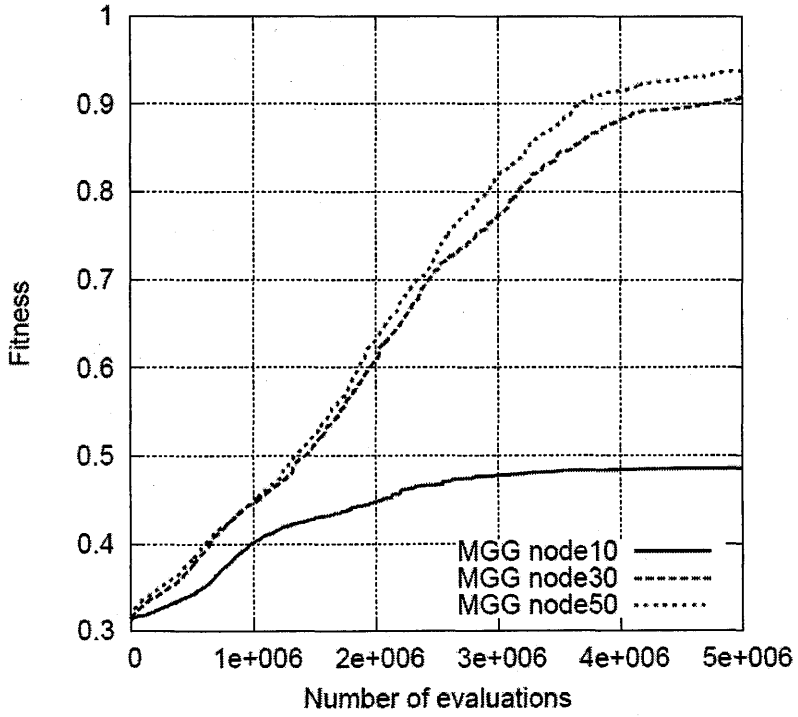


図 4.20　平均適応度の推移（Sorting a list, 100 回試行の平均）

表 4.6 成功率の比較 (Sorting a list)

	Training set	Test set
MGG node10	3%	3%
MGG node30	62%	**39%**
MGG node50	**67%**	28%

次に，成功率と獲得したプログラムについて実験結果の考察を行う．

成功率

図 4.19 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率の上昇が著しい．一方，ノード数が 10 の場合は最終的な成功率は 3%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．進化の前半では成功率がほぼ 0%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，ソートアルゴリズムの自動獲得が難しいことがうかがい知れる．図 4.20 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率はそれぞれ 62%，67%であった．半数以上の試行でトレーニングデータを満足するソートアルゴリズムが獲得されており GRAPE の性能の高を示しているといえる．また，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際の成功率が表 4.6 に示されている．結果はノード数が 30 の場合に 39%，50 の場合に 28%とトレーニングデータに比べ成功率が下がっ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(4.2)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．

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[37, 38]であったり，再帰を用いる手法[44, 47]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．また，各手法で設定が異なるため一概に比較はできないが，成功率 6 割という成果は当該分野においても世界トップレベルの性能であるといえる．

獲得したプログラム

ここではGRAPEが自動生成したプログラムについて考察を行う．GRAPEが自動生成したソートプログラムの一例を図 4.21 に示す．この構造にはループが 2 つ存在する．図 4.21 のソートプログラムを，C 言語の形式に変換すると，図 4.22 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．獲得した構造は先に行った 4 つの実験と比べノード数が多く，複雑なプログラムである．

提案手法である GRAPE は理にかなった正しいソートアルゴリズムを自動生成することに成功した．今回自動生成したプログラムはソートアルゴリズムの中でも計算時間がかかってしまうものであったが，クイックソートやマージソートと比べて実装しやすいプログラムであるといえる．今後，評価関数や自動生成手法に改良を加えることで，より効率の良いアルゴリズムをうまく自動生成することが可能になるかも知れない．

```
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 4.22　図 4.6 を C 言語形式に変換したプログラム

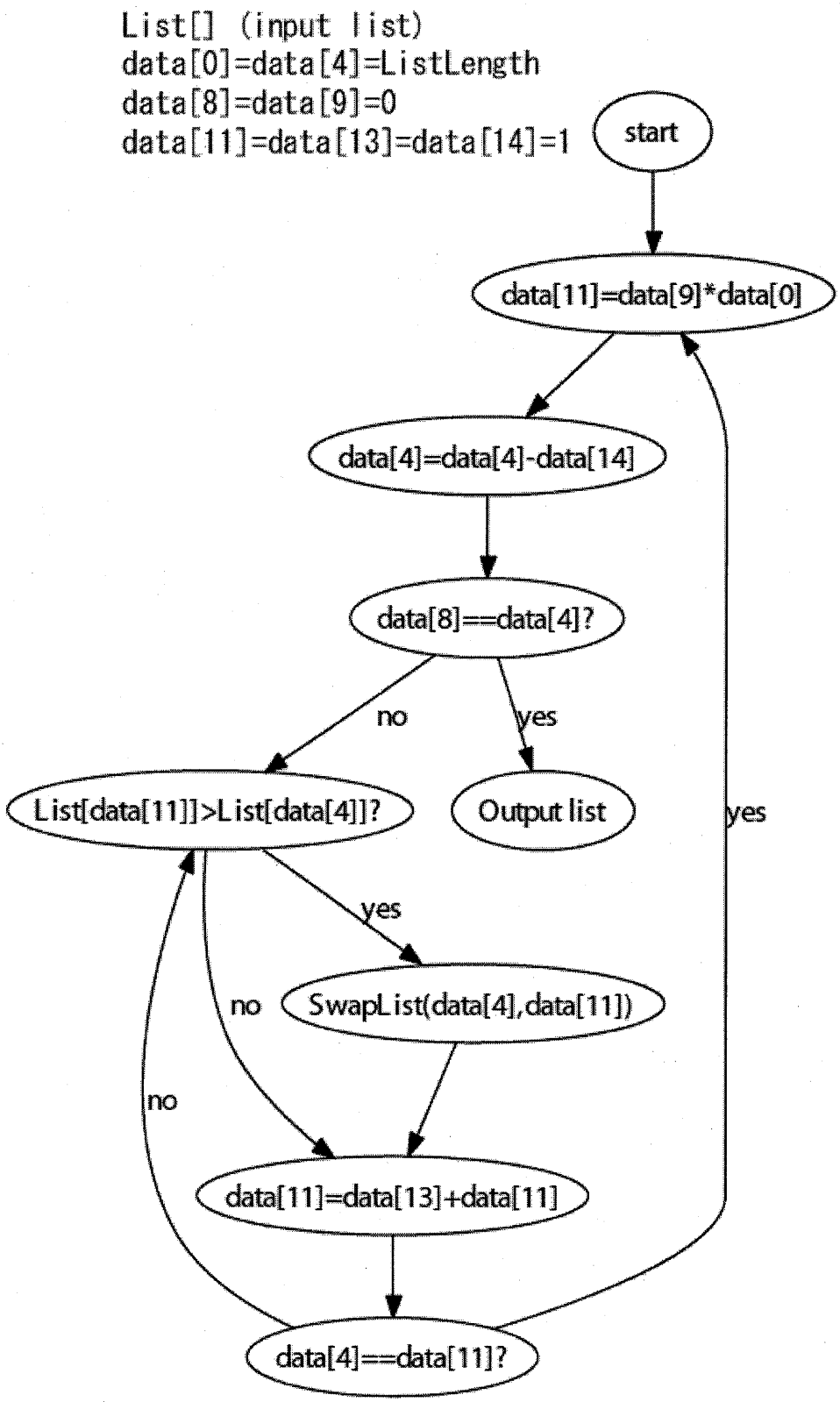


図 4.21　GRAPE が獲得した構造の例（Sorting a list）

４．４．３　本章のまとめ

本章では提案手法の GRAPE をいくつかのプログラムの自動生成の問題へ適用し結果を考察した．全ての実験において GRAPE はプログラムを自動的に生成することに成功し，提案手法の性能の高さと有効性を示すことができた．また，本実験の結果から，提案手法の GRAPE が満たすべき 3 つの要件，

1. モジュール性の確保（Modularity）
2. 複数のデータ型の取り扱い（Multiple data types）
3. 効率の良い自動構築（Efficient automatic construction）

が実際に実現できていることが示された．

4．5　本章の結論

4．5．1　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 Graph Structured Program Evolution（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 では解くことができない．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構築が行うことができていた．また，本章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研究成果は世界的にもまだ少なく，特にループ構造を使用して解いている例はほとんどない．各問題に対する成功率やテストデータに対する成功率は従来手法と同等以上であり，GRAPE の性能は非常に優れているといえる．

4．5．2　今後の課題

今後の課題として，まず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構造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．さらに，本章では既に知られているアルゴリズムの自動構築を行ったが，本手法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．その際，プログラムの評価に最終的な出力値の評価だけでなく，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必要になるかもしれない．

また，GRAPE の性能をより高めるために，モジュール性の面から ADF のような機能を導入することや再帰構造を表現できるようにすることが考えられる．再帰構造を用いることによって，本章で扱った問題も簡単に表現することができる．そのため，再帰構造を導入することによってより複雑なプログラムの自動生成ができるようになる可能性がある．

最後に，本章ではプログラムの進化の方法として GA を用いているが，よりプログラムの自動生成に向いた手法を考え出すことも今後の課題として挙げられる．提案手法ではグラフ構造状のプログラムを一次元の整数配列に変換し，数値列の遺伝子に対して遺伝操作を行っている．このことから，GA だけでなく近年注目されている，Ant Colony Optimization（ACO）[50,51]や Particle Swarm Optimization（PSO）[52,53]などの，他のメタヒューリスティック手法を適用することも可能である．これらの手法の GRAPE に対する探索能力の検討も興味深い．さらに，探索手法としてこれまでに提案されているような手法を用いるのではなく，“自動プログラミングのための探索手法” を考案することが最も重要であると考えられる．

第５章　結論

５．１　本研究のまとめ

本稿は平成１７年度から平成１８年度の２年間，基盤研究（B）として遂行された「**複雑な構造に対する新しい進化的計算法の開発とその動画像処理への応用**」の研究成果報告書である．本研究では，**文字列・数値列・木構造・ネットワーク構造・グラフ構造・メモリなどの構造が内部に混在して存在し，複雑に相互作用するような，従来の進化的計算法では適用不能な対象に対する有効な新しい進化的計算法を開発するとともに，その有効性を実問題に適用して検証する**ことを目的として研究を遂行した．

本研究では次の３つの新たな進化計算法を提案し，それらを実問題に適用した．

- GMA（Genetic Matrix Algorithm）：数値と構造を同時に最適化する進化計算法．
- GIN（Genetic Image Network）：ネットワーク構造を最適化する進化計算法．
- GRAPE（GRAph structured Program Evolution）：グラフ構造を最適化することで自動プログラミングを可能にする進化計算法．

これらはいずれも複雑な構造を最適化するもので，本研究の目的を達成するものである．

５．２　本研究の今後の課題

各方式に対する今後の課題を次に示す．

- GMA：学習データに対してパラメータも合わせて細かく調整できる反面，データによっては**過学習**になることがあるため，その対策が必要である．
- GIN：フィードバックループがあるため，ネットワーク中に何回信号を回すのかが重要であるが，その頑健性を確保する必要がある．
- GRAPE：Ｃ言語のポインタを実現することによって，リスト構造や木構造を含む動的データを扱うことができるよう改良する必要がある．

これらはいずれも現在検討中であり，近い将来に実現することができると考えられる．

参考文献

第2章の参考文献

1. Thomas B¨ack and Hans-Paul Schwefel. An overview of evolutionary algorithms for parameter optimization. Evolutionary Computation, Vol. 1, No. 1, pp. 1.23, 1993.
2. David B. Fogel. Evolutionary computation: toward a new philosophy of machine intelligence. IEEE Press, 1995.
3. A.E. Eiben and J.E. Smith. Introduction to Evolutionary Computing. Springer, 2003.
4. J.H. Holland. Adaptation in Natural and Artificial Systems. The Univ. Michigan Press, 1975.
5. D.E. Goldberg. Genetic Algorithm in Search, Optimization, and Machine Learning. Addison Wesley, 1989.
6. 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.
7. 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.
8. J.R. Koza. Genetic Programming: on the Programming of Computers by Means of Natural Selection. MIT Press, 1992.
9. 伊庭斉志. 遺伝的プログラミング. 東京電機大学出版局, 1996.
10. Jr. Kenneth E. Kinnear. Advances in Genetic Programming, Vol. 1, 2, 3. MIT Press, 1994, 1996, 1999.
11. W. Lee, J. Hallam, and H.H. Lund. A Hybrid GP/GA Approach for Co-evolving Controllers and Robot Bodies to Achieve Fitness-Specified Tasks. In Proceedings of IEEE ICEC, Nagoya, Japan, 1996.
12. L.M. Howard and D.J. D'Angelo. The GA-P: A Genetic Algorithm and Genetic Programming Hybrid. IEEE Expert Special Track on Evolutionary Programming, Vol. 10, pp. 11.15, 1995.
13. A.G. Ivakhnenko. Polynomial Theory of Complex Systems. IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-1, No. 4, pp. 364.378, 1971.
14. 伊庭斉志, 佐藤泰介. システム同定アプローチに基づく遺伝的プログラミング. 人工知能学会誌, Vol. 10, No. 4, pp. 100.110, 1995.
15. M. O'Neill and C. Ryan. Grammatical Evolution: Evolutionary Automatic Programming

in an Arbitrary Language. Kluwer Academic Publushers, 2003.
16. Conor Ryan, J.J. Collins, and Michael O'Neill. Grammatical evolution: Evolving programs for an arbitrary language. In Lecture Notes in Computer Science 1391, First European Workshop on Genetic Programming, EuroGP, pp. 83.96, Paris, France, 1998.
17. Conor Ryan, Michael O'Neill, and J.J. Collins. Grammatical evolution: Solving trigonometric identities. In Mendel, 4th International Mendel Conference on Genetic Algorithms, Optimization Problems, Fuzzy Logic, Neural Networks, Rough sets, pp. 111.119, 1998.
18. W. Banzhaf. Genotype-Phenotype-Mapping and Neutral Variation . A case study in Genetic Programming. In Lecture Notes in Computer Science 866, International Conference on Evolutionary Computation, pp. 322.332, Jerusalem, Israel, 1994.
19. 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. 情報処理学会論文誌, Vol. J70-D, No. 11, pp. 2147.2153, 1987.
20. 松山隆司, V. Hwang. 画像理解システムsigma.ボトムアップ, トップダウン解析の統合.. 情報処理学会論文誌, Vol. 26, No. 5, pp. 877.889, 1985.
21. 松山隆司, 尾崎正治. Llve: トップダウン・セグメンテーションのための画像処理エキスパートシステム. 情報処理学会論文誌, Vol. 27, No. 2, pp. 191.204, 1986.
22. 田村秀行, 佐藤宏明, 坂上勝彦, 久保文雄. Dia-expert システムとその知識表現方法. 情報処理学会論文誌, Vol. 29, No. 2, pp. 199.208, 1988.
23. 久保文雄, 佐藤宏明, 坂上勝彦, 田村秀行. 粒子画像解析エキスパートシステム dia-expert/pa1. 情報処理学会論文誌, Vol. 29, No. 2, pp. 209.219, 1988.
24. 青木紳也, 長尾智晴. 木構造状画像変換の自動構築actit. 映像情報メディア学会誌, Vol. 53, No. 6, pp. 888.894, 1999.
25. S. Aoki and Tomoharu Nagao. Automatic Construction of Tree-structural Image Transformation Using Genetic Programming. In International Conference of Image Processing, pp.529.533, Kobe, Japan, 1999.
26. 内藤孝平, 長尾智晴. 教師画像を自動修正する木構造状画像変換の自動生成法. 画像情報システム研究会. 映像情報メディア学会, 2001.
27. Random Number Generators. http://random.mat.sbg.ac.at/.

第3章の参考文献

1. Koza, J. R.: *Genetic Programming: On theProgramming of Computers by Means of Natural Selection*, MIT Press, Cambridge, MA, USA (1992).
2. Koza, J. R.: *Genetic Programming II: Automatic Discovery of Reusable Programs*, MIT Press, Cambridge, MA, USA (1994).
3. Teller, A. and Veloso, M.: PADO: A New Learning Architecture for Object Recognition, *Symbolic Visual Learning* (Ikeuchi, K. and Veloso, M., eds.), Oxford University Press, pp. 81–116 (1996).
4. Teller, A. and Veloso, M.: Program Evolution for Data Mining, *The International Journal of Expert Systems*, Vol.8, No.3, pp.216–236 (1995).
5. Teller, A. and Veloso, M.: Algorithm Evolution for Face Recognition: What Makes a Picture Difficult, *International Conference on Evolutionary Computation*, Perth, Australia, IEEE Press, pp.608–613 (1995).
6. Poli, R.: Evolution of Graph-like Programs with Parallel Distributed Genetic Programming, *Genetic Algorithms: Proceedings of the Seventh International Conference*, Michigan State University, East Lansing, MI, USA, Morgan Kaufmann, pp.346–353 (1997).
7. Miller, J. F. and Thomson, P.: Cartesian Genetic Programming, *Genetic Programming, Proceedings of EuroGP'2000*, LNCS, Vol.1802, Edinburgh, Springer-Verlag, pp.121–132 (2000).
8. Miller, J.F. and Smith, S.L.: Redundancy and Computational Efficiency in Cartesian Genetic Programming, *IEEE Transactions on Evolutionary Computation*, Vol.10, No.2, pp.167–174 (2006).
9. Katagiri, H., Hirasawa, K., Hu, J. and Murata, J.: Network Structure Oriented Evolutionary Model-Genetic Network Programmingand its Comparison with Genetic Programming, *2001 Genetic and Evolutionary Computation Conference Late Breaking Papers*, San Francisco, California, USA, pp.219–226 (2001).
10. 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).
11. 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).
12. 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).

13. 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-DII, No.11, pp.1982–1989 (1999).
14. 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).
15. 長尾智晴：進化的画像処理，昭晃堂(2002).
16. 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).
17. 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).
18. Nakano, Y. and Nagao, T.: 3D medical image processing using 3D-ACTIT; Automatic Construction of Tree-structural Image Transformation, *Pcoc of the International Workshop on Advanced Image Technology IWAIT-04*, Singapore, pp.529–533 (2004).
19. Nakano, Y. and Nagao, T.: Automatic extraction of internal organs region from 3D PET image data using 3D-ACTIT, *Pcoc of the International Workshop on Advanced Image Technology IWAIT-2006*, Okinawa, Japan (2006).
20. 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).

第4章の参考文献

1. 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.
2. 北野宏明. 遺伝的アルゴリズム. 産業図書, 1993.
3. 伊庭斉志. 遺伝的アルゴリズムの基礎. オーム社, 1994.
4. John R. Koza. Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, 1992.
5. John R. Koza. Genetic Programming II: Automatic Discovery of Reusable Programs. MIT Press, 1994.
6. 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, San Francisco, CA, USA, January 1998.
7. 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.
8. 伊庭斉志. 遺伝的プログラミング. 東京電機大学出版局, 1996.
9. A. 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.
10. Astro Teller. Genetic programming, indexed memory, the halting problem, and other curiosities. In Proceedings of the 7th annual Florida Artificial Intelligence Research Symposium, pp. 270.274, Pensacola, Florida, USA, May 1994. IEEE Press.
11. Astro Teller. Turing completeness in the language of genetic programming with indexed memory. In Proceedings of the 1994 IEEE World Congress on Computational Intelligence, Vol. 1, pp. 136.141, Orlando, Florida, USA, 27-29 June 1994. IEEE Press.
12. 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.
13. Conor Ryan, J. J. Collins, and Michael O' Neill. Grammatical evolution: Evolving programs for an arbitrary language. In Wolfgang Banzhaf, Riccardo Poli, Marc Schoenauer, and Terence C. Fogarty, editors, Proceedings of the First European Workshop on Genetic Programming, Vol. 1391 of LNCS, pp. 83.95, Paris, 14-15 April 1998. Springer-Verlag.
14. Michael O' Neill and Conor Ryan. Grammatical evolution. IEEE Transactions on Evolutionary Computation, Vol. 5, No. 4, pp. 349.358, August 2001.
15. Michael O' Neill and Conor Ryan. Grammatical Evolution: Evolutionary Automatic Programming in a Arbitrary Language, Vol. 4 of Genetic programming. Kluwer Academic

Publishers, 2003.

16. 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.
17. Astro Teller and Manuela Veloso. Program evolution for data mining. The International Journal of Expert Systems, Vol. 8, No. 3, pp. 216.236, 1995.
18. 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.
19. Riccardo Poli. Evolution of graph-like programs with parallel distributed genetic programming. In Thomas Back, editor, Genetic Algorithms: Proceedings of the Seventh International Conference, pp. 346.353, Michigan State University, East Lansing, MI, USA, 19-23 July 1997. Morgan Kaufmann.
20. Julian F. Miller and Peter Thomson. Cartesian genetic programming. In Riccardo Poli, Wolfgang Banzhaf, William B. Langdon, Julian F. Miller, Peter Nordin, and Terence C. Fogarty, editors, Genetic Programming, Proceedings of EuroGP' 2000, Vol. 1802 of LNCS, pp. 121.132, Edinburgh, 15-16 April 2000. Springer-Verlag.
21. 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.
22. 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.
23. Kotaro Hirasawa, M. Okubo, J. Hu, and J. Murata. Comparison between genetic network programming (GNP) and genetic programming (GP). In Proceedings of the 2001 Congress on Evolutionary Computation CEC2001, pp. 1276.1282, COEX, World Trade Center, 159 Samseong-dong, Gangnam-gu, Seoul, Korea, 27-30 May 2001. IEEE Press.
24. Hironobu Katagiri, Kotaro Hirasawa, Jinglu Hu, and Junichi Murata. Network structure oriented evolutionary model-genetic network programming-and its comparison with genetic programming. In Erik D. Goodman, editor, 2001 Genetic and Evolutionary Computation Conference Late Breaking Papers, pp. 219.226, San Francisco, California, USA, 9-11 July 2001.
25. 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.
26. John H. Holland. Adaptation in Natural and Artificial Systems. University of Michigan Press, 1975.
27. David E. Goldberg. Genetic Algorithms in Search, Optimization, and Machine Learning. Addison

Wesley, 1989.

28. L. J. Fogel, A. J. Owens, and M. J. Walsh. Artificial Intelligence through Simulated Evolution. John Wiley, 1966.
29. 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. 1.10, 1997.
30. H. Kita, I. Ono, and S. Kobayashi. Multi-parental extension of the unimodal normal distribution crossover for real-coded genetic algorithms. In Proceedings of the 1999 Congress on Evolutionary Computation (CEC99), Vol. 2, pp. 1581.1587, 1999.
31. S. Tsutsui, M. Yamamura, and T. Higuchi. Multi-parent re-combination with simplex crossover in real coded genetic algorithms. In In Proceedings of Genetic and Evolutionary Computation Conference (GECCO' 99), pp. 657.664, 1999.
32. 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.
33. P. J. Angeline and J. B. Pollack. Evolutionary module acquisition. In D. Fogel and W. Atmar, editors, Proceedings of the Second Annual Conference on Evolutionary Programming, pp. 154.163, La Jolla, CA, USA, 25-26 1993.
34. J. P. Rosca and D. H. Ballard. Learning by adapting representations in genetic programming. In Proceedings of the 1994 IEEE World Congress on Computational Intelligence, Orlando, Florida, USA, 27-29 June 1994. IEEE Press.
35. 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, Cambridge, MA, USA, 1996.
36. Peter Nordin and Wolfgang Banzhaf. Genetic reasoning evolving proofs with genetic search. In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, editors, Genetic Programming 1997: Proceedings of the Second Annual Conference, pp. 255.260, Stanford University, CA, USA, 13-16 July 1997. Morgan Kaufmann.
37. Kenneth E. Kinnear, Jr. Generality and difficulty in genetic programming: Evolving a sort. In Stephanie Forrest, editor, Proceedings of the 5th International Conference on Genetic Algorithms, ICGA-93, pp. 287.294, University of Illinois at Urbana-Champaign, 17-21 July 1993. Morgan Kaufmann.
38. 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.
39. [39] W. B. Langdon. Evolving data structures using genetic programming. In L. Eshelman, editor, Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95), pp. 295.302, Pittsburgh, PA, USA, 15-19 July 1995. Morgan Kaufmann.

40. William B. Langdon. Genetic Programming and Data Structures: Genetic Programming + Data Structures = Automatic Programming!, Vol. 1 of Genetic Programming. Kluwer, Boston, 24 April 1998.
41. David J. Montana. Strongly typed genetic programming. Evolutionary Computation, Vol. 3, No. 2, pp. 199.230, 1995.
42. Lee Spector. Autoconstructive evolution: Push, pushGP, and pushpop. In Lee Spector, Erik D. Goodman, Annie Wu, W. B. Langdon, Hans-Michael Voigt, Mitsuo Gen, Sandip Sen, Marco Dorigo, Shahram Pezeshk, Max H. Garzon, and Edmund Burke, editors, Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001), pp. 137.146, San Francisco, California, USA, 7-11 July 2001. Morgan Kaufmann.
43. 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.
44. Lee Spector, Jon Klein, and Maarten Keijzer. The push3 execution stack and the evolution of control. In Hans-Georg Beyer, Una-May O' Reilly, Dirk V. Arnold, Wolfgang Banzhaf, Christian Blum, Eric W. Bonabeau, Erick Cantu-Paz, Dipankar Dasgupta, Kalyanmoy Deb, James A. Foster, Edwin D. de Jong, Hod Lipson, Xavier Llora, Spiros Mancoridis, Martin Pelikan, Guenther R. Raidl, Terence Soule, Andy M. Tyrrell, Jean-Paul Watson, and Eckart Zitzler, editors, GECCO 2005: Proceedings of the 2005 conference on Genetic and evolutionary computation, Vol. 2, pp. 1689.1696, Washington DC, USA, 25-29 June 2005. ACM Press.
45. Simon Lucas. Exploiting reflection in object oriented genetic programming. In Maarten Keijzer, Una-May O' Reilly, Simon M. Lucas, Ernesto Costa, and Terence Soule, editors, Genetic Programming 7th European Conference, EuroGP 2004, Proceedings, Vol. 3003 of LNCS, pp. 369.378, Coimbra, Portugal, 5-7 April 2004. Springer-Verlag.
46. Alexandros Agapitos and Simon M. Lucas. Learning recursive functions with object oriented genetic programming. In Pierre Collet, Marco Tomassini, Marc Ebner, Steven Gustafson, and Anik ´ o Ek ´ art, editors, Proceedings of the 9th European Conference on Genetic Programming, Vol. 3905 of Lecture Notes in Computer Science, pp. 166.177, Budapest, Hungary, 10 -12 April 2006. Springer.
47. Alexandros Agapitos and Simon M. Lucas. Evolving efficient recursive sorting algorithms. In Proceedings of the 2006 IEEE Congress on Evolutionary Computation, pp. 9227.9234, Vancouver, 6-21 July 2006. IEEE Press.
48. Wolfgang Kantschik and Wolfgang Banzhaf. Linear-graph GP.A new GP structure. In James A. Foster, Evelyne Lutton, Julian Miller, Conor Ryan, and Andrea G. B. Tettamanzi, editors, Genetic Programming, Proceedings of the 5th European Conference, EuroGP 2002, Vol. 2278 of LNCS, pp. 83.92, Kinsale, Ireland, 3-5 April 2002. Springer-Verlag.
49. 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.

50. 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.
51. Marco Dorigo and Thomas Stutzle. Ant Colony Optimization. The MIT Press, 2004.
52. J.Kennedy and R.Eberhart. Particle swarm optimization. IEEE International Conference on Neural Networks, Perth, Australia, pp. 1942.1948, 1995.
53. J.Kennedy and R.Eberhart. Swarm Intelligence. Morgan Kaufmann Publishers, 2001.

謝辞

本研究に対して科学研究費補助金（基盤研究（B））を交付して下さり，本研究の実施を可能にして下さった独立行政法人 日本学術振興会様に御礼申し上げます．また，GMA の開発担当者である博士課程後期学生の藤嶋　航君，および GIN と GRAPE の開発担当者である博士課程後期学生の白川真一君のご協力に感謝します．さらに，本補助金の経理・事務の仕事を担当してくれた長尾研究室秘書の上北紗枝さん，橋本愛花さん，谷 由希子さんに感謝いたします．