



HOKKAIDO UNIVERSITY

Title	Image-Value ペアに基づく遺伝的プログラミングを用いた画像処理プログラムの自動生成に関する研究
Author(s)	土居, 意弘
Degree Grantor	北海道大学
Degree Name	博士(情報科学)
Dissertation Number	甲第11939号
Issue Date	2015-06-30
DOI	https://doi.org/10.14943/doctoral.k11939
Doc URL	https://hdl.handle.net/2115/59678
Type	doctoral thesis
File Information	Munehiro_Doi.pdf



博 士 論 文

Image-Value ペアに基づく遺伝的プログラミングを用いた
画像処理プログラムの自動生成に関する研究

Studies on Automatic Generation of Image Processing
Programs by using Genetic Programming based on
Image-Value Pairs.

北海道大学大学院情報科学研究科

土 居 意 弘

要旨

近年、画像処理アプリケーションの利用が、ますます広がっている。背景には、コンピュータの小型高性能化と、認識や識別の分野において信頼性が高く優れた画像処理アルゴリズムの登場がある。しかし、扱う画像データは、映り込むものの種類や数が異なったり、画像撮影の環境や使用機器などに起因して画像の明るさや陰影が変化したりするなど、幅広い条件を考慮する必要がある。そのため、実用的なアプリケーションでは、主となるアルゴリズムだけで目的を満足することは難しく、ノイズ除去や注目する特徴量の強調などを目的とした前処理を行うことが一般的である。前処理の開発は、既存の標準的な画像フィルタの組み合わせで行われることが多い。しかしながら、画像処理設計は知識や経験に基づくノウハウに依存している部分が多く、専門家であっても手間のかかる作業であり、開発生産性上の課題となっている。この課題を解決するため、遺伝的プログラミングにより画像処理プログラムを自動生成する方法が提案されている。遺伝的プログラミングとは、プログラムを生物個体に見立てて、生物の進化のように最適化を進める手法である。プログラムはサブルーチンが木構造に接続されたもので表現され、良いプログラムはその特長を引き継いだ子孫を残し、悪いプログラムは淘汰される。次世代の個体は、2つのプログラムの一部を交換したり、あるいはプログラムの一部をランダムに入れ替えたりする遺伝的操作を用いて生成される。

画像処理プログラムは、画像データだけでなく閾値や分岐条件などに利用される数値データも取り扱うことが一般的である。しかし、遺伝的プログラミングの交叉や突然変異操作によって異なる入出力データ型のサブルーチンが接続されてしまうと、実行できないプログラムが生成されてしまうため、複数のデータ型が混在したプログラムを生成することは困難であった。従来手法では、接続ルールを与えて入出力型に配慮した遺伝的操作を行う方法や、画像フィルタの順序に制約を与える方法などが用いられているが、適用する課題によりルールや制約を変える必要があるため、より一般的に利用できる簡潔な方

法が望ましい。また、遺伝的プログラミングでは、世代交代を重ねると個体サイズが急速に肥大してしまう現象があり、プログラムサイズを制限することが行われる。しかし、事前に適切なプログラムサイズはわからないため、サイズ制約は避けたい。

本論文では、これらの問題を解消するため、Image-Value ペアという概念を提案する。遺伝的プログラミングの個体表現を、Image-Value ペアを入出力として処理するサブルーチン（Image-Value フィルタ）をノードとする木構造にすることで、遺伝的操作におけるデータ型の混在を無理なく解消することができる。Image-Value フィルタの内部構造は画像演算と数値演算の2つのフィルタからなっており、個体の評価を行う場合は、これらの内部フィルタで構成されたプログラムに変換してから実行する。また、変換と同時に個体の評価結果に寄与しない冗長なフィルタを除去することにより、個体サイズの肥大が増大しても、実質的な演算量の増加を抑えることができる。

提案手法の有効性を確認するため、自然画像からの背景除去課題および画像クラス分け課題に適用し、入力画像と期待する出力画像を与えるだけで、未知画像に対しても頑健な性能を示すプログラムを自動生成できることを確認した。

目次

要旨	3
第 1 章 序論	13
1.1 研究背景	13
1.2 提案手法の概要	18
1.3 本論文の構成	18
第 2 章 遺伝的プログラミング	21
2.1 遺伝的アルゴリズム	21
2.1.1 個体の遺伝子型と表現型	23
2.1.2 初期個体群の生成	24
2.1.3 選択	24
2.1.4 遺伝的操作	25
2.1.5 世代交代モデル	25
2.2 遺伝的プログラミング	28
2.2.1 個体表現	28
2.2.2 遺伝的操作	29
2.2.3 適応度の評価	31
2.2.4 初期世代の生成	31
2.2.5 ブロートとイントロン	31
2.2.6 種々の遺伝子表現	33
2.3 多目的遺伝的アルゴリズム	34
第 3 章 画像処理プログラムの自動生成と課題	37
3.1 関連研究	37
3.2 課題	38
3.2.1 画像処理と数値演算の混在	39

3.2.2	プログラムサイズ制限	39
3.3	本研究の目的	40
第 4 章	提案手法	41
4.1	Image-Value ペア	41
4.1.1	Image-Value ペア	41
4.1.2	Image-Value フィルタ	42
4.1.3	Image-Value ツリー	43
4.2	遺伝的プログラミングへの導入方法	44
4.3	提案手法の課題解決アプローチ	47
第 5 章	背景除去プログラムの自動生成	49
5.1	課題の定義	49
5.2	背景除去プログラムの自動生成	49
5.2.1	システム設計	49
5.2.2	適応度関数の設計	51
5.2.3	パラメータ	52
5.2.4	適用結果	55
5.3	考察	59
5.4	まとめ	60
第 6 章	画像分類プログラムの自動生成	63
6.1	画像分類プログラムの自動生成	63
6.1.1	システム設計	63
6.1.2	適応度関数の設計	65
6.1.3	パラメータ	66
6.1.4	適用結果	66
6.2	背景除去プログラムの強化への応用	69
6.2.1	システム設計	69
6.2.2	適応度関数の設計	70
6.2.3	パラメータ	70
6.2.4	適用結果	70
6.3	考察	70
6.4	まとめ	72
第 7 章	考察	73

7.1	個体サイズの増加	73
7.2	画像, 数値以外のデータへの拡張性	74
第 8 章	結論	75
付録 A	Image-Value ツリー評価の最適化	77
A.1	Image-Value ツリー評価の課題	77
A.2	メモリ使用効率を最適化する評価順序	81
A.3	並列計算の利用	82
A.4	個体間共通部分の計算の統合	83
A.5	保存データの圧縮	84
付録 B	本論文で用いた Image-Value フィルタ	87
謝辞		91
参考文献		93
学術研究論文等研究業績一覧		101
学会誌		101
国際会議		101
学術講演		101
索引		103

目次

1.1	画像フィルタの設計手順.	17
2.1	遺伝的アルゴリズムの一般的な手順.	22
2.2	遺伝的アルゴリズムの GTYPE, PTYPE の例.	23
2.3	遺伝的アルゴリズムの交叉オペレータ.	26
2.4	遺伝的アルゴリズムの突然変異オペレータ.	27
2.5	木構造.	29
2.6	S 式の例.	29
2.7	遺伝的プログラミングの交叉オペレータ.	30
2.8	遺伝的プログラミングの突然変異オペレータ.	30
4.1	IV フィルタの例.	42
4.2	IV フィルタの内部構造の例.	42
4.3	Image-Value ツリーの概念図.	43
4.4	Image-Value ツリーの例.	44
4.5	Image-Value ツリーの内部構造.	45
4.6	Image-Value ツリーの PTYPE.	46
5.1	背景除去課題の教師データセット.	50
5.2	背景除去フィルタのブロック図.	51
5.3	終端ノードとして与えた画像.	54
5.4	最良個体の適応度推移.	55
5.5	獲得したプログラムの PTYPE.	56
5.6	獲得したプログラムの概略.	57
5.7	獲得したフィルタを教師データに適用した結果.	58
5.8	平均演算数およびイントロンを除いた平均演算数の推移.	59

5.9	獲得したプログラムで処理された教師画像および未知画像と目標画像との PSNR.	60
5.10	獲得したフィルタを未知画像に適用した結果.	61
6.1	画像分類器のブロック図.	64
6.2	強化型背景除去フィルタのブロック図.	69
A.1	偏ったノード配置を持つ木の例.	80
A.2	一般的なトポロジカルソートを用いた場合のノードの実行順序.	81
A.3	Sarkar-Serrano-Simons 法による効率的なノードの実行順序.	82
A.4	Image-Value グラフへの変換.	83

表目次

3.1	プログラム自動生成手法の比較.	40
5.1	背景除去フィルタ生成課題の設計.	53
5.2	背景除去フィルタ自動生成課題で使用する終端ノード.	53
6.1	画像分類器作成に使用するトレーニング画像.	64
6.2	画像分類課題の設計.	66
6.3	サブ分類器のトレーニング結果.	67
6.4	分類器を未知画像に適用した結果.	68
6.5	強化型背景除去フィルタを未知画像に適用した場合の PSNR.	71
A.1	Image-Value フィルタの相対的実行時間.	78
A.2	第 5000 世代の最良個体および全個体に含まれる Image-Value フィルタ の種類別個数.	79
A.3	サブ分類器の自動生成に必要なとなったディスク容量	85
B.1	利用可能な IV フィルタ.	88
B.2	内部フィルタ.	89

第 1 章

序論

画像処理を応用した実用アプリケーションが、様々な場面で利用されている。背景には、コンピュータの高性能化と信頼性が高く優れた画像処理アルゴリズムの登場がある。ただし、単一のアルゴリズムを適用することで目的が達成できることは少なく、主アルゴリズムがより安定に精度良く働くよう、不要な情報を削除したり特徴を強調したりする前処理が課題に合わせて開発されることが多い。しかしながら、画像処理設計は知識や経験に基づくノウハウが必要な、専門家であっても手間のかかる作業であり、開發生産性上の課題となっている。この課題を解決するため、遺伝的プログラミングにより画像処理プログラムを自動生成する方法が提案されている。本論文では、Image-Value ペアという概念を導入し、一般の遺伝的プログラミングでは困難であった、複数のデータ型の演算ができるプログラムの生成を可能とする方法を提案する。

本章では、本論文の研究背景および画像処理プログラムの自動生成における従来研究を概観し、提案手法についての概要と、本論文の構成について述べる。

1.1 研究背景

画像処理の応用が、産業分野などだけでなく、スマートフォンと組み合わせたものなどの消費者向け機器にも広がりつつある。背景として、コンピュータの高性能化と優れた画像処理アルゴリズムの登場がある。コンピュータは CPU の高速化だけでなく、マルチコアや Single Instruction Multiple Data (SIMD) 演算が取り込まれて、並列演算性能が向上したことで、より高度な画像処理を行うことができるようになった。さらに、従来 2D や 3D コンピュータグラフィックスの描画のために用いられてきた Graphic Processing Unit (GPU) を汎用的に利用する手法が広がり演算能力が大幅に向上した [1]。スマートフォンなどの消費者向け機器にも標準的に組み込まれている。画像処理アルゴリズムは、セグメンテーション [2, 3, 4, 5], 顔検出 [6], 物体認識 [7, 8, 9], クラス分類 [10], などの

分野で、定番として使えるほど信頼性が高く優れた画像処理アルゴリズムが提案されてきた。これらは複雑な演算を含む場合もあるが、近年のコンピュータ性能の向上に伴い実用的な速度で実行できるようになり、利用が急速に広がってきたものと考えられる。

しかし、実用的なアプリケーションを開発する場合、画像に映り込むものの色、角度、大きさ、あるいは、明るさ・陰影のできかたなど、幅広い条件を考慮する必要がある。産業用途では、撮影環境を厳密にコントロールすることができるが、スマートフォンやデジタルカメラなどで撮影されるような、自然に近い環境で撮影された画像の場合は、特に難しい問題となる。そのため、単一のアルゴリズムで目的が達成できることは少なく、主となるアルゴリズムがより安定に働くよう、注目する特徴量を強調するフィルタなどを前処理として適用することが一般的である。

前処理はその特性上、画像条件に合わせて個別に設計されるため、応用アプリケーションごとに、既存のフィルタを組み合わせたり、新しいフィルタを設計したりして、新しいアルゴリズムを開発する必要がある。

部品となる基礎的なフィルタにはこれまで十分な研究がなされており、エッジ抽出、平滑化、微小領域抽出など標準的なものが多数ある [11, 12, 13]。前処理の開発は、その中から選んで組み合わせることで行うことができる。しかしながら、画像処理設計は知識や経験に基づくノウハウに依存しているところが大きく [14, 15]、画像処理専門家であっても試行錯誤が必要で手間のかかる作業であり、開發生産性上の課題となっている。

この課題に対する研究として、開発手法の標準化による、専門性を低減する試みがある。

阿刀田らは、画像処理の設計は一般的な指針がなく提案者の優れたアイデアに依存していると指摘し、知識や経験が少ない開発者でも、平滑化やエッジ抽出を拡張した低水準フィルタリング処理の設計を可能にするための方法論として、専用の記述方法を用いたプログラミックフィルタを提案している [14]。画像処理開発経験の浅い学生であっても、わずかなトレーニング時間で新しい非線形フィルタを設計することができるようになると報告している。

知識ベースのシステムあるいはエキスパートシステムと呼ばれる分野に属する手法も多数提案されている。エキスパートシステムは、知識を「A は B である」「C ならば D」のような論理記述（ルール）の形で蓄積したデータベースと、目的をパラメータやプログラムの形で入力した場合にそのデータベースからルールの連鎖を導き出す推論エンジンからなるものである。

坂上らは、画像処理・解析手続きを能率良く設計するにはかなりの知識や経験を必要とするが、この種の知識は画像処理専門家の個人的なノウハウの形で蓄積されているに過ぎないと指摘している [15]。この問題の一解決法として、ある画像処理モジュールを実行するためのデータやパラメータ等を正しい形式で与える作業を、処理モジュールの引数に関するデータベースを用いて、自動的にコーディングするシステムを提案している。

松山らは、画像理解のためのエキスパートシステム SIGMA [16] のモジュールとして、画像処理エキスパートシステム Low Level Vision Expert (LLVE) [17] を提案している。LLVE は、「画像特徴フレーム」と「処理プロセスフレーム」を用いて画像処理をルール化する。画像特徴フレームは、データ型の名称と属性を定義するクラスで、例えば LINE という画像特徴には、属性として開始点座標、チェインコード、長さなどが含まれる、ということを書述する。処理プロセスフレームは、画像特徴を別の画像特徴に変換する操作を定義するクラスで、例えば REGION-TO-POLYGON という処理プロセスは、REGION 画像特徴を POLYGON 画像特徴に変換する処理を表す。属性には、変換に使われる具体的なアルゴリズムとパラメータ、入出力の画像特徴インスタンス、などが含まれる。2 値化、モルフォロジーフィルタ、輪郭抽出、多角形近似などが記述可能で、表現力は高い。LLVE で画像処理プログラムを導出する手順は、次のようになる。まず、ユーザーが、ある画像特徴のインスタンスとして表現された入力データと、画像特徴フレーム種別やその属性の制約条件を記述したゴールを、システムに入力する。次に、システムが、入力画像特徴からゴール画像特徴に到達するような、処理プロセスの連鎖（ネットワーク）を導出する。つづいて、システムは、入力画像特徴側から処理プロセスを activate していく。activate された処理プロセスは、入力画像特徴インスタンスから出力画像特徴に対する制約条件を満たすような画像処理アルゴリズムとパラメータを、実際に画像処理を実行しながら探索する。アルゴリズムとパラメータが見つかったら、出力画像特徴インスタンスが生成されて、次の処理プロセスを activate していく。これをゴールまで繰り返していく。

IMPRESS は、事例ベースで画像から図形を抽出するプログラムを生成するエキスパートシステムである [18, 19, 20]。IMPRESS では、原画像とサンプル図形の組（標本）を与えることで、原画像からそれを抽出するための処理手順を自動構成するため、ゴールのルールをプログラミングする必要がない。自動構成は、標本個別手順の探索と、それらの集約の 2 段階で行われる。まず、標本ごとに求める画像処理を行う手順を探索する。平滑化、差分、二値化、図形融合、小領域除去というおおまかな処理手順は決まっており、「処理手順の良さの尺度（評価値）」が最大となるように、各手順の利用有無とパラメータ（閾値やマスクサイズ）を変えていく。評価値は、典型的にはサンプル図形と抽出された図形が一致する度合いのことである。次に、各標本に対する最適手順を、全標本に適用可能な単一あるいは少数の手順に集約する処理を行う。この際、1) 設計標本に対する平均的な評価値最大、2) 設計標本に対する最小の評価値がある閾値よりも大きいこと、3) 集約処理の計算コストを可能な限り削減、の 3 点を考慮する。

2000 年代に入り、エキスパートシステムは、専門家が新しい知識のデータ化を継続的に行わなければならない、運用コストが高いことが認識されるようになり [21, 22]、事例から自動的にデータ化を行う方法に移行してきた。

事例データから画像処理プログラム自体を自動生成する手法として代表的なものの一つ

は、遺伝的プログラミング (Genetic Programming, GP) を用いた方法である [23, 24, 25]. 遺伝的プログラミングは、生物集団の進化の仕組みから着想を得たプログラムの最適化フレームワークである [26]. 演算子が木構造に組み合わされたプログラムを生物個体に見立て、良い個体やその子孫を次世代に残していくという戦略で世代交代を繰り返し、徐々に良いプログラムを獲得していくというものである. 個体の良さは適応度 (fitness value, fitness) で測られ、プログラムの実行結果が目的に近いほど良い値を持つような適応度関数 (fitness function) で評価される. 例えば、プログラムの構成単位として標準的な画像フィルタを多数しておき、目的とする結果が得られるようなフィルタの選び方や実行順序を探索する. これは、LLVE や IMPRESS などのエキスパートシステムが画像処理ルールの知識データベースを基礎としているのに対し、知識の発見も含めて自動化しようとする考え方といえる. ルールの蓄積が人手に依存しているため運用上のボトルネックとなりやすいという、エキスパートシステムの課題に対する一つの解決アプローチである.

前述のとおり、遺伝的プログラミングによるプログラミングの自動生成は、前処理開発の多くがフィルタの組み合わせで行われる点との整合性が高く、また事例ベースで生成できることから画像処理の専門的な知識が少なく済むため、実用アプリケーションの開発生産性の課題解決に適していると考えられる.

図 1.1(a) は、従来の人手による画像処理システム開発の手順である. (1) 処理目的を定義し、(2) 処理前のデータと処理後に期待される結果を定義する. (3) さらに処理結果の評価基準を定め、(4) 画像処理プログラムを開発して試行し、(5) 評価を行う. 評価基準を満たすものが出来上がるまで (4)(5) を繰り返す. (6) 完成したプログラムを利用する. (4) と (5) の画像処理プログラムの生成と試行を繰り返す部分は、評価基準を満たすものが出来上がるまで何十回と繰り返されることもあり、手間がかかる作業である. 完成したプログラムは (6) で画像処理システムに組み込んで前処理として利用される.

自動生成による画像処理プログラムの開発は、図 1.1(b) のようになる. (1) 処理目的を定義し、(2) 処理前のデータと処理後に期待される結果を定義する. (3) さらに処理結果の評価関数を定め、(4) 画像処理プログラムを自動生成して試行し、(5) 評価を行う. 評価基準を満たすものが出来上がるまで (4)(5) を繰り返す. (6) 完成したプログラムを利用する. 自動生成の場合は、(4) と (5) は何十万回も行われる可能性があるが、コンピュータによって自動的に行われるため、人手は不要となる. 完成したプログラムを利用する方法は、手作業による開発と同様である.

しかし、課題もある. 一番目の課題は、遺伝的プログラミングでは、画像処理と数値演算の両方を含むプログラムの生成に関して、制約がある点である. 遺伝的プログラミングを用いた画像処理の自動生成では、処理手順の種類や順序をランダム的に組み替えていくため、画像を入力して数値を出力する処理のように異なるデータ型を扱うことが難しい. これは前後の処理のデータ受け渡しに矛盾が生じると実行できないプログラムが生成され

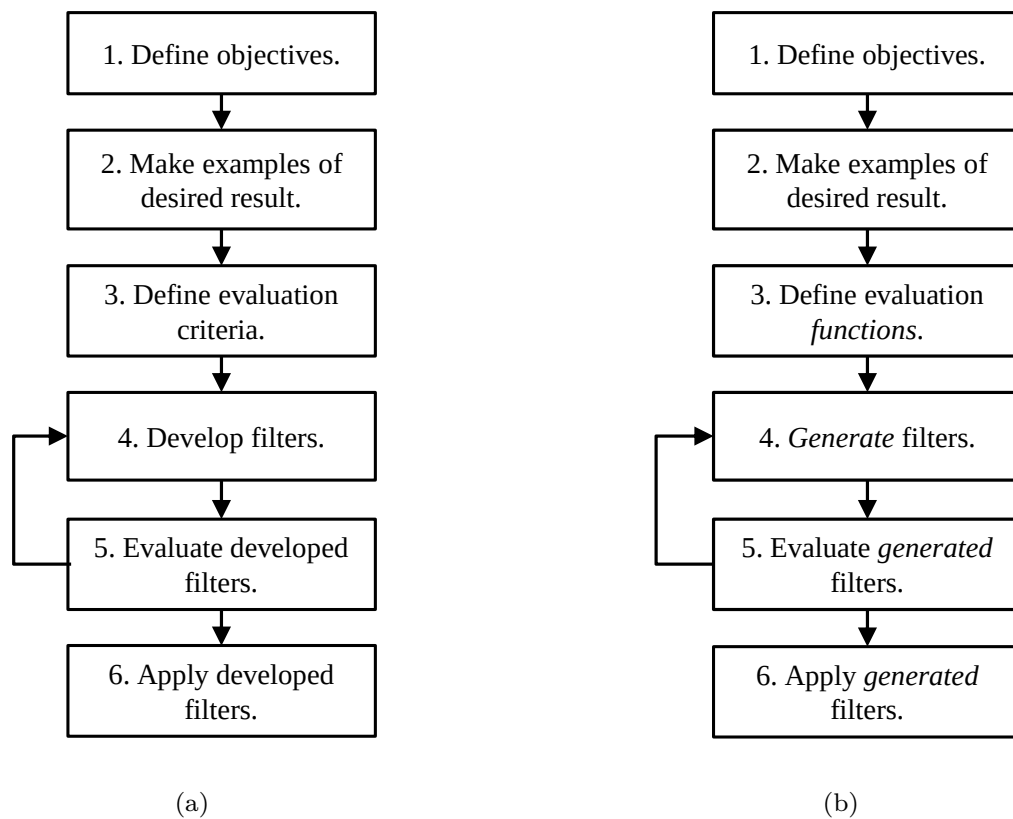


図 1.1: 画像フィルタの設計手順. (a) 従来の画像処理フィルタ開発の設計手順. (b) 自動生成を用いた場合の画像処理フィルタ開発の設計手順.

てしまうためである. しかし, 画像処理プログラムには画像フィルタの閾値, 条件分岐など数値パラメータが必要である. また, 入力画像から抽出した特徴量を別のフィルタのパラメータとして用いることができれば, 画像条件に対して頑健なプログラムの設計にもつながるため, 画像と数値の2種類のデータが混在するプログラムを生成できることが望まれる. 従来手法では, 画像と数値のように複数のデータ型がある場合は, 遺伝的操作に特別な設計や制約を加える必要があった.

二番目の課題は, プログラムサイズに関する制約である. 遺伝的プログラミングの個体表現は可変長であり, プログラムサイズが大きくなれば適応度評価や遺伝的操作に要する時間が増加する. このため, プログラムサイズの大きさと探索実行時間とのトレードオフを検討する必要がある. さらに, 世代交代を繰り返すと世代の平均プログラムサイズが指数関数的に増加するブloat (bloat) と呼ばれる現象への対策が重要である. ブloatが発生すると演算時間やメモリ使用量が急速に増大し, 時には遺伝的プログラミング自体が実行不可能になることもある. このため, 従来手法のほとんどでサイズ制限や増大に対するペナルティなどの対策を行っている. しかし, 適切なプログラムサイズは事前にはわか

らないため、設定が難しいパラメータの一つである。プログラムサイズはプログラムの表現力に強い影響があるため、ブロート対策を行いつつ、プログラムサイズに対する制約を大幅に緩和することが望ましい。

1.2 提案手法の概要

本論文では、遺伝的プログラミングで画像処理と数値演算が混在したプログラムを生成するための汎用的な枠組みとして、Image-Value ペアを提案する [27, 28, 29]。Image-Value ペアとは、画像データと数値データを組にしたもので、これによりノードの接続順序などに制約を導入することなく、遺伝的プログラミングで数値パラメータを持つ画像処理および数値パラメータの計算の両方を含むプログラムを生成することができる。さらに、Image-Value ペアを利用した遺伝的プログラミングで効率良く個体の評価を実行するために、結果に寄与しない冗長な計算を事前除去する方法について提案する。

例えば、提案手法を多目的遺伝的プログラミングと組み合わせ、平均的な性能向上と安定性を両立した頑健なプログラムを生成することができる。提案手法により、画像処理専門家ではない一般の画像処理アプリケーション開発者であっても事例ベースで画像処理プログラムが開発できることを目標とする。

提案手法の貢献は、画像処理と数値演算が混在したプログラム自動生成法を、遺伝的プログラミングに新しい制約を課すことなく実現したことである。遺伝的プログラミング操作に対しては単一のデータ型として振る舞い、画像処理と数値演算は個体評価時にのみ現れるため、特別な操作や制約が不要である。また、ブロートのなどにより遺伝子サイズの増大が発生しても、個体の適応度評価では冗長ノードを除去し、必要な演算時間とメモリ容量を著しく抑えることができるため、多くの遺伝的プログラミングで導入されている個体サイズ制限が不要である。

1.3 本論文の構成

次章以降の本論文の構成は以下のとおりである。

第2章で、遺伝的プログラミングについて概要を述べる。まず、その源流である遺伝的アルゴリズムについて述べたあと、遺伝的プログラミングにおける違いについて述べる。そして、目的関数が複数ある場合にパレート最適解を探索する手法である多目的遺伝的アルゴリズムについて述べる。

第3章で、遺伝的プログラミングを応用した画像処理プログラム自動生成に関する従来研究と、課題について述べる。

第4章で、提案手法について述べる。基本となるアイデアである Image-Value ペアな

らびに Image-Value フィルタ，Image-Value ツリーについてそれぞれ定義を行い，自動生成フレームワークの設計と Image-Value ペアの導入方法について述べる．

第 5 章では，提案手法を，実験課題に適用して自動生成が行えることを確認する．花画像の背景除去プログラムの自動生成では，システム設計および適応度関数の設計について述べたのち，自動生成させて得られたプログラムについて構造の理解と従来手法との比較を行う．

第 6 章では，また，提案手法が数値データを取り扱うことができるという特長に着目し，画像のクラス分類プログラムの自動生成に適用する方法について述べる．また，クラス分類プログラムと背景除去プログラムをそれぞれ自動生成して組み合わせることで，全体としてより高性能な背景除去プログラムを作成する．

第 7 章では，提案手法を適用した遺伝的プログラミングにおける，個体サイズ増加の影響と課題，および Image-Value フィルタの拡張について議論を行う．

第 8 章では，以上を総括し，本論文における結論を述べる．また，提案手法の課題や今後の展望について述べる．

付録 A では，Image-Value ツリーの評価を実装する場合に，計算リソースを最適化する手法について述べる．

付録 B として，本論文で使用した Image-Value フィルタおよびその内部フィルタの一覧表を載せた．

第 2 章

遺伝的プログラミング

遺伝的プログラミング (Genetic Programming, GP) は、生物の進化過程における遺伝のメカニズムに着想を得た、プログラム自動生成のフレームワークの一つである。プログラムを一つの生物個体と見立て個体集団を作り、目標とする結果に近い良い個体（プログラム）の子孫がなるべく増えていくように世代交代を繰り返すうちに、集団全体が良くなっていくという仕組みで良いプログラムを探していく。

本章ではまず、遺伝的プログラミングの源流である遺伝的アルゴリズム (Genetic Algorithm, GA) について概説する。つづいて、遺伝的プログラミングの概要と遺伝的アルゴリズムとの違い、特に遺伝子構造および遺伝的プログラミングに特徴的な課題であるイントロンとブロートについて述べる。そして、複数の目的関数を同時に最適化する手法である多目的最適化について述べ、多目的遺伝的アルゴリズム (Multi-Objective Genetic Algorithm, MOGA) の代表的なアルゴリズムについて概説する。そして、多目的遺伝的プログラミングを画像処理へ応用する方法について述べ、有効性と課題を明らかにする。

2.1 遺伝的アルゴリズム

遺伝的アルゴリズムは、生物の進化過程における遺伝のメカニズムに着想を得た、パラメータ最適化フレームワークの一つである [30, 31]。

最適化したいパラメータをコード化し、一つの生物個体と見立て、個体集団を形成する。パラメータは一般に複数個のセットであり、初期個体集団は、各パラメータにランダムな数値が与えられる。個体の良さは、パラメータが目的に合致している程度を数値化した **適応度 (fitness, fitness value)** で測られる。そして、高い適応度の個体ほど生き残りやすいという原則の下で個体を選択し、交叉や突然変異などの遺伝的操作を用いて子個体を生成する。子個体は親の良さの一部、すなわちパラメータのコード化表現の一部を受け継いでいることが期待されるため、世代交代を繰り返すことで、集団の中に最適なパラ

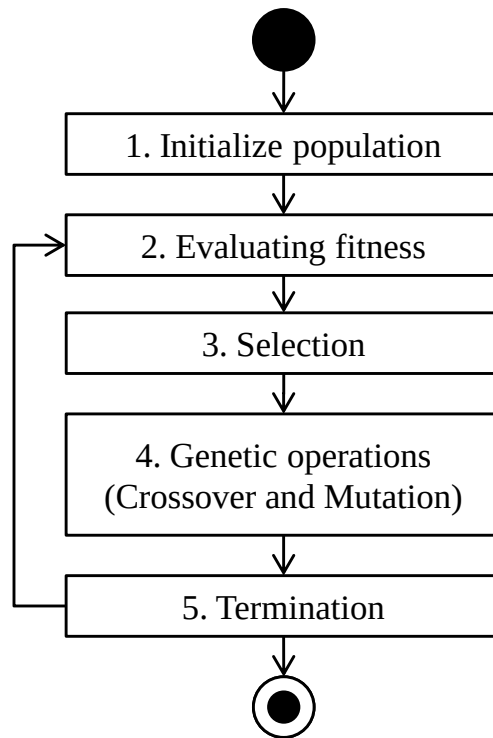


図 2.1: 遺伝的アルゴリズムの一般的な手順.

メータに近い個体が生成されていく。

適応度の計算は、個体の持つコードをパラメータに戻して実際に利用した場合の結果から行われる。たとえば、あるシステムの複数のパラメータを最適化しようとしている場合であれば、個体の持つパラメータを適用してシステムを動作させ、その結果が目標に近ければ高い適応度を、遠ければ低い適応度を与える。適応度を与える関数を、**適応度関数 (fitness function)** と呼び、問題ごとに設定される。なお、値が小さい場合に適応度が高いとする場合と、大きい場合に高いとする場合があり、選択オペレータのアルゴリズムに合わせる必要がある。

遺伝的アルゴリズムの基本的な手順を図 2.1 に示す。まず、(1) 初期集団を生成する。そして、(2) 各個体の適応度を計算し、(3) 適応度に応じて個体を選択する。(4) 選択された個体に対して、交叉や突然変異などの遺伝的操作を行い、次世代の個体群を形成する。(5) (2) から (4) を、終了条件を満たすまで繰り返す。世代交代を終了する条件としては、適応度が目標の値に到達したところで終了、既定の世代に達したら終了、などがよく利用される。

遺伝的アルゴリズムはフレームワークであるため、手順の一つ一つの具体的なアルゴリズムには様々なものがある。特に重要な選択と遺伝的操作を合わせたものを 世代交代モデルと呼び、Simple GA [32], Minimal Generation Gap (MGG) [33] など、様々な手

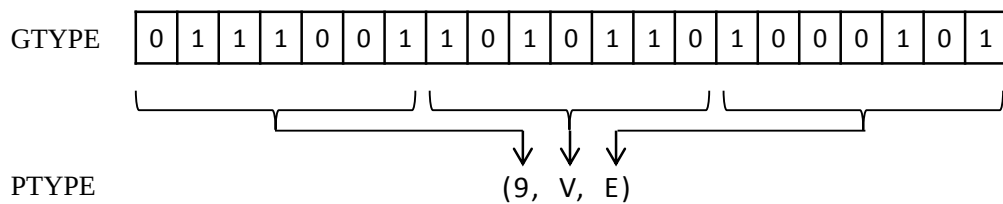


図 2.2: 遺伝的アルゴリズムの GTYPE, PTYPE の例.

法が提案されている。

以下，文献 [34] に基づいて遺伝的アルゴリズムの基本的な知識について述べる。

2.1.1 個体の遺伝子型と表現型

遺伝的アルゴリズムにおいて，解は**個体 (individual)** と呼ばれ，遺伝的操作が可能な**記号列 (string)** で表現される。個体は，遺伝子コードの表現方法である **GTYPE** (遺伝子型) と，実際に評価される形態にデコードされた表現である **PTYPE** (表現型) の二重構造で取り扱われる^{*1}。GTYPE は，生物における遺伝子型のアナロジーで，パラメータのセットをコード化したものである。遺伝的操作は GTYPE に対して行われる。PTYPE は，表現型あるいは発現型と呼ばれる生物における構造や特性の発現のアナロジーである。GTYPE を意味のある数値や文字などに変換した結果で，PTYPE によって適応度が計算される。最も基本的な遺伝的アルゴリズムでは，0 と 1 を用いた**バイナリ記号列 (binary string)** が，GTYPE として用いられることが多い。

記号列は，実際の生物では**染色体 (chromosome)** に対応し，個々の記号が遺伝子に対応する。記号列中の位置は**遺伝子座 (locus)** と呼ばれる。遺伝子座の違いによって対応するパラメータや寄与の度合いが決まっている。

図 2.2 に，GTYPE および PTYPE の例を示す。GTYPE は 21 文字のバイナリ記号列である。PTYPE は GTYPE を 7 ビットずつに区切り，ASCII コードとみなした 3 文字である。そして，この 3 文字が辞書に載っている単語に近いほど適応度が高い，などとする。この例では，アルファベット以外の文字になってしまうことがある。この場合は，そもそも評価ができないため，実行不可能解であり，**致死遺伝子 (lethal gene)** とも呼ばれる。

^{*1} 遺伝子型，表現型を表す用語としては **genotype**, **phenotype** という遺伝学の用語をそのまま用いることも多いが，本論文では遺伝的アルゴリズムあるいはプログラミングの文脈であることを区別するため，GTYPE, PTYPE を用いる [31]。

2.1.2 初期個体群の生成

初期個体群 (initial population) は、最も基本的な遺伝的アルゴリズムでは、0 と 1 を等確率で遺伝子座に割り当てることでバイナリ記号列を生成する。

一般に、初期個体群の選び方は、その後の探索の効率を決めるため、極めて重要である。初期個体群の多様性が少ない場合、局所解に陥る可能性が高くなる。問題に関する事前知識を利用することで、良好な初期個体群を生成する手法もある。探索空間を狭めていることになるので性能を大きく向上させることが期待されるが、大域解からはずれている場合には潜在的には探索性能に悪影響がある可能性もある。

2.1.3 選択

選択 (selection) とは、現在の個体群の中から次世代に残すべき個体を選び出す操作で、淘汰とも呼ばれる。ここで選ばれた個体群に対して交叉や突然変異が適用されて次世代の個体群を形成するため、遺伝的アルゴリズムの性能に大きく影響する最も重要な部分である。

原則としては、適応度が高いものほど選択されやすくして次世代に子孫を残すようにしたい。そこで、適応度に応じて確率的な選択を行う。特によく使われるものにルーレット方式とトーナメント方式がある。

ルーレット (roulette) 方式 は、集団全体の適応度の合計に対する、個体の適応度の比率に応じた確率で選ぶ方式である。 n 個の個体と、 i 番目の個体の適応度 f_i , $i = 1, \dots, n$ が与えられた場合、 i 番目の個体を選択される確率 p_i は、

$$p_i = \frac{f_i}{\sum_{i=1}^n f_i} \quad (2.1)$$

となる。

トーナメント (tournament) 方式 は、現世代の集団から S_t 個の個体をランダムに選び、そのうち最も良いものを次世代に残すというものである。 $S_t = 2$ の場合を、特にバイナリトーナメント方式と呼ぶ。

確率的に選ぶ方式では、現世代の最良個体が次世代に引き継がれるとは限らないため、集団の適応度が下がってしまうこともある。これを防ぐため、現世代の優良個体をそのまま無条件に残す**エリート戦略 (elitist strategy)** と呼ばれる戦略がある。通常は残すべき個数を決め、その他の個体は確率的な方式で選ばれる。

また、世代交代を経るうちに集団に含まれる個体が類似したものばかりになって**多様性**

(**diversity**) を失ってしまうことがある。このような状態になると、交叉による変化が起きにくくなり、その後の世代交代で新しい形質を生み出すことが難しくなってしまう。これに対処するため、類似している個体が多くなるにつれて適応度にペナルティを与える戦略があり、**シェアリング (sharing)** と呼ばれている。

2.1.4 遺伝的操作

選択された個体に対して、交叉や突然変異などの**遺伝的操作**を行って、現世代の形質を継承した次世代の個体を作る。

交叉 (crossover) は、GTYPE の一部を 2 つの親個体から受け継いで、親個体の特徴を継承した子個体を生成する操作である。交叉する場所の選び方によって、図 2.3 に示すような種類がある。一点交叉 (one-point crossover) は、交叉点が 1 か所の場合である。 n 点交叉 (n -point crossover) は、交叉点が n か所の場合。 $n = 1$ の場合が一点交叉である。一様交叉 (uniform crossover) は、GTYPE と同じ長さのランダムビット列を生成し、ビット列の値が 1 なら親 A から子 a、親 B から子 b へ、0 なら親 A から子 b、親 B から子 a へ、それぞれ受け継ぐ方式である。

突然変異 (mutation) は、GTYPE の一部をランダムに選び、値を入れ替える操作である。ビット列型の GTYPE の場合は、ビット反転突然変異が多く用いられる。図 2.4 に具体例を示す。

その他、逆位 (inversion) などの操作があり、GTYPE のコード方法によっては重要であるが、本論文では扱わない。

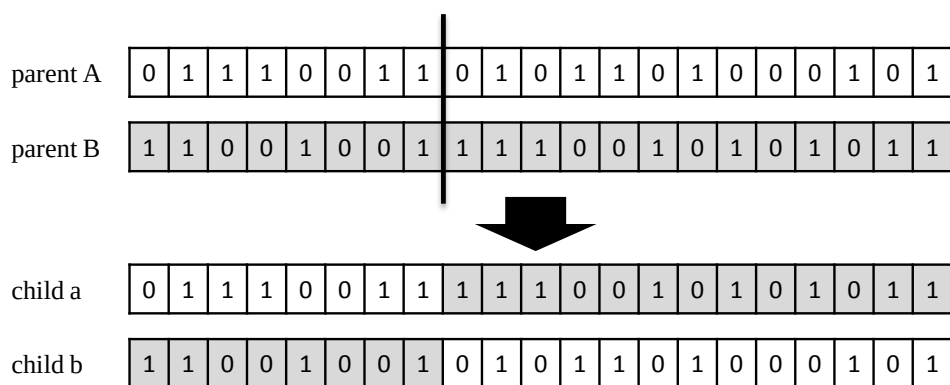
2.1.5 世代交代モデル

個体集団の世代交代では、多様性を維持しながら探索空間を適切に絞り込んでいくことができるよう、個体の選択や子の生成方法を選ぶことが重要である。

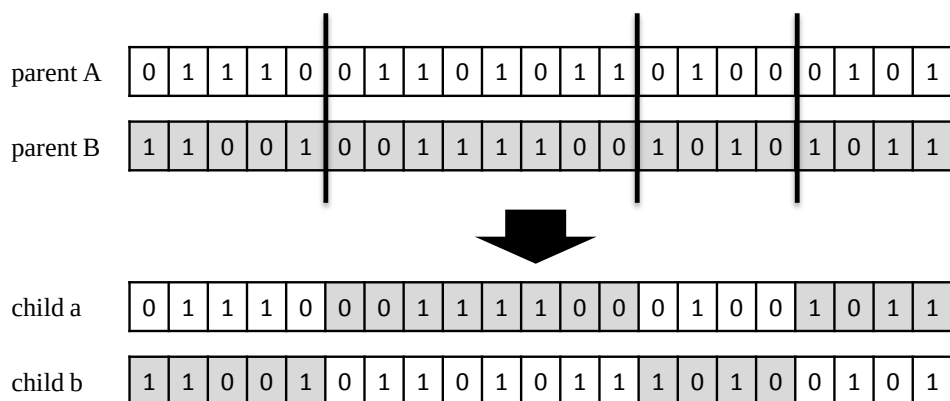
佐藤による定義 [33] では、世代交代モデルの一般的な枠組みは

1. 複製選択
2. 子の生成
3. 生存選択

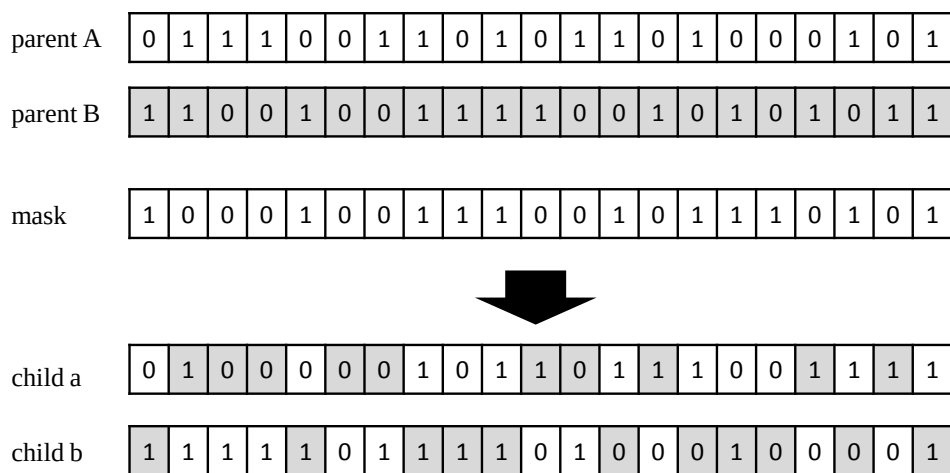
からなる。すなわち、複製選択 (selection for reproduction) と生存選択 (selection for survival) の 2 種類の選択操作が含まれている。複製選択は子を生成する親を選ぶための選択であり、生存選択は現世代とその子から次世代に生き残る個体を選ぶための選択である。子の生成では選択された親から交叉や突然変異によって新しい個体を生成し、その適



(a)



(b)



(c)

図 2.3: 遺伝的アルゴリズムの交叉オペレータ. (a) 1 点交叉. (b) n 点交叉. (c) 一様交叉.

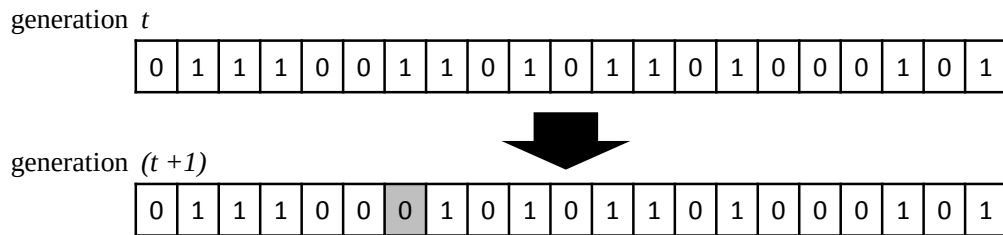


図 2.4: 遺伝的アルゴリズムの突然変異オペレータ。

応度を評価する。各個体の生成は遺伝子コードや交叉オペレータの設計で決定されるため、世代交代モデルと独立している。

既存の世代交代モデルのうち、代表的なものを下記に示す。

■ Simple GA (SGA) [32]

複製選択 ルーレット選択によって、集団から個体を復元抽出する。

生存選択 無条件で、子集団を次世代集団とする。

最も基本的なモデルであり、遺伝的アルゴリズムの基本的性質を研究するためにも用いられる。ルーレット選択は、探索初期に適応度の突出した個体が存在するとその個体が過剰に選択されやすくなってしまい、探索が進まないうちにその個体に収束してしまう**初期収束** (early convergence) と呼ばれる現象が起こりやすい。また、探索が収束期にさしかかり個体間の適応度の差が小さくなってくると、選択圧がかかりにくくなりそれ以上の探索が進まない問題もある。

■ Elitist Recombination (ER) [35]

複製選択 適応度を無視して、集団から個体をランダムに非復元抽出する。

生存選択 親2個体とその子2個体から、適応度の高い2個体を次世代に残す。

適応度の値を確率のような形で直接利用せず、順序を決めるためにだけ利用するランキング選択を用いている。親も次世代に残る機会が与えられており、このような戦略を**世代交代の連続化**と呼ぶ [33]。生存選択が家族（親とその子）のみに局所化されており、特定の個体が集団内に急速に広まることを抑制している。

■ Minimal Generation Gap (MGG) [33]

複製選択 適応度を無視して、集団から個体をランダムに非復元抽出する。

生存選択 親2個体とその子2個体から、適応度の最も高い1個体とルーレット選択に

よって選ばれた1個体を次世代に残す。

複製選択にランダム抽出を用いることで初期収束の回避が期待される。また、生存選択では最良個体に加えてルーレット選択の確率的要素を含めることで、多様性の維持に最大限配慮が行われている。特に問題に適した世代交代モデルが明確でない場合には、第一候補として検討するのが適切である [31]。

■Non-Dominated Sorting GA II (NSGA-II) [36]

複製選択 バイナリトーナメントによって、集団から個体を復元抽出する。

生存選択 親世代と、同数生成された子世代を合わせた全体に対し、**高速非優越ソート** (Fast Non-Dominated Sorting) によってランク付けされたグループを作成する。その後、上位ランクのグループを一括で集団個数まで選んでいく。端数が出るグループは、crowded comparison operator による混雑度により、類似している個体が少ないものを優先的に選ぶ。

多目的遺伝的アルゴリズムの代表的手法である。詳細は2.3節で述べるが、多目的遺伝的アルゴリズムでは、通常トレードオフ関係にある別々の目的関数から計算された複数の適応度を同時に考慮して探索を行うため、必ずしも最適な解を一意に定めることができない。そのため、ランク付けは集合に対してなされる。多様性の維持は得られた解集合に対して、多くの解が集中している部分には低い適応度を与えるシェアリング手法を用いている。NGSA-IIのシェアリングは、パラメータがないことが特長である。

2.2 遺伝的プログラミング

遺伝的プログラミング (Genetic Programming, GP) は、遺伝的アルゴリズムの GTYPE を木構造に拡張し、プログラム生成に使えるようにしたものである [26]。GTYPE の拡張に伴って遺伝的操作が変更されているが、それ以外は遺伝的アルゴリズムと共通のフレームワークを用いることができる。以下、異なる部分を中心に述べる。

2.2.1 個体表現

遺伝的プログラミングでは、プログラムを木構造で表現したものを GTYPE として用いる。木構造とは、ノード (node, 頂点) とエッジ (edge, 辺) からなる循環がないグラフ構造である。出力のないノードをルート (root), 入力のないノードを終端ノード (terminal node), それ以外を非終端ノード (non-terminal node) とそれぞれ呼ぶ。また、エッジでつながれた2つのノードの関係について、ルートに近い方を親 (parent), もう一方を子

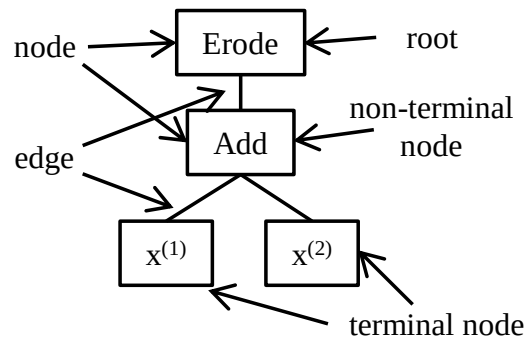


図 2.5: 木構造.

$$(\text{Erode}(\text{Add}(x^{(1)})(x^{(2)})))$$

図 2.6: S 式の例. 図 2.5 を S 式で表したもの.

(child) と呼ぶ. 図 2.5 に木構造の例を示す. プログラムとしての側面から見た場合, 非終端ノードは関数 (function) に, 終端ノードは記号 (symbol) あるいは定数 (constant) に, それぞれ対応する.

遺伝的プログラミングの研究ではよく Lisp (List Processor) 言語が用いられることから, 木構造を S 式によって表すことが慣習となっている [26, 31]. S 式とは, 木構造の並列関係と親子関係を, 括弧の並列性と入れ子で表現したものである. 図 2.6 に, 図 2.5 を S 式で表現した例を示す. 本論文でも, S 式を用いて表現する. S 式には子が終端ノードの場合に括弧を省略する簡略表現があるが, 本論文では使用しない.

なお, 特に他のプログラミング言語で実装する場合などにおいては, 計算機内部における木構造の表現として S 式を扱う必要はなく, 何らかのグラフ構造ライブラリなどを用いてもよい.

2.2.2 遺伝的操作

遺伝的プログラミングの交叉は, 部分木の交換として行われる. 例えば, 一点交叉の場合, 2つの親個体それぞれの部分木をランダムに選択して, それらを交換したものを子個体の遺伝子とする. 図 2.7 に例を示す.

突然変異は, 一つの親個体の遺伝子から部分木をランダムに選び, 新しく生成した別の

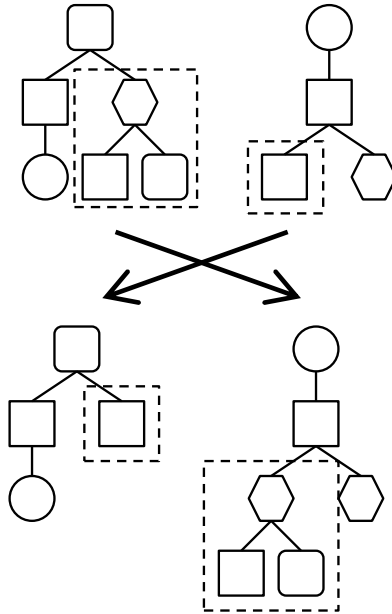


図 2.7: 遺伝的プログラミングの交叉オペレータ.

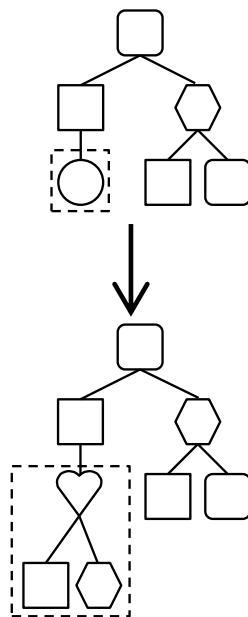


図 2.8: 遺伝的プログラミングの突然変異オペレータ.

木に置き換える操作となる。図 2.8 に例を示す。

2.2.3 適応度の評価

遺伝的アルゴリズム同様、必要に応じて GTYPE から PTYPE に変換し、問題設定ごとに定められた適応度関数を用いて、個体の良さを評価する。遺伝的プログラミングにおいて、個体とはプログラムのことであるので、このプログラムを実行した結果が求めるものと近ければ適応度を高くする。

遺伝的アルゴリズムでは PTYPE から GTYPE にコード化する方法が非常に重要であったが、遺伝的プログラミングでは、木構造による GTYPE はそのままプログラムとしての表現でもあり、遺伝子全体の木構造と演算子の配置が適応度に強い影響力を持つ一方で遺伝子座ごとの意味は曖昧であるため、PTYPE への変換意義が明確ではない。特に、遺伝的プログラミングの初期の研究では、木構造の並列関係と親子関係を括弧の並列性と入れ子で表す S 式と呼ばれる記法でプログラミングを行う Lisp 言語が用いられたため、GTYPE と PTYPE は同一視されていた。

しかしながら、評価の高速化を目的として木構造中のノードに対応する機械語命令を処理する順に並べたり [37]、木以外の構造を持つ遺伝子表現を用いたりするなど [38, 39]、PTYPE への変換を積極的に用いる手法もある。

2.2.4 初期世代の生成

初期世代の生成は、遺伝的アルゴリズム同様重要である。遺伝的アルゴリズムでは GTYPE の遺伝子座によって PTYPE への影響度合いが異なるが、木構造 GTYPE の場合は遺伝子座ごとの意味は明確ではないため、木のサイズと構造および含まれるオペレータの種類に多様性を持たせる点についてのみ注意すれば良い。そのため、課題ごとに応じた配慮をせず、Koza による **Ramped Half-and-Half** と呼ばれる方式が多く使われている [26]。これは、定めたノード数を含む個体をランダムに生成する **Grow** と、すべての枝分かれで定めた深さまでノードを含む個体をランダムに生成する **Full** を、世代人口の半分ずつ生成するものである。

2.2.5 ブロートとイントロン

遺伝的アルゴリズムにはない、遺伝的プログラミングに特徴的な問題として、世代を経るにしたがって遺伝子のサイズが徐々に巨大化してしまう **ブロート (bloat)** と呼ばれる現象がある。ブロートが発生すると、以下のような問題が発生するため好ましくないものと考えられている [34]。

- 適応度評価に要する時間が増大する.
- 集団の記憶に必要なメモリ量が多く必要となる.
- プログラムの可読性を大きく損なう.
- ノード数が多い個体では過学習が生じている場合が多く、汎化誤差が大きくなる.

ブロートが起こる背景を詳細に検討した研究には removal bias 理論などがあり [40], **イントロン (intron)** と呼ばれる冗長部分が影響しているという考え方が一般的である. イントロンとは, 遺伝的プログラミングの GTYPE 中に木構造の一部として存在しているものの, 実行結果に影響を与えない冗長なノードのことである. 例えば, $0 + 1 \times x$ というプログラムが生成された場合, $0 +$ と, $1 \times$ はイントロンである.

遺伝的プログラミングでは, 個体の適応度がノード一つ一つの種類だけでなく全体的な構造まで含めて決定されるため, 交叉や突然変異による遺伝子の破壊効果が大きい. しかし, イントロン部分であれば適応度に影響がないため, 遺伝的操作によって適応度が低下した個体よりも, 個体が生き残りやすくなる. このような交叉を繰り返すことで, 個体サイズが巨大化してしまうと考えられる. この効果はブロートを誘発していると考えられている一方, 遺伝的オペレータによる破壊に対する緩衝材として働くことで特徴を継承しやすくすることで, 解の多様性に好影響があるという指摘もある [41].

ブロートを抑制する最も典型的な方法は, 固定のパラメータを与えて, 最大サイズを定めることである. 別の方法として, サイズに応じて適応度にペナルティを与える方法があるが, 性能に悪影響があることが報告されている [26, 42, 43]. また, 木のサイズ分布のヒストグラム形状をコントロールする方法も提案されている [44]. いずれの方法もサイズの基準が必要だが, 遺伝子サイズがどの程度であれば良い解が得られるかは事前にはわからないため, 最大サイズを小さくし過ぎれば良い解を見つけることができず, 大きくし過ぎれば抑制効果が低くなる.

遺伝的操作を改変して, サイズを抑える方法も提唱されている [42]. 50-150% Fair Mutation は, 突然変異で置き換える木のサイズを, 取り除かれる部分木のサイズに対して 50-150% にする. また, Subtree Fair Mutation は, 取り除かれるサブツリーのサイズに対して, 同じサイズのサブツリーを, 同じ木から探して交換する. 突然変異ではサイズが増加しないことを保証する. Tarpeian Method [45] は, サイズを超えた個体のうちある確率で適応度を強制的に低く設定することにより生き残りにくくし, 集団サイズを抑える.

絶対的な基準値を設けることを避けるため, 本来の適応度とは独立した目的関数とし, 多目的最適化を行う方法がある. de Jong は, 多目的遺伝的プログラミングを使い, 目的関数にサイズと多様性を加える方法を提案している [46]. しかし, 世代交代の初期にサイズ要素が強く働き過ぎて, 集団をサイズの小さい個体が占めてしまうと, 交叉で発生する

バリエーションが小さくなり、進化速度が大幅に遅くなってしまう場合がある。多目的最適化については第 2.3 節で述べる。

2.2.6 種々の遺伝子表現

遺伝的プログラミングの個体表現には木構造が最も多く用いられているが、木構造以外の構造を持つ方式も提案されている。例えば、非循環有効グラフ (Directed Acyclic Graph, DAG) を一次元配列の GTYPE にエンコーディングする Linear Genetic Programming や Cartesian Genetic Programming といった手法も存在する。

Linear Genetic Programming (LGP) [47, 38] は、木あるいは DAG 構造のプログラムをバイナリ記号または整数値の一次元配列 (string) にコード化したものを GTYPE として用いる方式である。

Banzhaf らによる方式 [47] では、逆ポーランド記法で記述されたプログラムをビット列に変換したものを GTYPE に用いて、ビット列の任意の位置で交叉や突然変異を行うことができた。

Brameier らによる方式 [38] では、整数の固定長一次元配列に、演算、入力レジスタ番号 1、入力レジスタ番号 2、出力レジスタ番号の 4 つ組を単位として複数個並べた構造になっている。交叉は組の区切りの位置で、突然変異は個々の整数値の単位で行う。LGP としては後者の方式が主流となっている。後者の方式では、レジスタ番号を介してグラフ構造が表現されるため、レジスタ個数分の出力を持つ DAG 構造となっている。レジスタに書き込まれた値が、他の入力として使われないうまま別の値で上書きされることがあり、使われなかった値を計算した演算はイントロンとなる。イントロンが存在することにより、遺伝子表現としては固定長配列でありながら、実質的には可変長のプログラムを表現できる。

遺伝的アルゴリズムの遺伝的オペレータが利用できること、必要なメモリ容量が少ない、逐次的に実行できる、などの要因で計算に必要なリソースが標準的な遺伝的プログラミングに比べて少ない特徴がある。

Cartesian Genetic Programming (CGP) [39, 48] も LGP と類似の一次元配列型 GTYPE を持つが、レジスタ番号を介したグラフ表現ではなく、どの命令の出力を入力とするかを与える命令番号を使って構造を表現する点に特徴がある。CGP でも、LGP 同様のメカニズムでイントロンが発生する。評価の際には、GTYPE から逐次実行が可能な命令列で表現された PTYPE に変換し、この際にイントロンの除去も行う。このイントロン除去の効果により、ブロートの問題が起こりにくいという報告もある [48]。

2.3 多目的遺伝的アルゴリズム

遺伝的アルゴリズムや遺伝的プログラミングでは解の良さを適応度で表現するが、解きたい課題に対するプログラムの設計目標が、一つの目的関数で測ることができるとは限らない。さらに、複数の目的関数があった場合、ある目的関数を改善すると他の目的関数が悪化するような、互いにトレードオフの関係にあることもある。線形結合のような手法を用いて単一の適応度にする手法の適用も考えられるが、各目的関数が単位が違うなどの理由で比較可能ではないこともあり、一般に適切な設計は困難である。

このような場合は、多目的最適化を利用する [49, 50, 51]。一般に、複数の目的関数を最小化する多目的最適化問題は、決定変数ベクトル $\mathbf{x} = (x_1, x_2, \dots, x_n)$ に対して次の様に定式化される [34]。

$$\begin{aligned} \min \{ & f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_q(\mathbf{x}) \} \\ \text{s.t. } & g_i(\mathbf{x}) \leq 0, \quad i = 1, 2, \dots, m \\ & x_j \geq 0, \quad j = 1, 2, \dots, n \end{aligned} \quad (2.2)$$

ここで、 $f_k(\mathbf{x})$ ($k = 1, \dots, q$) は q 個の目的関数、 $g_i(\mathbf{x})$ は m 個の制約条件の関数である。

各目的関数がトレードオフの関係にある場合、最適解を一意に定めることができない。そのため、パレート最適解 (Pareto optimal solution) を求める。解がパレート最適であるとは、ある解がその他の解にすべての目的関数で劣ることがないことで、非劣解ともいう。パレート最適解は複数存在しうるため、その集合をパレートフロントと呼ぶ。

遺伝的アルゴリズムに多目的最適化を取り入れた多目的遺伝的アルゴリズム (Multi-objective Genetic Algorithm, MOGA) が活発に研究されている [36, 52, 53, 54, 55, 56]。

初期の多目的遺伝的アルゴリズムは、複数の目的関数で計算したスコアを要素とするベクトル適応度を単純に用いたものであったが、その後パレート解の概念とエリート保存を積極的に利用する世代交代戦略が提案されて、安定した性能を得られるようになった。代表的な多目的遺伝的アルゴリズムについていくつかの文献で世代ごとに分類されている [57, 58]。文献 [34] による分類を以下に引用する。

第1世代：非パレートのアプローチ

VEGA (vector evaluated GA) [52]

第2世代：パレートのランキング+多様性維持

MOGA (multi-objective GA) [53], NSGA (non-dominated sorting GA) [54]

第3世代A：第2世代+エリート保存

SPEA2 (strength Pareto EA II) [55], NSGA-II (non-dominated sorting GA

II) [36]

第3世代B：多目的関数の重み付け＋エリート保存

RWGA (random-weight GA), AWGA (adaptive-weight GA), i-awGA (interactive awGA)

第3世代の各手法は優れた探索性能を示すが、目的関数が4以上になるとほとんど解が見つけれないという問題が明らかになっている。これに対して、多目的 (many-objective) に適用可能なアルゴリズムの研究が活発化しており、NSGA-III [59] などが提案されている。

第 3 章

画像処理プログラムの自動生成と課題

遺伝的プログラミングによる画像処理の自動生成がこれまで試みられている。画像処理プログラムは一般に画像データだけでなく数値データを取り扱う必要があるが、複数のデータ型を持つプログラムの生成は、サブルーチンの接続関係に矛盾が生じないように、特別な遺伝的操作を用いて回避する必要があった。また、自動生成に要する時間を抑える目的の他、世代交代を繰り返すとプログラムサイズが指数関数的に増大する遺伝的プログラミングに特有のブロートと呼ばれる現象があり、個体サイズの制限を加えることが多い。しかし、適切なプログラムサイズは事前にはわからないため、制約は避けたい。

本章では、画像処理プログラムの自動生成における従来研究を概観し、その課題として画像処理と数値演算の混在、およびサイズ制限について述べる。

3.1 関連研究

青木、長尾らは、平滑化・エッジ抽出・セグメンテーション・合成のフィルタを木構造状に組み合わせた画像変換を遺伝的プログラミングで自動構築する手法として、Automatic Construction of Tree-structural Image Transformations (ACTIT)[23] を提案した。

ACTIT では、フィルタが使用する数値パラメータはフィルタ内部で適応的に計算していたが、藤嶋らによる Parameter-Tunable ACTIT (PT-ACTIT) は、遺伝的プログラミングと遺伝的アルゴリズムの両方を用いて、フィルタを組み合わせたプログラム構造(木構造)と、フィルタ内部で用いる数値パラメータを交互に最適化する手法を提案した [60]。

白川らは、Genetic Image Network (GIN)[61] で、個体表現を ACTIT の木構造から閉路を持つネットワーク構造に拡張した。単一の木構造では表現できない多出力や、ループによって何度も同じフィルタを実行することが可能であるため生成されたプログラムが

コンパクトになる利点がある。ループ回数を制御するためにステップ数というパラメータが導入されている。

さらに、GIN を拡張した Genetic Image Network for Image Classification (GIN-IC) [62] では、画像分類問題に適用できるよう、特徴量抽出や数値演算等の画像以外の入出力を持つ演算に対応した [62, 63]. 画像変換ノード、特徴量抽出ノード、演算ノードの順序関係に制約を設けることで、遺伝的操作に伴う演算ノード同士の無効な接続を回避している。個体のノード数は固定だが、接続の有無で可変長プログラムを表現できる。

Krawiec らは、Linear GP を応用し、画像認識向け特徴量計算プログラムを自動構築する手法を提案している [64]. Linear GP では演算とレジスタ番号の組を並べた固定長次元配列としてプログラムを表現することで、遺伝的アルゴリズムの遺伝的操作で遺伝的プログラミングを擬似的に実行する。これには、実行速度上の利点と、各演算のデータやり取りにレジスタ番号を使用して型を直接扱わないことで、演算ノード同士の無効な接続の問題を回避できる利点がある。各レジスタは、画像データ、マスクデータ、数値データの組になっており、各演算は画像または数値の演算を行うことが可能である。

Mixed Type Cartesian Genetic Programming (MT-CGP) [65] は、データおよび演算に実行時多態性を持たせることで、型の混在に対応する方法を提案している。MT-CGP におけるプログラム表現では、データは、実数スカラー型、実数値を要素とするベクトル型のいずれかをとることができる。ベクトル型は次元によっても区別される。演算子は、受け取ったデータ型によって実際の演算内容を決める。例えば、加算演算子の2つの引数が両方とも実数スカラー型の場合はスカラーの、両方同じ次元を持つ実数ベクトル型の場合はベクトルの加算を実行する。しかし、2つの引数の型がスカラー型とベクトル型、あるいは異なる次元を持つベクトル型のように異なる場合は、スカラー型あるいは次元の小さいベクトルに合わせて「キャスト」を行ってから演算を行う。このため、構造やノードの種類が同一の遺伝子であっても、プログラムとしては入力データの型に応じた多態性があることになる。

Montana らは、画像ではないが、スカラー、ベクトル、行列などが混在したプログラムを生成できる、Strongly Typed Genetic Programming (STGP) を提案している [66]. ノードの入出力型の表を持ち、異なる型を接続しないよう遺伝的操作を行う。例えば、交叉では同じ型を持つ場所のみが交叉点の候補となる。

3.2 課題

遺伝的プログラミングによる自動生成を用いた方法は、画像処理アプリケーションの開発生産性を向上させることが期待されるが、課題もある。

3.2.1 画像処理と数値演算の混在

遺伝的プログラミングでは、プログラムを多種のサブルーチンを木構造に連結したものとして考える。プログラムの自動生成は、処理手順の種類や順序をランダム的に組み替えてことで行われるが、画像を入力して数値を出力する処理のように異なるデータ型を扱うことが難しい。これは前後の処理のデータ受け渡しに矛盾が生じると実行できないプログラムが生成されてしまうためである。

しかし、画像処理プログラムには画像フィルタの閾値、条件分岐など数値パラメータが必要である。また、入力画像から抽出した特徴量を別のフィルタのパラメータとして用いることができれば、画像条件に対して適応的に処理内容を変化させるような頑健なプログラムの設計にもつながるため、画像と数値の2種類のデータが混在するプログラムを生成できることが望まれる。従来手法では、画像と数値のように複数のデータ型がある場合は、遺伝的操作に特別な設計や制約を加える必要があった。

ルールが比較的単純な遺伝的プログラミングでデータ型の混在が難しい点は、エキスパートシステムが種々のデータ構造をカバーする代わりに文法が複雑化していた点と対照的であるが、遺伝的プログラミングの明快さを維持したままデータ型混在を解決することが望まれる。

3.2.2 プログラムサイズ制限

遺伝的プログラミングの個体表現は可変長であるため、プログラムサイズの大きさと探索実行時間とのトレードオフを検討する必要がある。特に、世代交代を繰り返すと世代の平均プログラムサイズが指数関数的に増加するブロー現象への対策が重要である。従来手法のほとんどでサイズ制限や増大に対するペナルティなどの対策を行っている。プログラムサイズはプログラムの表現力に強い影響があるが、適切なプログラムサイズは事前にはわからないため、設定が難しいパラメータの一つである。ブロー対策を行いつつ、プログラムサイズに対する制約を大幅に緩和することが望ましい。

また、画像処理特有の問題もある。通常遺伝的プログラミングで取り扱う問題は、数式の合成など数値データと数学の基本演算子を組み合わせた者が多い。しかし、画像処理の場合、扱うデータサイズと計算量がけた違いに大きいため、個体サイズの影響がより深刻である。

表 3.1: プログラム自動生成手法の比較. v は多いほど良い, x は少ないほど良い, *1: PT-ACTIT は GA で数値パラメータの最適化を行えるが, 一般の数値演算は行えない.

	ACTIT [23]	PT-ACTIT [60]	GIN-IC [62]	LGP [64]	STGP [66]	MT-CGP [65]	Goal
Image Processing	v	v	v	v	-	-	v
Numerical Operations	-	*1	v	v	v	v	v
GP Syntax Constraint	-	x	x	-	x	-	-
Size Limitation	-	-	x	x	-	x	-

3.3 本研究の目的

表 3.1 に従来手法が生成できるプログラムの種類と制約を示す. 従来手法では, 画像演算と数値演算のどちらも生成できる場合であっても, (1) 遺伝的操作, (2) 遺伝子の長さ, のいずれかに制約がある. 制約 (1) は, ノードの入出力データ型が正しく接続されることを保証するためのものである. 制約 (2) は, ブロート対策や実行時間の短縮を目的としているが, GIN-IC, LGP, MT-CGP は一次元配列型の GTYPE を用いていることから, イントロンによって PTYPE サイズ可変を実現しているが, 最大サイズは事前に決められる必要がある.

本研究では, 遺伝的プログラミングを用いて, 画像データと数値データが混在したプログラムの自動生成を, 特殊な遺伝的操作やプログラムサイズの制約を課すことなく実現する手法の提案を目的とする.

第 4 章

提案手法

画像処理と数値演算が混在したプログラムを遺伝的プログラミングで生成するための汎用的枠組みとして、Image-Value ペアと呼ぶ概念の導入を提案する。Image-Value ペアとは画像と数値をペアにしたもので、Image-Value ペアを入出力とする演算（Image-Value フィルタ）をノードとする木構造を Image-Value ツリーと呼ぶ。この Image-Value ツリーを GTYPE として用いることで、特別な遺伝的操作を用いなくても、画像処理と数値演算が混在した画像処理プログラムを遺伝的プログラミングで取り扱うことができる。

本章では、Image-Value ペア、Image-Value フィルタ、ならびに Image-Value ツリーについて定義を行い、Image-Value ツリーを GTYPE として遺伝的プログラミングへ導入する方法について述べる。次に、Image-Value ツリーを冗長なノードを除去しながら PTYPE に変換するアルゴリズムについて述べる。

4.1 Image-Value ペア

Image-Value ペアは画像と数値を同時に扱うためのキーとなる。本節では、データ型である Image-Value ペア、Image-Value ペアを処理するオペレータである Image-Value フィルタ、Image-Value フィルタを木構造に組み合わせたプログラムである Image-Value ツリー、それぞれについて述べる。

4.1.1 Image-Value ペア

画像 I と数値 V をペアにしたものを、**Image-Value ペア (Image-Value pair)**、あるいは IV ペア (IV pair) と呼ぶこととする。Image-Value ペア $\mathbf{x}$ は次の様に定式化される。

$$\mathbf{x} = (I, V). \quad (4.1)$$



図 4.1: IV フィルタの例. (a) 1 入力 IV フィルタ. (b) 2 入力 IV フィルタ.

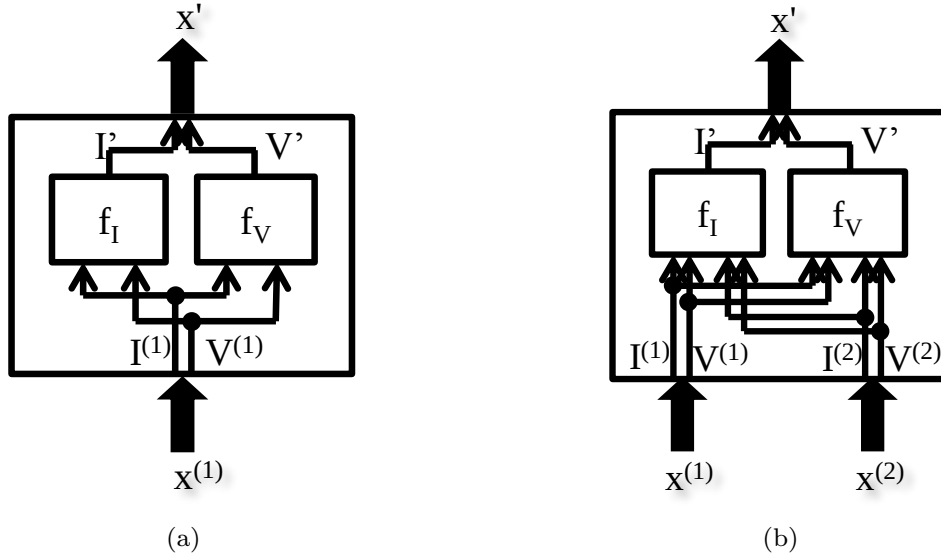


図 4.2: IV フィルタの内部構造の例. (a) 1 入力 IV フィルタ. (b) 2 入力 IV フィルタ.

I は, 課題ごとに設定される固定のサイズを持つ 2 次元画像である. 本論文では, 8 ビットグレースケールの 2 次元画像を用いるが, 一貫していれば実数値ピクセルを使用することもできる. V は, $0.0 \sim 1.0$ の実数値とする.

4.1.2 Image-Value フィルタ

Image-Value ペアに対する演算として, **Image-Value フィルタ (Image-Value filter)**, あるいは IV フィルタ (IV filter) を, 次のように定義する.

$$\mathbf{x}' = f(\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)}). \quad (4.2)$$

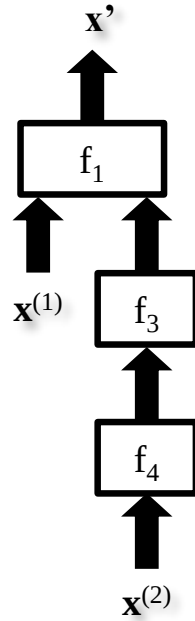


図 4.3: Image-Value ツリーの概念図.

Image-Value フィルタ f は, Image-Value ペア $\mathbf{x}^{(i)} = (I^{(i)}, V^{(i)})$ を入力として何らかの演算を施し, Image-Value ペア $\mathbf{x}' = (I', V')$ を出力する. 右肩の添え字 (i) は i 番目の入力データであることを表す. 入力の数 n は自然数で, 出力の数は 1 である. 図 4.1 に, 1 入力 Image-Value フィルタおよび 2 入力 Image-Value フィルタの模式図を示す.

Image-Value フィルタ f の内部は, 画像成分の演算を行う f_I と, 数値成分の演算を行う f_V のペアからなっている. f_I , f_V のいずれも入力されたすべての Image-Value ペアの画像成分および数値成分の関数である. すなわち,

$$I' = f_I(I^{(1)}, \dots, I^{(n)}, V^{(1)}, \dots, V^{(n)}) \quad (4.3)$$

$$V' = f_V(I^{(1)}, \dots, I^{(n)}, V^{(1)}, \dots, V^{(n)}) \quad (4.4)$$

と表される. 図 4.2 に例として, 1 入力 Image-Value フィルタ, その内部構造, 2 入力 Image-Value フィルタ, その内部構造を示す. なお, f_I および f_V は, 必ずしも入力された $I^{(i)}$, $V^{(i)}$ すべてを演算に使わなくてもよい.

4.1.3 Image-Value ツリー

Image-Value フィルタをノードとする木構造を, **Image-Value ツリー (Image-Value tree)**, あるいは IV ツリー (IV tree) と定義する. Image-Value ツリー F は, 終端ノードの Image-Value ペア $\mathbf{x}^{(i)} = (I^{(i)}, V^{(i)})$ を入力として, ルートノードから

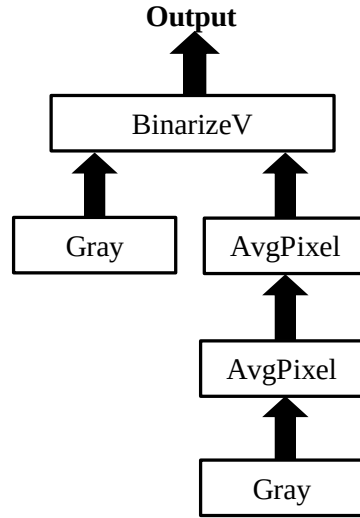


図 4.4: Image-Value ツリーの例.

Image-Value ペア $\mathbf{x}' = (I', V')$ を出力するプログラムの表現である. すなわち,

$$\mathbf{x}' = F(\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)}) \quad (4.5)$$

入力の数 n は自然数で, 出力の数は 1 である. F_I, F_V は, Image-Value の出力の I 成分, V 成分をそれぞれ表すものとする. 図 4.3 に Image-Value ツリーの概念図を示す.

4.2 遺伝的プログラミングへの導入方法

Image-Value ツリーを遺伝的プログラミングに導入するためには, Image-Value ツリーを GTYPE として用い, 個体の適応度評価では内部フィルタに対して計算を行う. まず, GTYPE として用いる点については, Image-Value ツリーの表現に特殊な点はなく一般の木構造や S 式で表現できる. 次に, 個体の適応度を計算する場合について, Image-Value ツリーの評価は, ノードである Image-Value フィルタの内部フィルタの木構造に対して行われる. 例えば, 図 4.4 のような Image-Value ツリーがあった場合, 内部フィルタは図 4.5 のようになる. ノード間の接続は, Image-Value ペアではなく, I 成分および V 成分を直接連結する. この内部フィルタツリーについて, ノードである内部フィルタ f_I と f_V を, 葉から根に向かって順に処理していけば良い.

しかし, 図 4.5 を見ると自明のように, 内部フィルタツリーには, 遺伝子中に含まれているが, 演算結果が使われない冗長ノード (イントロン) が含まれている.

Image-Value ツリーで現れるイントロンは, Andre ら [67], Brameier ら [38] の分類に基づき, 下記の 3 類型に分類できる.

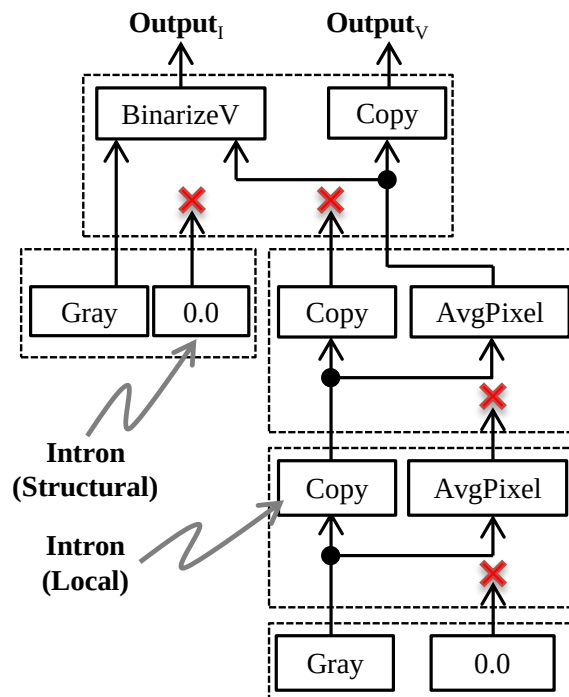


図 4.5: Image-Value ツリーの内部構造.

局所的イントロン (local introns) 入力そのまま出力するノード.

構造的イントロン (structural introns) 出力が演算に利用されないノード.

意味的イントロン (semantical introns) 演算結果に変化がないノード.

局所的イントロンは、入力をそのまま出力するノードで、図 4.5 の例ではすべての Copy ノードが該当する。構造的イントロンは、出力が演算に利用されないノードで、フィルタが入力を一部しか利用しない場合に発生する。図 4.5 の例では 2 つある 0.0 ノードと、AvgPixel ノードのうち一つ、Copy のノードうち一つが該当する。意味的イントロンは、例えば $1 \times x$ の $1 \times$ の部分が該当する。

イントロンは適応度には影響しないが、遺伝的オペレータによる破壊に対する緩衝材として働くことで特徴を継承しやすくすることで、解の多様性に好影響があるという指摘がある [41]。一方、演算時間やメモリ消費に悪影響を及ぼすため、適応度を評価する際は取り除く場合がある [38]。

局所的イントロンは、Copy ノードを見つけたら前後のフィルタを直接つなぐようエッジ接続を変えることで、取り除くことができる。構造的イントロンも、出力がどこにも接続されていないノードを見つけたら、そのノードをルートとする部分木全体を削除することで、機械的に取り除くことができる。次のアルゴリズムで、内部ツリー F の局所的イントロンと構造的イントロンを同時に取り除くことができる。

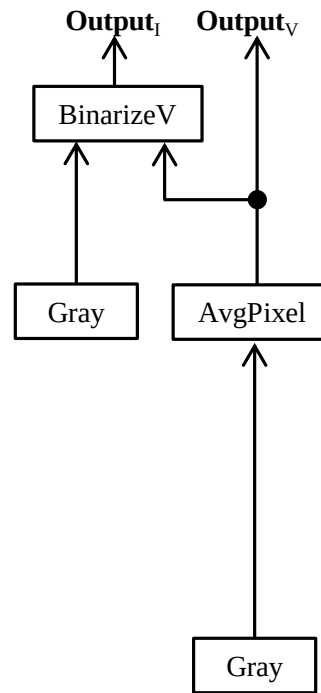


図 4.6: Image-Value ツリーの PTYPE.

1. 空のツリー G を用意する.
2. F のルートノードを, 現在のノードに設定する.
3. 現在のノードがCopyであった場合は, 現在のノードの子ノードを, 現在のノードに設定する.
4. 現在のノードを, G にコピーする.
5. 現在のノードを, 現在のノードに接続された子ノードに設定する.
6. 3 へ戻る.

局所的イントロンと構造的イントロンがグラフ構造から容易に判定できるのに対し, 意味的イントロンであることを判定するにはノードの種類と構造が表す意味が定義されている必要があり, またその定義が集められた辞書と個体に存在する組み合わせを突き合わせる必要があるため取り除くのは容易ではない. 局所的イントロンと構造的イントロンを削除することで十分な効果が得られることから, 意味的イントロンはそのまま演算した.

図 4.5 から, 局所的イントロンおよび構造的イントロンを取り除いた内部フィルタツリーを, 図 4.6 に示す. 10 個あったノードが, 4 個に削減されている.

この冗長性が削減されたツリーに含まれるすべての内部フィルタを, 葉から根に向かって順に実行していけばフィルタ出力が得られる. そして, フィルタ出力と目標画像から生適応度 (raw fitness) を計算し, 全教師画像の生適応度を平均するなどの標準化を行って,

最終的な適応度を得る.

4.3 提案手法の課題解決アプローチ

提案手法を用いると, PTYPE では「画像」と「数値」の 2 種類のデータを扱えるにもかかわらず, GTYPE では「ペア」型一種類を扱えばよいため, 遺伝的プログラミングで特殊な遺伝的操作を必要とせず, 多くの遺伝的プログラミングフレームワークで画像処理と数値演算が混在したプログラムの自動生成が可能となる.

また, GTYPE から PTYPE へ変換する際に構造的イントロンと局所的イントロンを効果的に取り除くため, GTYPE はブロートしても, 画像処理演算量を決める PTYPE サイズの増大は大幅に緩和される.

なお, 遺伝的プログラミングの探索性能向上や, 生成したプログラムの適応度向上に対して, Image-Value ペアの導入が直接寄与するわけではない.

第 5 章

背景除去プログラムの自動生成

本章では，Image-Value ツリーを用いた遺伝的プログラミングを，背景除去プログラムの自動生成に適用する方法について述べる．背景除去プログラムのシステム設計，遺伝的プログラミングパラメータの設計，ならびに適応度関数の設計および利用するフィルタの選択について述べたのち，自動生成させて得られたシーケンスの構造の理解と従来手法との比較を行う．

5.1 課題の定義

花のカラー写真を入力すると，背景部分が除去された画像を出力するフィルタを自動生成させる課題に適用し，実験を行った．屋外の自然環境下で撮影された花の画像を原画像とし，グレースケール変換した原画像の花部分だけを残して黒く塗りつぶした画像を目標画像とした．図 5.1 に原画像および目標画像を示す．

5.2 背景除去プログラムの自動生成

提案手法を，背景除去プログラムの自動生成に適用するための，背景除去システムの設計，自動生成で使用する適応度関数と遺伝的プログラミングパラメータの設計，および自動生成の結果についてそれぞれ述べる．

5.2.1 システム設計

背景除去プログラムは，入力された画像の注目部分を残してそれ以外は黒く塗りつぶした画像を出力する，画像フィルタで構成されている．Image-Value ペア $\mathbf{x}$ を入力すると，出力画像 $\mathbf{x}'$ を出力する．図 5.2 にブロック図を示す．

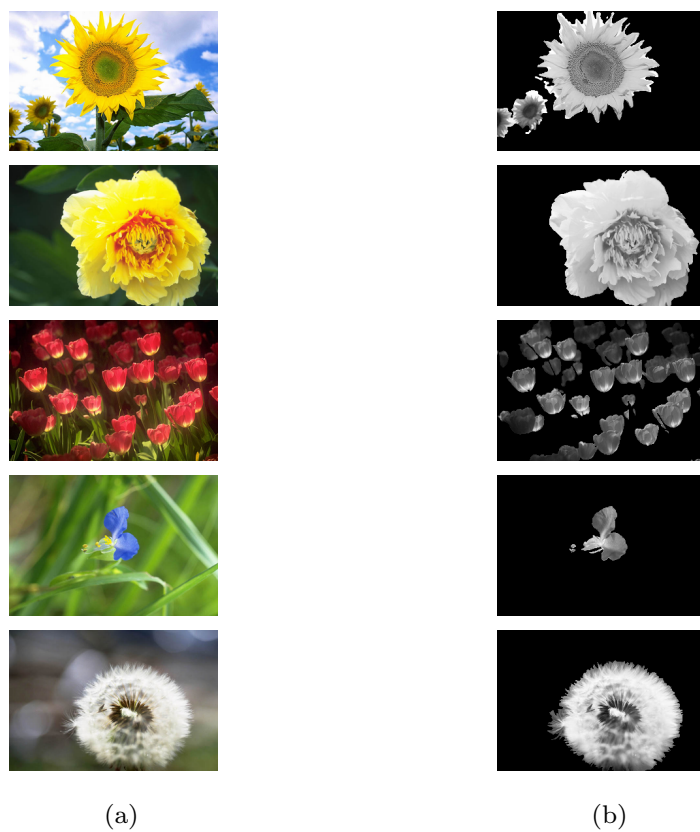


図 5.1: 背景除去課題の教師データセット. (a) 原画像, (b) 目標画像.

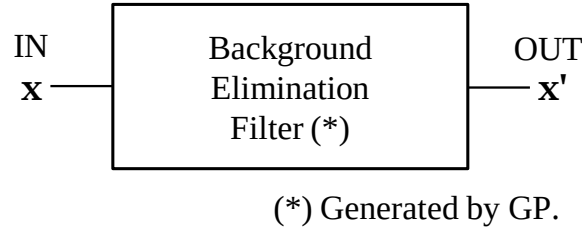


図 5.2: 背景除去フィルタのブロック図.

5.2.2 適応度関数の設計

適応度関数は，トレーニングセットに含まれる全サンプルについてフィルタ出力と目標画像の Peak Signal-Noise Ratio (PSNR) を求め，それらの平均値および最小値を同時に最適化する 2 目的適応度関数とした (5.1).

$$Fitness_V = (ScoreA, ScoreM). \quad (5.1)$$

$$ScoreA = \frac{1}{\varepsilon + \frac{1}{K} \sum_{k=1}^K WPSNR_{\mathcal{W}}^{(k)}} + \frac{1}{\varepsilon + \frac{1}{K} \sum_{k=1}^K WPSNR_{\overline{\mathcal{W}}}^{(k)}} \quad (5.2)$$

$$ScoreM = \frac{1}{\varepsilon + \min_{k=1 \dots K} WPSNR_{\mathcal{W}}^{(k)}} + \frac{1}{\varepsilon + \min_{k=1 \dots K} WPSNR_{\overline{\mathcal{W}}}^{(k)}} \quad (5.3)$$

$ScoreA$ は，トレーニングサンプルをフィルタで処理させた結果の出力画像について，目標画像との PSNR を計算し，全サンプルで平均したものである (式 (5.2)). ただし，重み付き PSNR (Weighted PSNR, WPSNR) (式 (5.4)) を用いて，花部分と背景部分に分けてそれぞれ計算し，加算することとした．これは，花部分と背景部分の面積の比率がサンプルによって異なるため，画像全体に対して PSNR を計算すると花部分と背景部分の寄与が異なってしまうためである． $ScoreM$ も同様であるが，平均の代わりに最小値を用いた (式 (5.3)). K は教師データセットサイズ， ε は 0 による除算を回避するための微小な定数である．本論文では $\varepsilon = 10^{-6}$ を用いた．なお，NSGA-II は最小値を求めるフレームワークであるため，逆数をとってから足し合わせた．式 (5.1) の各成分の最小値は $2 - \frac{2\varepsilon}{1+\varepsilon} \approx 2$ となる．

$ScoreA$ はフィルタの平均的な性能向上を目的とし， $ScoreM$ は偏差を小さくすることを目的としている．これは，生成したプログラムが，平均的な性能は優れていても，あるサンプルでは極端に良いスコアを示したり，あるサンプルでは極端に悪いスコアを示した

りすることを避けるためである。これにより、花部分の保存と背景の除去を両立し、画像に含まれる花の数や色、背景の条件によらず安定した性能を発揮するフィルタを生成する目標を、目的関数に反映している。

重み付き PSNR の計算は式 (5.4) で行い、最大値の制限および 0 除算の回避を行っている。

$$WPSNR_{\mathcal{W}}^{(k)} = \begin{cases} 1 & WMSE_{\mathcal{W}}^{(k)} > C \text{ のとき} \\ 0.2 \log \frac{255}{\sqrt{WMSE_{\mathcal{W}}^{(k)}}} & 0 < WMSE_{\mathcal{W}}^{(k)} \leq C \text{ のとき} \\ 0 & WMSE_{\mathcal{W}}^{(k)} = 0 \text{ のとき} \end{cases} \quad (5.4)$$

C は $WPSNR$ を飽和させるための画像サイズによらない定数で、 $C = 6.5025 \times 10^{-6}$ である。重み付き平均二乗誤差 $WMSE_{\mathcal{W}}^{(k)}$ は式 (5.5) で計算される。

$$WMSE_{\mathcal{W}}^{(k)} = \frac{1}{mn} \sum_{i=1}^m \sum_{j=1}^n [\mathcal{I}_{i,j}^{(k)} - \mathcal{T}_{i,j}^{(k)}]^2 \mathcal{W}_{i,j}^{(k)} \quad (5.5)$$

$\{\mathcal{I}_{i,j}^{(k)}\}$ は出力画像、 $\{\mathcal{T}_{i,j}^{(k)}\}$ は目標画像、 $\{\mathcal{W}_{i,j}^{(k)}\}$ は $\{\mathcal{I}_{i,j}^{(k)}\}$ と同じサイズのマスク画像で、花部分は 1、背景部分は 0 のいずれかの値をとる。また、 $\{\overline{\mathcal{W}_{i,j}^{(k)}}\} = \{1 - \mathcal{W}_{i,j}^{(k)}\}$ である。 m, n はそれぞれ画像の X 方向、Y 方向のサイズである。また、原画像をそのまま出力するような画像フィルタが生成されるのを防ぐため、フィルタ出力が全画素で 0 または 255、または Gray0 と完全に一致した場合は、式 (5.5) を特別な値 0 とし、低い適応度を与えるようにした。

5.2.3 パラメータ

遺伝的プログラミングのパラメータは表 5.1 のように設定した。終端ノードとして、表 5.2 に示した 5 種類の Image-Value ペアを用いた。入力画像として、 I 成分は図 5.3 に示した原画像の輝度成分（グレースケール）、赤成分、緑成分、または青成分を、 V 成分は 0 を、それぞれ持つような Image-Value ペアを与えた。これに加えて、 I 成分は全ピクセルが 0、 V 成分も 0 であるような Const_0 を与え、定数 0 の代用として使用する。非終端ノードとして、表 B.1 に示した Image-Value フィルタを使用した。教師データセットサイズ $K = 5$ 、画像サイズ $m, n = 80, 53$ とした。

遺伝的プログラミングパラメータは、個体数 $M = 100$ 、5000 世代の世代交代を終了条件とした。交叉率は 0.6、突然変異確率は 0.0333333 とした。

表 5.1: 背景除去フィルタ生成課題の設計.

Objective:	Find a filter sequence that removes background area from a given RGB-color flower image.
Terminal set:	See Table 5.2
Function set:	See Table B.1
Fitness cases:	5 pairs of an input image and a target image, containing a variety of the color of and the number of flowers.
Raw fitness:	See Equation (5.4).
Standardized fitness:	See Equation (5.2) and (5.3).
Parameters:	Population size $M = 100$, Maximum generation $G = 5000$, Crossover ratio $r_{crossover} = 0.6$, Mutation ratio $r_{mutation} = 0.0333333$,
Success predicate:	None. Evolve until the G generations have been processed.

表 5.2: 背景除去フィルタ自動生成課題で使用する終端ノード.

Name	f_I	f_V
Gray0	Intensity of $\mathcal{I}$	0
Red0	Red component of $\mathcal{I}$	0
Green0	Green component of $\mathcal{I}$	0
Blue0	Blue component of $\mathcal{I}$	0
Const_0	0	0

$\mathcal{I}$ is one of training data.

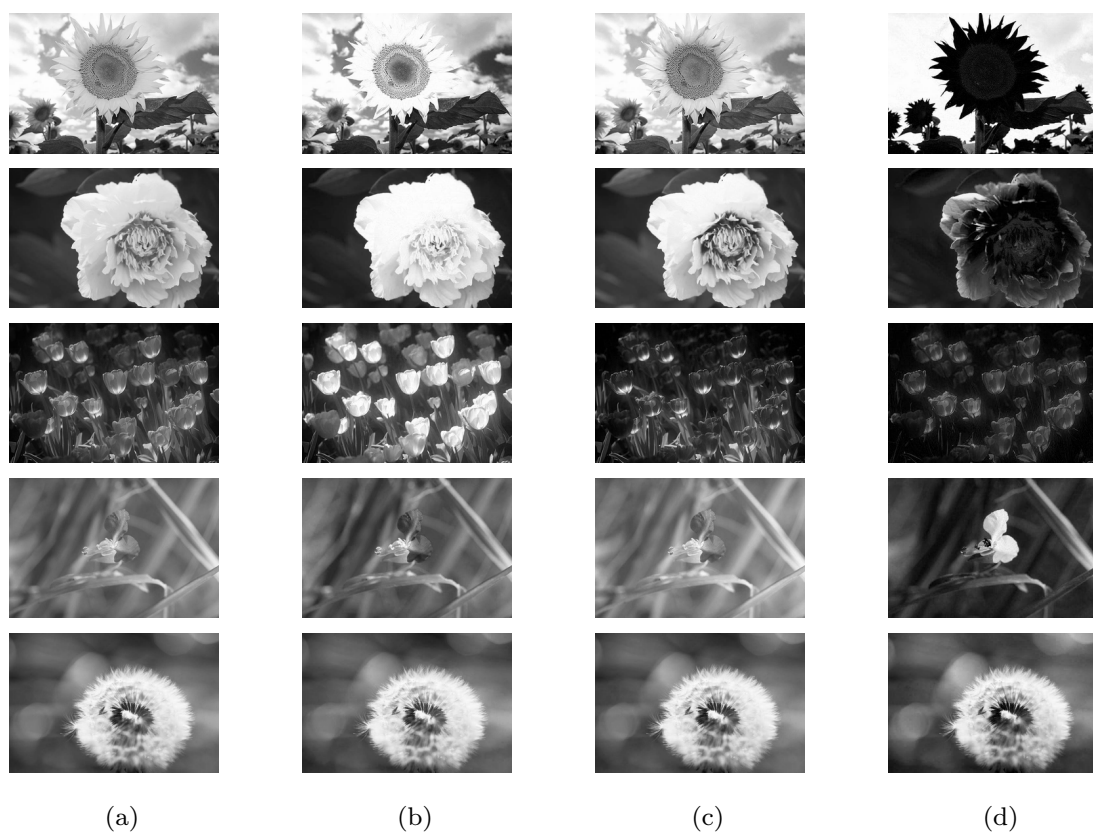


図 5.3: 終端ノードとして与えた画像. (a) グレースケール, (b) R 成分, (c) G 成分, (d) B 成分.

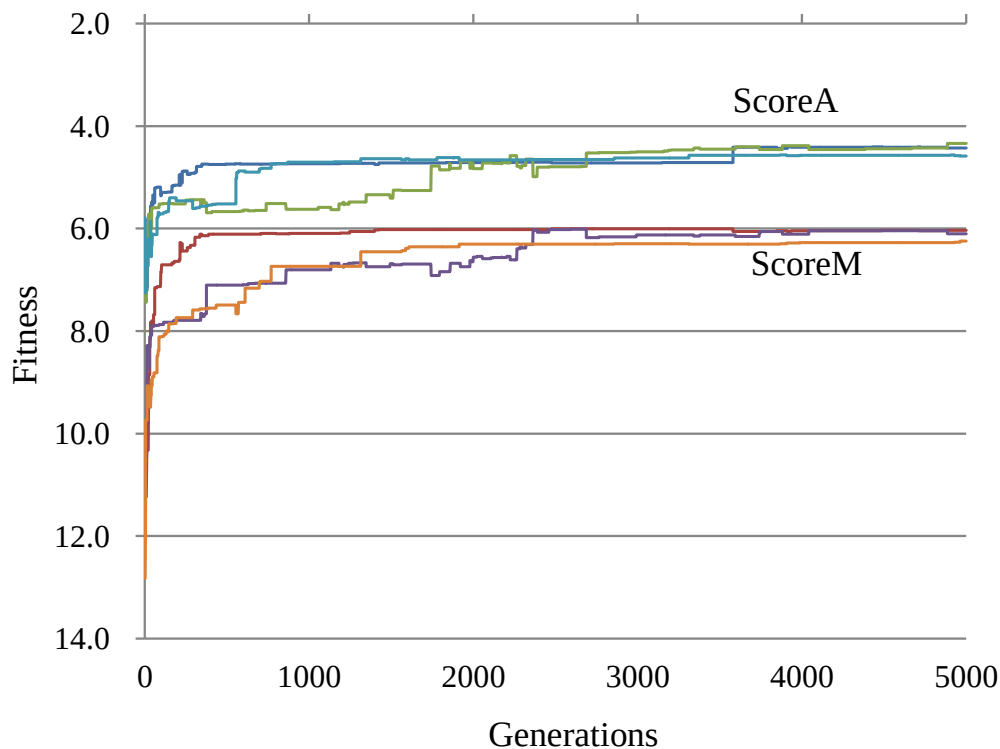


図 5.4: 最良個体の適応度推移.

5.2.4 適用結果

初期乱数シードの異なる 3 回の試行をした. 図 5.4 は各世代の適応度の推移を示す. ランダムに生成された初期世代は非常に適応度が低い (数値は大きい) が, いずれの試行でもすぐに急激に改善しその後緩やかな変化になっていく. 試行ごとに推移は異なるが, およそ 3000 世代まで進むと, *ScoreA* は約 4.5 (約 44dB に相当), *ScoreM* は約 6.1 (同 32dB) と, ほぼ同じ値に収束している. PSNR が 35~40dB を超えると画質の劣化がほぼわからなくなると言われており [68], 生成されたプログラムが良好な性能を持っていることがわかる. また, 3 回の試行で同様の結果が得られていることは, 本手法の再現性があることを示している.

生成されたフィルタが妥当なものであることを確認するため, 1 つの試行の第 5000 世代のパレートフロントから最良個体を選び, 解析を行った. 最良個体の選び方は, 適応度を並べたベクトルと各目的関数の理論的最小値を並べたベクトルの L2 ノルムが最も小さくなるものとした [69]. 選び出された個体の内部構造を展開して得られた内部 Image フィルタツリーを, 図 5.5 に示す. わかりやすさのため, ノードの名称は対応する Image-Value

```

(AlgebraicProd(1)(AddV(2)(AlgebraicSum(Subtract(Add(Dilate3x3(BoundedProd(Negate
(Median3x3(AlgebraicProd(Blue0)(Add(Red0)(ProjectionX(LowPass(Gray0)(VAlgebrai
cSum(MinPixel(Red0))(Const_0)))))))(Red0))(Blue0))(Diff(ProjectionX(Multiply
V(Negate(AlgebraicProd(Add(Median3x3(Dilate3x3(AlgebraicSum(ProjectionX(Negate
(Negate(ProjectionX(LogicalProd(Median3x3(Sharpen3x3(SobelY3x3(LogicalProd(Blu
e0)(Green0)))))(Red0)))))(Blue0))))(Red0))(Binarize(Subtract(Gray0)(Blue0)))
)(MinPixel(LogicalSum(Median3x3(BinarizeV(BinarizeInv(SobelY3x3(Erode3x3(Red0)
))))(VLogicalProd(Select(VLogicalSum(VLogicalProd(VAlgebraicProd(Const_0)(Const
_0))(Const_0))(Const_0))(Const_0))(Const_0))(Blue0)))))(Blue0))(AlgebraicPr
od(Smooth3x3(Select(HighPass(AddV(4)(Diff(5)(LogicalProd(Sharpen3x3(ThresholdV(N
egate(ModAvgV(BinarizeInv(BinarizeInv(Blue0)))(VSubtract(VAlgebraicSum(Const_0
)(VAdd(MinPixel(HighPass(BinarizeInvV(Add(MultiplyV(AddV(LogicalSum(Select(Pro
jectionY(Add(Red0)(Gray0)))(Red0)(Const_0)(Const_0))(Blue0))(Const_0))(Const_0
))(Red0))(Const_0))(Const_0))(Const_0))(VAlgebraicProd(VDiff(Const_0)(Const_
0))(Const_0)))))(MinPixel(SubtractV(AddV(ThresholdV(SobelY3x3(SobelX3x3(Subtra
ctV(Diff(Gray0)(BinarizeInvV(MultiplyV(MultiplyV(HighPass(Gray0)(MinPixel(AddV
(HighPass(Blue0)(Const_0))(Const_0)))))(Const_0))(Const_0))(Const_0))(Const_0
))))(Const_0))(AvgPixel(Subtract(Smooth3x3(Select(ProjectionX(BinarizeInv(Sharp
en3x3(Add(Dilate3x3(Green0))(Gray0)))))(Select(Red0)(Gray0)(Const_0)(Const_0)
)(VLogicalProd(Const_0)(Const_0))(Select(Const_0)(VAlgebraicSum(VSubtract(Const
_0)(VDiff(Select(Const_0)(Const_0))(Const_0))(Const_0))(Const_0)(Const_0))))(
Blue0))))(Const_0)))))(Blue0))(Red0))(AvgPixel(6)(BinarizeInvV(AddV(AlgebraicSu
m(Diff(AlgebraicProd(Subtract(Negate(MultiplyV(Blue0)(VAlgebraicSum(Const_0)(C
onst_0)))))(Select(BinarizeV(AlgebraicProd(Red0)(Blue0))(Const_0))(Green0)(Cons
t_0)(Const_0))(Median3x3(LogicalSum(Sobel3x3(BinarizeInv(SubtractV(Red0)(AvgP
ixel(Select(Median3x3(ThresholdV(Blue0)(Const_0)))(Gray0)(Const_0)(Const_0))))
))(Gray0)))))(Blue0))(Green0))(VNegate(MaxPixel(Red0)))))(VAlgebraicProd(AvgPixe
l(Select(Sobel3x3(Sharpen3x3(Green0)))(ProjectionY(Binarize(Red0)))(Const_0)(C
onst_0))(VAdd(Select(VLogicalSum(MaxPixel(Select(Sharpen3x3(Blue0))(Subtract(
Sharpen3x3(AlgebraicProd(Red0)(LowPass(BinarizeV(Sharpen3x3(ThresholdInvV(Sobe
l3x3(Gray0))(Const_0))(Const_0))(Const_0)))))(Green0))(Const_0)(Const_0))(Con
st_0))(Const_0))(VDiff(VDiff(Select(MinPixel(Gray0))(Const_0))(Const_0))(MaxPi
xel(SobelX3x3(Select(Red0)(BinarizeV(SubtractV(Median3x3(Sobel3x3(Gray0)))(Con
st_0))(Select(AvgPixel(Red0))(Const_0))(Const_0)(Const_0))))))))) (VAdd(Const
_0)(Const_0))(Blue0)(VAlgebraicProd(VLogicalSum(Const_0)(Const_0))(Const_0))(
MinPixel(BinarizeV(BoundedProd(SubtractV(Erode3x3(MultiplyV(ThresholdV(Sharpen
3x3(Select(Erode3x3(SobelX3x3(LogicalProd(BoundedProd(Blue0)(BinarizeV(Add(Sub
tractV(Red0)(Const_0))(Blue0))(Const_0))(Green0)))))(Red0)(VNegate(Const_0))(C
onst_0))(VAlgebraicSum(Const_0)(Const_0))(Const_0))(AvgPixel(AlgebraicProd(
Dilate3x3(BinarizeInvV(Dilate3x3(Green0))(VAlgebraicProd(VNegate(VAdd(Const_0)
(Const_0))(Const_0))))(ProjectionX(Sharpen3x3(LogicalSum(Red0)(SubtractV(Medi
an3x3(Green0))(VNegate(MinPixel(FillV(VLogicalSum(Const_0)(VNegate(VAdd(Const_
0)(Const_0))))))))) (Gray0))(Const_0))))(Add(ProjectionX(Binarize(Subtrac
tV(BinarizeInvV(LogicalSum(Laplacian3x3(BinarizeV(HighPass(LogicalProd(Select(
Green0)(Gray0)(Const_0)(Const_0))(Blue0))(Const_0))(Const_0))(Gray0))(Const_0
))(AvgPixel(AddV(LowPass(ProjectionY(SobelX3x3(Green0)))(Const_0))(Const_0))))
))(SubtractV(Negate(ModAvgV(BinarizeInvV(FillV(Const_0))(Const_0))(VAlgebraicP
rod(VLogicalSum(MaxPixel(Sharpen3x3(Median3x3(BoundedProd(BinarizeInv(SobelX3x
3(Add(Subtract(Select(Green0)(Gray0)(Const_0)(Const_0))(Gray0))(Gray0))))(Red0
)))))(Const_0))(Const_0))))(Const_0))))(Const_0))(Gray0(3)))

```

図 5.5: 獲得したプログラムの PTYPE. (1)~(6) は図 5.6 の番号に対応.

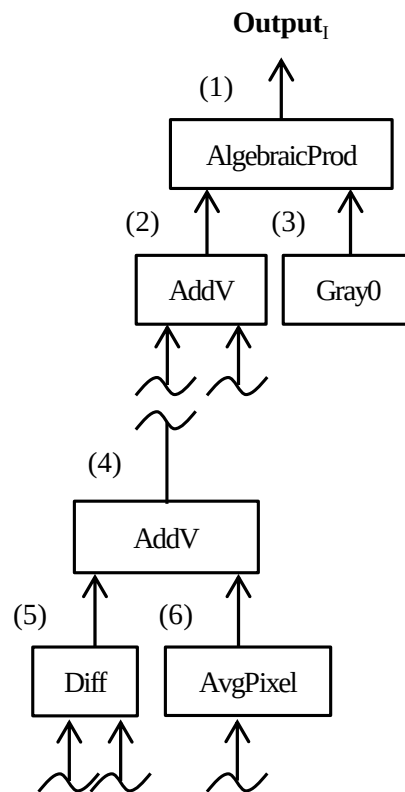


図 5.6: 獲得したプログラムの概略.

フィルタの名称に置き換えた。また、イントロンは除去されている。このツリーの各ノードにおける I 成分と V 成分の変化を調べたところ、図 5.5 に下線で示した 6 ヶ所で大きな変化が見られた。この 6 ヶ所のみを抜き出したツリーを、図 5.6 に示す。また、図 5.5 で示したツリーにトレーニングサンプルを入力した場合の、番号 (1)～(6) に示す位置での結果を図 5.7 に示す。最終出力 (1) は、(2) をマスクとして用いて (3) の原画像から AlgebraicProd により注目部分を抽出した結果となっている。(2) において、花が 1 つだけのサンプル A,B,D,E ではマスク後に残る白い部分は花の形状と一致しているの対し、花の小さい花が全体にあるサンプル C だけはほぼ全体を抽出するマスクになっている。このマスク画像の作り分けは、(2) の原型となる (4) の生成時に行われている。(4) は、注目部分がやや強調された画像 (5) に対して、(6) の数値成分を輝度に加算して生成されている。(6) は画像輝度の平均値を数値成分に与える AvgPixel 演算の結果であり、画像 C の画像全体に注目部分が広がっている特徴を数値として抽出し、効果的に利用していることがわかる。なお、別の試行の最良個体でも (6) の部分に MaxPixel が使われている場合があったが、共通の構造を持っていた。MaxPixel は画像輝度の最大値を与える演算であるため AvgPixel よりも強調効果が強く出るが、AvgPixel の場合と同様の効果が得られると考えられる。なお、各試行の第 5000 世代の個体はすべてルートに AlgebraicProd を持つ

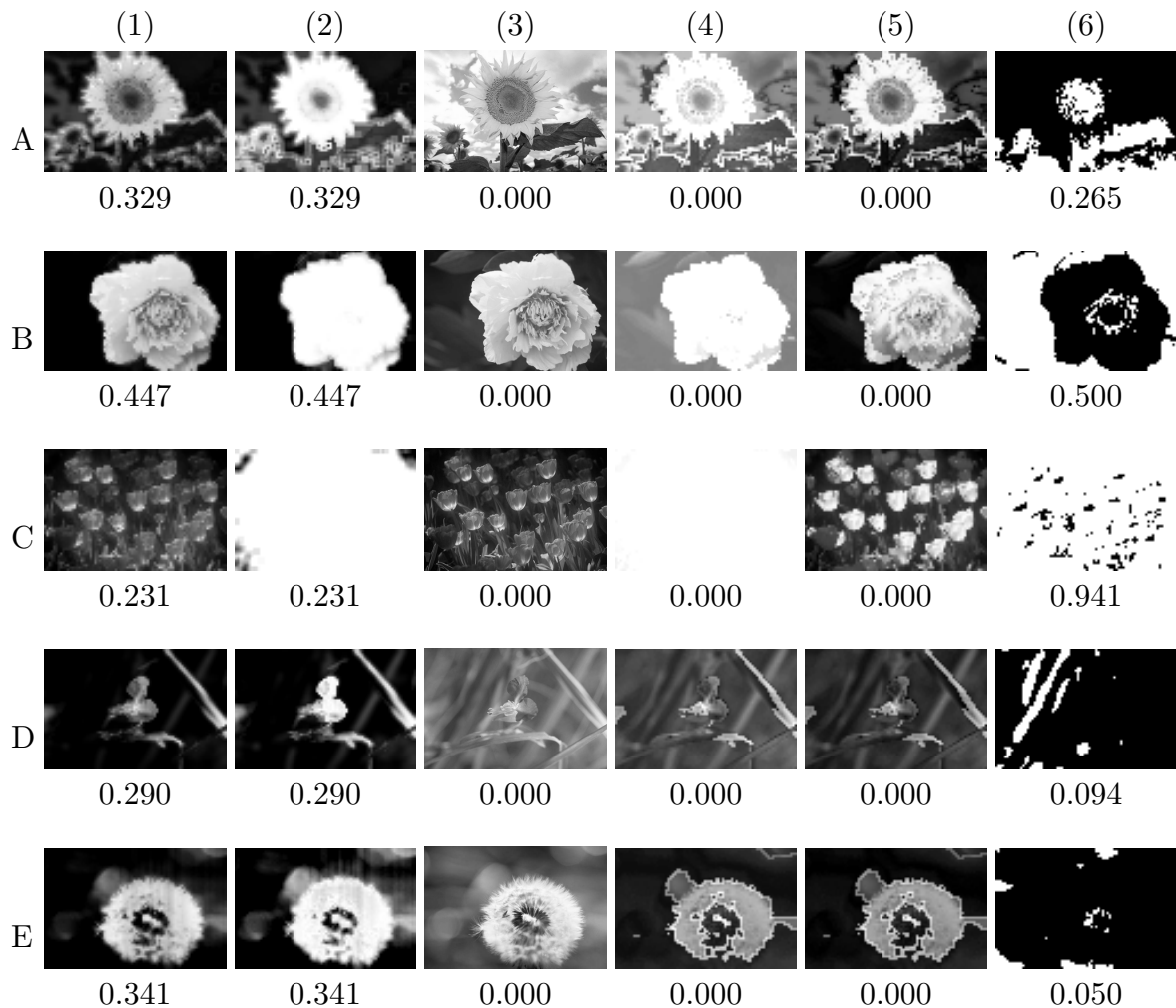


図 5.7: 獲得したフィルタを教師データに適用した結果. 番号 (1)~(6) の出力.

構造になっていた. これは, この構造が本課題に効果的であることを示している一方, 遺伝子の多様性が失われた状態になっているとも考えられる. 多様性が失われると, 教師画像に対して過学習した状態になったり, 解空間中を局所的にしか探索できずより良い解を見つけられなくなったりする可能性がある.

個体に含まれる平均演算数 (f_I と f_V の合計, 平均 IV フィルタ数の 2 倍) およびイントロンを除いた平均演算数の推移を図 5.8 に示す. 世代が進むにつれて Image-Value ツリーのノード数は 8000~9000 程度まで劇的な増加をしているのは, 適応度の向上に伴って大きな個体サイズまで探索範囲を広げて行っているためであると読み取れる. 一方, 実際に演算されるフィルタの数は 400 程度と極めて低く抑えられている.

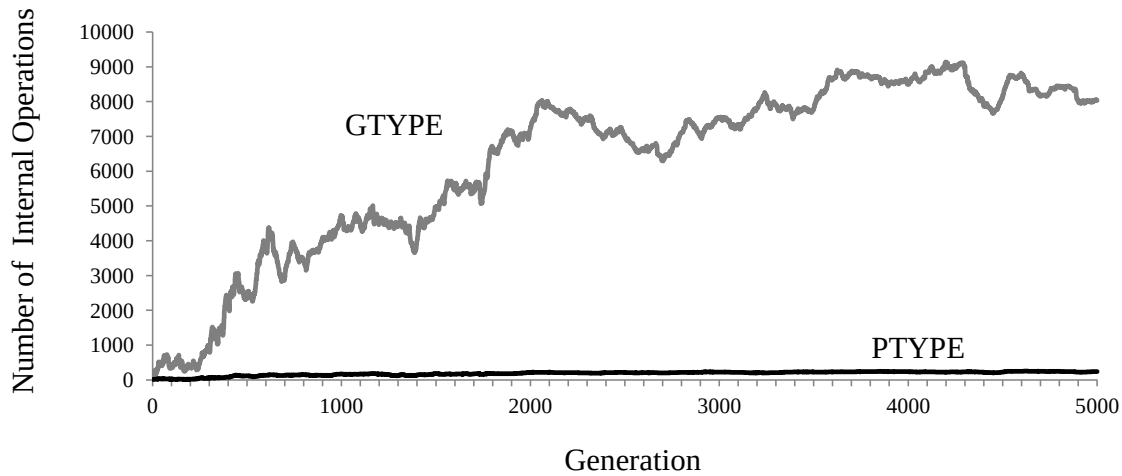


図 5.8: 平均演算数およびイントロンを除いた平均演算数の推移.

5.3 考察

従来手法である ACTIT との比較を行った. ACTIT は, 提案手法と同様に遺伝的操作制限なし, 長さ制限なしという特徴を持つため, 数値演算フィルタおよび多目的最適化の有無の影響に着目した比較を行うことができる. 非終端ノードには比較のため提案手法, 従来手法の両方で表 B.1 を使用することとし, 数値データ型の入出力を含まない表 B.1(c) のみを用いた場合と, それに同 (d) の各フィルタに固定閾値 $V^{(2)} = \{0, 0.25, 0.5, 0.75, 1\}$ を与えたもの (55 種) を加えた場合について実験した. 終端ノードは表 5.2 を用いた. また, ACTIT は多目的最適化ではないため, 適応度関数として $ScoreA$ だけを使用した.

図 5.9 は, 従来手法および提案手法について, それぞれ第 5000 世代まで進化させた初期乱数シードの異なるプログラム 3 個ずつを, 5 枚の教師画像 (Training) および 10 枚の未知画像 (Test) にそれぞれ適用した結果と, 目標画像との PSNR の分布である. 未知画像の目標画像は結果検証用に手作業で作成した.

平均値については, 従来手法, 提案手法いずれの場合も画質評価で良いとされる基準を十分満たしている. 一方, 最低値については, 従来手法では画像によっては 10dB 台と非常に低い値となることがあったが, 提案手法ではすべての場合で 25dB を上回っている. これは, 図 5.5 で示したような数値演算を活用して画像条件に対して適応的に処理内容を変えるプログラムの生成が, 多目的最適化の導入により促された結果と考えられる.

従来手法に固定閾値を用いた場合は, 用いない場合および提案手法のどちらよりも劣る結果となった. 早期に局所解に収束してしまったケースが見られた他, 獲得したプログラムサイズが相対的に小さかった. これには, フィルタが多数増えたことにより, 適切な

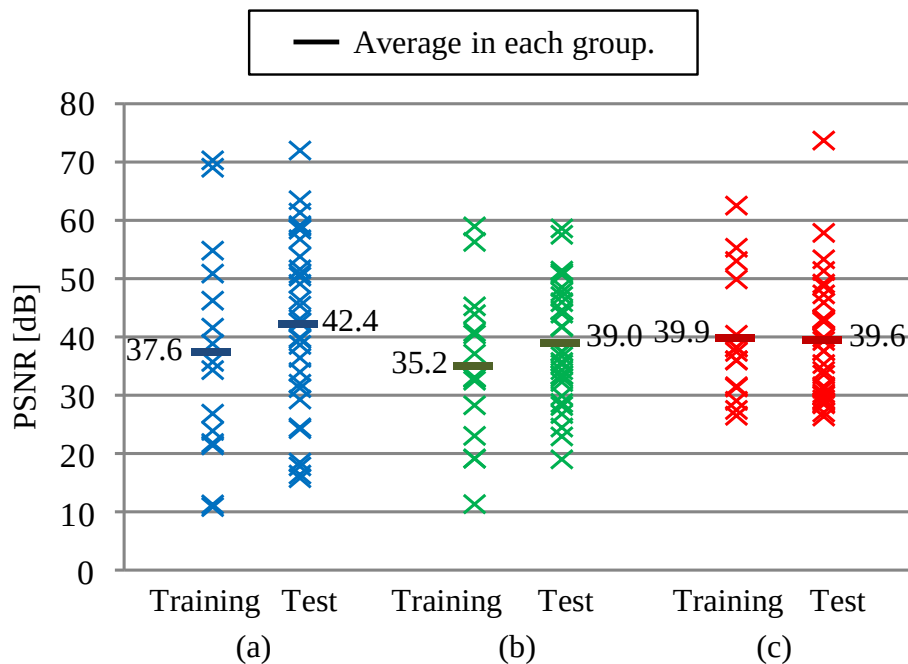


図 5.9: 獲得したプログラムで処理された教師画像および未知画像と目標画像との PSNR. (a) ACTIT, (b) ACTIT(固定閾値フィルタ使用), (c) 提案手法.

フィルタが確率的に選ばれにくくなっている, 十分な探索のためにより多くの世代数が必要になっている, などの可能性が考えられる.

図 5.10 に, 提案手法および従来手法 (固定閾値フィルタなし) それぞれで *ScoreA* が最も高かったプログラムを未知データに適用し, その結果画像と目標画像との PSNR を示す. 花の色が黄, 赤, 青, 白と異なっており, 背景も葉や空, 土など, 条件が多様であるにも関わらず, 良好な結果が得られている. PSNR の平均値は従来手法 (45.5) が提案手法 (44.7) に比べてやや高いが, 提案手法では, 値が極端に低いものがなく, 背景を少なくとも半分程度は除去できているため, 人間が最終的な調整を加えることは容易である. 一方, 従来手法では背景が多く残っているものがあり難しい.

5.4 まとめ

本章では, 画像と数値のペアをデータ型とする多目的遺伝的プログラミングを用いて, 数値演算を含む画像フィルタを自動生成する手法を提案した. 花写真の背景消去課題に適用し, 未知画像に対しても頑健であるという結果を得た.

今後は, 背景除去以外の画像処理や, スコアリング, クラスタリングなどの数値演算を出力するプログラムの生成への応用について, 検討を行いたい. また, 個体の多様性を維

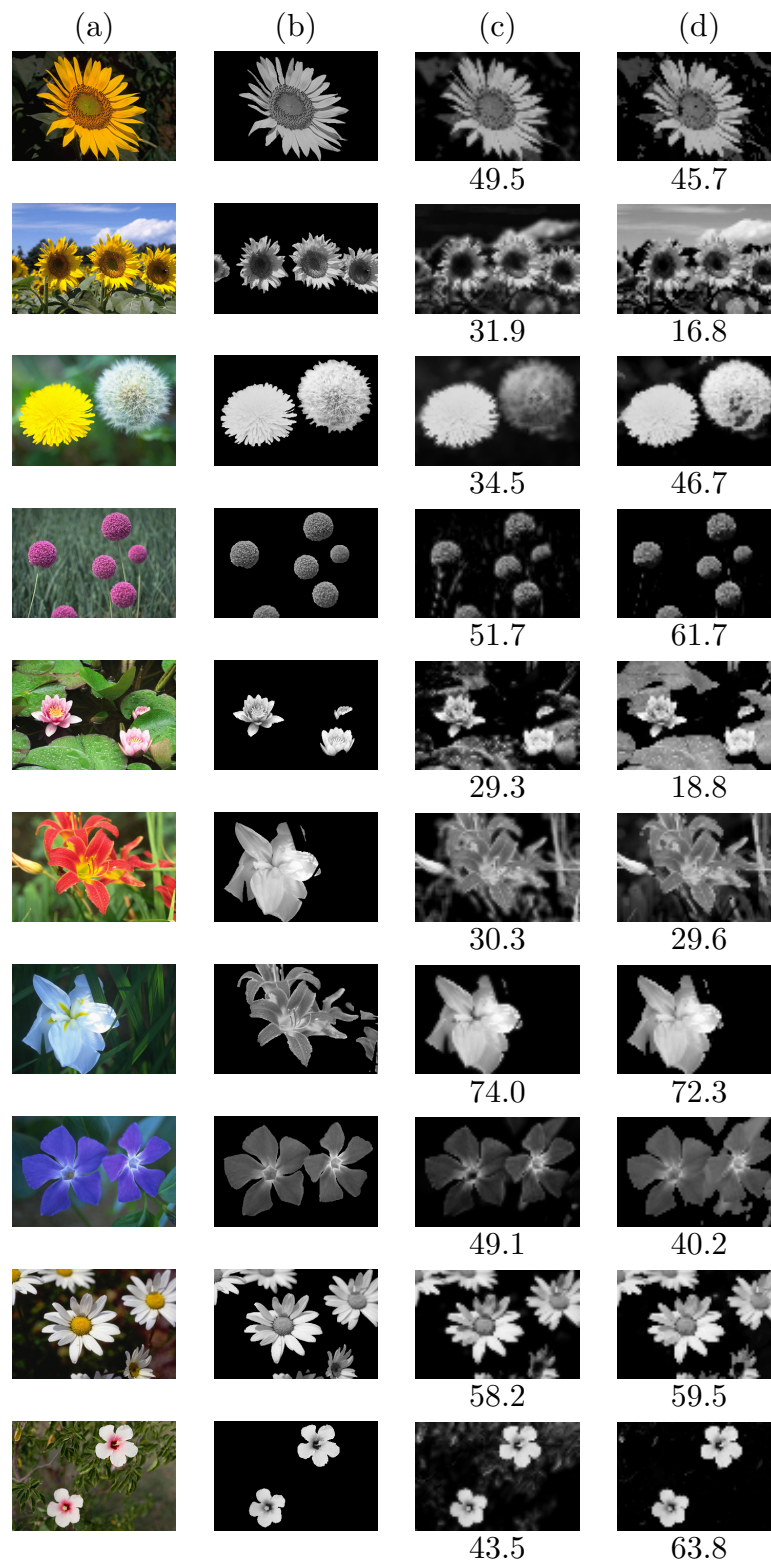


図 5.10: 獲得したフィルタを未知画像に適用した結果. (a) 未知画像, (b) 目標画像, (c) 提案手法による結果と PSNR, (d) 従来手法による結果の PSNR.

持した効率的な探索や、プログラムの適応度として実行時間などを取り入れることについても、検討を行いたい.

第 6 章

画像分類プログラムの自動生成

本章では、提案手法が数値データを取り扱うことができるという特長に着目し、Image-Value ツリーを用いた遺伝的プログラミングを用いて、画像分類プログラムを自動生成する方法について検討を行う。また、自動生成された画像分類プログラムと、同じく自動生成された背景除去プログラムを組み合わせ、全体としてより高性能な背景除去プログラムを作成する方法についても述べる。

6.1 画像分類プログラムの自動生成

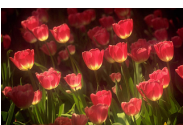
提案手法を、画像分類プログラムの自動生成に適用するための、画像分類システムの設計、自動生成で使用する適応度関数と遺伝的プログラミングパラメータの設計、および自動生成の結果についてそれぞれ述べる。

6.1.1 システム設計

花画像分類システムは、入力された画像を、特徴に基づいてどのクラスに属するかを判定するものとする。各クラスは、主に画像に含まれる花の色に基づいて決められており、本章ではクラス A：黄色の花、クラス B：赤い花、クラス C：青い花、の 3 つのクラスを定義する。また、いずれの特徴も持たない場合は、一般 (Generic) クラスに属すると判定する。すなわち、花画像分類システムの判定は 4 種類となる。表 6.1 に、画像とクラスの例を示す。

花画像分類システムは、自動生成された複数のサブ分類器 (Sub-classifiers) と、投票ロジック (Voting Logic) で構成されており、どちらも Image-Value ツリーで表現されたプログラムである。花画像分類システムに Image-Value ペア x を入力すると、画像クラス c として A, B, C または Generic を出力する。図 6.1 にブロック図を示す。複数のサブ分

表 6.1: 画像分類器作成に使用するトレーニング画像.

Image	Class	Image	Class	Image	Class
	A		B		Dummy
	A		B		Dummy
	A		C		Dummy
	A		C		Dummy
	B		C		Dummy

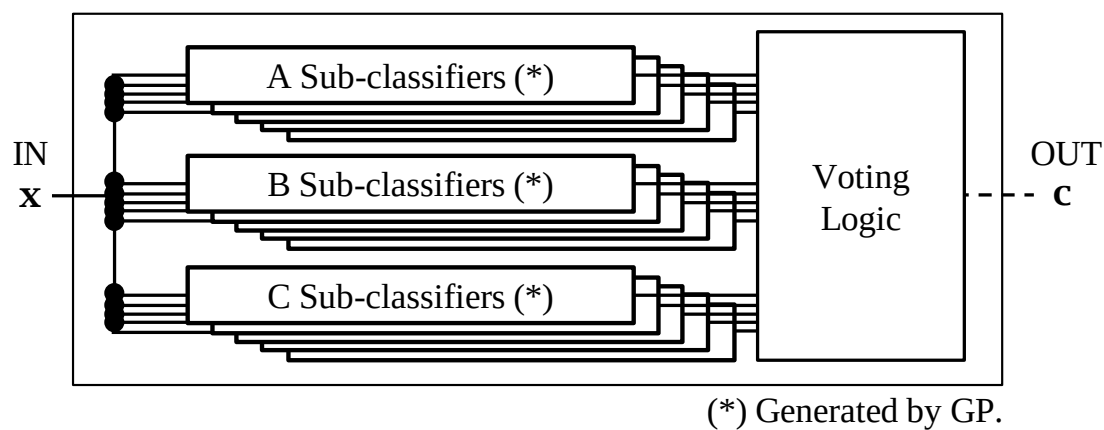


図 6.1: 画像分類器のブロック図.

類器と投票ロジックを用いる理由は、局所解による影響を取り除くためである。遺伝的プログラミングで得られるプログラムは確率的なものであり、適応度が高い解であってもトレーニングサンプルに極端に適応した局所解である可能性もあるし、そもそも適応度の高い解が見つからない場合もある。異なる初期状態から生成されたプログラムは少しずつ異なる特性を持っていると考えられるため、そのような複数のプログラムの投票を用いることで、全体として安定した性能を得られることが期待される。

サブ分類器は、あるクラスに属するか属さないかを判定する、one-versus-rest 型の 2 クラス分類器である。分類したい花画像を画像成分とした Image-Value ペアとして入力することで、クラスに属するかどうかの判定を Image-Value ペアの数値成分として出力する。投票ロジックは、全サブ分類器の出力を総合して、最終的な分類を決定する。投票は次の様に行う。1 つのサブ分類器について、出力 Image-Value ペアの数値成分が、0.7 以上であればその分類器が属するクラスに 1 票を投じるものとする。全サブ分類器の出力について同様の投票判定を行い、投票が最も多かったクラスを花画像分類システムに入力された画像のクラスと決定する。優越するクラスがない、あるいはいずれのクラスも得票がない場合は、一般クラスであると判定する。本論文では、グループごとに 5 個ずつ、合計 15 個のサブ分類器を用いた。

6.1.2 適応度関数の設計

適応度関数は、1 要素の多目的ベクトルとして定義する (6.1)。画像分類プログラムの目的関数は一つであるが、提案手法で使用している多目的遺伝的プログラミングの枠組みに適用するためである。

$$Fitness_{VC}(F) = (ScoreC(F)) \quad (6.1)$$

$$ScoreC(F) = \sum_{k=1}^K \phi(F_V(\mathbf{x}^{(k)}), \mathcal{V}^{(k)}), \quad (6.2)$$

$$\phi(V, \mathcal{V}) = \begin{cases} 0 & |V - \mathcal{V}| < 0.3, \\ 1 & |V - \mathcal{V}| \geq 0.3. \end{cases} \quad (6.3)$$

F は評価対象の Image-Value ツリー、 F_V はその画像成分、 $\mathbf{x}^{(k)}$ は k 番目のサンプルデータ、 $\mathcal{V}^{(k)}$ は k 番目のサンプルデータがこのクラスに属するかを与える入力データでこのクラスに属するならば 1、属さないならば 0 のいずれかの値をとる。3 つのクラスのサブ分類器を作成するので、実際には $\mathcal{V}^{(k)}$ は 3 種類のデータセットを作成する。 $\phi(V, \mathcal{V})$ は、評価対象のプログラム F が k 番目のサンプルのクラスを正しく判定したかどうかを評価する関数で、正しく判定した場合は 0、誤っている場合は 1 を返す。プログラムの出力した Image-Value ツリーの出力の数値成分 V' と、トレーニングサンプルが目的のクラスに属するか属さないかを与える関数 $\mathcal{V}_C^{(k)}(c)$ の差が、0.3 未満の場合は正しい判定、0.3 以上の場合は誤った判定であるとする。 K はトレーニングサンプルの数である。式 (6.1) の成分の最小値は 0 である。

表 6.2: 画像分類課題の設計.

Objective:	Find an image processing program that determines whether input images belong to the given class or not.
Terminal set:	See Table 5.2.
Function set:	See Table B.1.
Fitness cases:	15 pairs of an image and a class.
Raw fitness:	A value element of the IV pair resulted of the generated filter.
Standardized fitness:	See Equation (6.2).
Parameters:	Population size $M = 100$, Maximum generation $G = 1000$, Crossover ratio $r_{crossover} = 0.6$, Mutation ratio $r_{mutation} = 0.0333333$,
Success predicate:	None. Evolve until the G generations have been processed.

6.1.3 パラメータ

遺伝的プログラミングのパラメータは表 6.2 のように設定した. 終端ノードとして, 第 5.2.3 節と同様に, I 成分は図 5.3 に示した原画像の輝度成分 (グレースケール), 赤成分, 緑成分, 青成分を, V 成分は 0 を, それぞれ持つような Image-Value ペアを与えた (表 5.2). 非終端ノードとして, 表 B.1 に示した Image-Value フィルタを使用した. 教師データセットサイズ $K = 15$, 画像サイズ $m, n = 80, 53$ とした. 遺伝的プログラミングパラメータは, 個体数 $M = 100$, 最大世代数 $G = 1000$ とし, G 世代まで実行することを終了条件とした. 交叉率は 0.6, 突然変異確率は 0.0333333 とした.

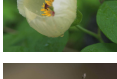
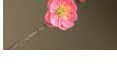
6.1.4 適用結果

表 6.3 に, 自動生成したプログラムによるトレーニングサンプルの分類結果を示す. Class は, 画像に与えたクラスである. Dummy は特定のグループに属さない攪乱サンプルである. A0 から A4 は, クラス A の 5 つのサブ分類器による判定結果で, サブ分類器がクラス A に属すると判断した 0.7 より大きい値を太字で示した. B0 から B4 および C0 から C4 も, クラス B, クラス C について同様である.

サブ分類器によっては属するクラス以外に値を与える場合もあるが, いずれのサンプル画像についても, 投票を考慮すれば良好に分類器が生成できている.

[illegible]

表 6.4: 分類器を未知画像に適用した結果.

	A0	A1	A2	A3	A4	B0	B1	B2	B3	B4	C0	C1	C2	C3	C4	Result
	0.0	1.0	1.0	0.1	0.9	0.3	0.0	0.9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	A
	1.0	1.0	0.0	0.0	1.0	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	A
	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.1	1.0	0.0	0.0	1.0	0.0	0.0	0.0	A
	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	A
	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	B
	0.3	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	G
	0.0	0.0	0.0	0.5	1.0	0.3	0.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	B
	0.1	0.0	0.0	0.5	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	G
	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	B
	1.0	0.0	1.0	0.5	0.9	0.2	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	A
	0.0	0.0	0.0	0.0	0.0	0.3	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	B
	0.0	1.0	0.0	0.2	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	0.0	0.0	C
	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	G
	0.2	0.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	1.0	1.0	C
	0.2	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	C
	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	0.9	C
	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	C
	0.2	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.9	C

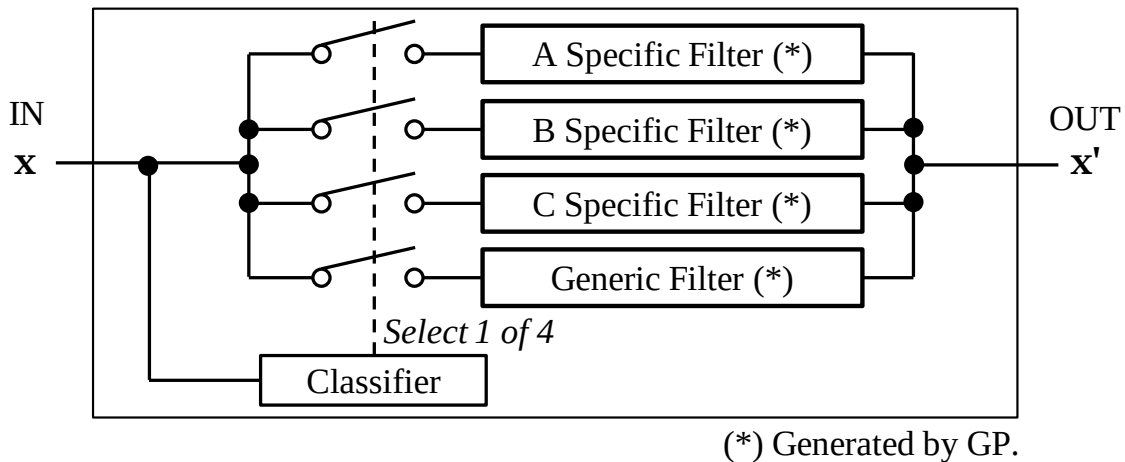


図 6.2: 強化型背景除去フィルタのブロック図.

表 6.4 は、テスト用の未知画像を分類した結果である。左の列がテスト画像、A0-C4 はサブ分類器の計算結果で太字は投票を示す。Results 列が投票によって決められたテスト画像のクラスである。

6.2 背景除去プログラムの強化への応用

花画像分類システムの評価のために、背景除去フィルタと組み合わせて強化型背景除去フィルタの構築を行った。

これは、人手で分類した画像クラスを正答としても主観的な部分が多く妥当性の評価が難しいため、分類器の正しいクラス分けに依存したシステムを構成し、その全体の挙動を評価することで客観的な判断を行うためである。

6.2.1 システム設計

強化型背景除去システムは、花画像分類器に、クラス特化型背景除去フィルタを組み合わせたものである。クラス特化型背景除去フィルタとは、予め特徴が偏ったトレーニングセットに対してのみ良い結果を与えるように設計された背景除去フィルタである。例えば、クラス A に特化した背景除去フィルタは、クラス A に属する画像については良好な示すように設計されているが、クラス A ではない画像については特定の配慮はなされていない。そのため、トレーニングに使用したクラスに類似した画像に対しては高い性能を示すと期待されるが、それ以外の特徴を持つ画像については低い性能を示す可能性も考えられる。しかし、分類器によって背景除去前に正しいクラス分類が行われるならば、システム全体として、汎用背景除去フィルタよりも良い性能を示すと期待できる。

図 6.2 に強化型背景除去システムのブロック図を示す。Classifier は 6.1 節で述べた花画像分類システムである。分類結果によって、どの特化型背景除去フィルタ (X Specific Filter) が使われるかが決定される。また、一般クラスに分類された場合は、第 5 章の背景除去フィルタで処理するものとする。

6.2.2 適応度関数の設計

適応度関数は、第 5 章と同じである。

6.2.3 パラメータ

パラメータは、以下の 2 点を除き、第 5 章と同じである。

トレーニングデータセットは、そのクラスに属する画像のみとした。例えば、クラス A 特化型背景除去フィルタの生成は、表 6.1 のクラス A の画像のみを用いた。

また、最大世代数 $G = 3000$ とした。これは、第 5 章での結果から、3000 世代まで世代交代を行えば十分と判断したためである。

6.2.4 適用結果

構築した強化型背景除去システムに、未知画像を入力し、性能を調べた。18 例中 11 例で一般フィルタよりも改善が得られた。16 例でグループ別フィルタが性能上優位性であるとの結果が得られたが、5 例正しいグループ分類が行われなかった。原因別に、1 例は誤った分類をしたもの、4 例は未知分類になったもの、となっている。

表 6.5 に、強化型背景除去フィルタの結果を示す。値は、フィルタで得られた画像と手作業により作成した模範解答画像の PSNR である。A, B, C, Generic は、内包する各クラス特化型および汎用背景除去フィルタの結果である。各フィルタのうち最良の結果を太字で、クラス分類器が決定したクラスの値を下線で示した。Improv.[%] は、汎用フィルタと比べた場合のスコア改善度合いを示す。分類器が汎用クラスと判定した場合は 0.0 の値となる。66% のケースで、性能改善が見られた。改善度合いの平均値は、18.6% であった。

6.3 考察

比較のため、libsvm [70] を用いてサポートベクターマシン (Support Vector Machine, SVM) による分類器の構成を試みた。SVM は 2 クラス分類器としてよく用いられているアルゴリズムの一つである。トレーニングデータとして、画像の輝度 (グレースケール)

表 6.5: 強化型背景除去フィルタを未知画像に適用した場合の PSNR.

Image	A	B	C	Generic	Improv. [%]
	<u>24.5</u>	21.1	23.6	28.4	-13.7
	27.9	25.9	8.5	9.5	195.0
	<u>23.8</u>	21.9	24.2	25.3	-5.9
	38.9	20.7	18.2	21.7	79.6
	56.4	<u>32.6</u>	23.0	28.8	13.4
	20.8	11.4	13.8	<u>18.0</u>	0.0
	47.7	51.2	40.2	41.1	24.5
	52.1	35.8	18.4	<u>37.2</u>	0.0
	30.7	63.8	50.9	49.5	28.8
	32.0	21.2	39.5	49.3	-35.1
	21.3	<u>64.3</u>	47.3	48.6	32.4
	12.8	12.6	57.8	55.6	3.9
	35.4	57.3	28.8	<u>29.2</u>	0.0
	27.7	43.7	31.4	22.2	41.1
	28.7	27.6	<u>30.5</u>	31.1	-2.0
	31.3	30.0	<u>62.4</u>	69.0	-9.7
	23.6	20.8	<u>32.7</u>	44.4	-26.2
	47.4	64.1	<u>59.4</u>	54.6	8.7

成分、赤成分、緑成分、青成分のそれぞれピクセル値を含む、16,960 次元のベクトルを用いた。しかし、libsvm では分類器を構成できなかった。これは、本論文で用いたトレーニングデータセット（15 画像）が、SVM には小さ過ぎたためと考えられる。ピクセル値の集合でなく、予め特徴量を抽出して利用することで SVM でも分類器を構成できる可能性が考えられるが、適切な特徴量の選択自体が困難な課題である。提案手法は、少ないトレーニングデータセットからでも、生成過程で特徴量の選択を行いながら、分類器を構成できることに優位性がある。

グループ別フィルタの方が性能が悪くなった 2 例および未知分類と判断したもの 4 例は、教師画像ではカバーできなかった条件を持っていたことが原因と考えられる。分類器作成の時点で処理対象として想定する画像サンプルを十分な数用意することで改善できると考えられる。

6.4 まとめ

本章では、Image-Value ペアに基づく遺伝的プログラミングを用いた画像分類器の生成の詳細と、それを強化型背景除去フィルタに用いる方法を述べた。分類器の性能は高くはないが、クラス特化型フィルタと組み合わせることで、汎用フィルタよりも性能の高い背景除去フィルタが構成できることを示した。

今後の課題として、多数の画像を用いた実験を行い、より多様な条件での検証を行いたい。また、他のアルゴリズムと組み合わせることでその性能を強化できるかについても検討を行いたい。

第 7 章

考察

本章では，個体サイズの増加と，Image-Value ペアの拡張可能性と課題について議論を行う．

7.1 個体サイズの増加

第 5 章で見たように，ブロートと見られる急激な GTYPE サイズの増大が観察されている．提案手法により PTYPE のサイズは小さい値に飽和していると読み取れるため，計算量に関しては低く保たれていると考えられる．しかしながら，GTYPE サイズが大きくなることから交叉や突然変異などの遺伝的操作のコストが高くなるため，個体サイズを小さく維持する方策を取り入れることが望ましい．

第 3 章で言及したように，最大サイズを制限する方法は，サイズ抑制効果は認められるが適切な値を決定することが困難である．

パラメータレスでサイズ抑制を行う手法として，Bleuler らが多目的遺伝的アルゴリズムである SPEA2 を用いて，複数目的関数の一つに個体サイズを導入する手法を提案している [71]．しかし，本研究の予備実験過程では早期に集団が小さいサイズの個体で占められてしまう初期収束の問題が観察された．これは，サイズ項が他の目的関数に対して大きく変動するため，他の目的関数に比べて強く働き過ぎるためであると考えられる．論理式を獲得する Bleuler の実験に比べ，本研究は非終端ノードの種類が圧倒的に多く課題も複雑であるため，大きな個体サイズの必要性が相対的に高いものと考えられる．

効果的なサイズ抑制については，今後の研究課題である．

7.2 画像，数値以外のデータへの拡張性

本論文では，Image-Value ペアという簡潔なアイデアを用いるだけで遺伝的プログラミングでも画像処理と数値演算が混在するプログラムを扱えるようにする方法を提案し，第5章，第6章でそれを確認してきた．さらに，画像，数値以外のデータ型も扱えるように拡張することは可能であろうか．

Image-Value ペアでは画像と数値を組にしているところを，例えば，画像と数値とベクトルの組，というように拡張すれば対応は可能と考えられる．

ただし，遺伝的プログラミングによる自動生成の実行時間が大幅に伸びるものと予想される．まず，イントロンが増えていくことが予想されるため，適応度に影響がない遺伝的操作が増えてしまい，GTYPEのブロートがさらに加速すると考えられる．さらに，遺伝的操作の効果が低下することを補わなければ探索性能が維持できないため，例えば集団の個体数を増やさなくてはならない．前述の個体サイズの抑制と，付録Aで述べる個体共通部分の統合などの対策を合わせた上で，型の種類を増やす効果とのトレードオフを検討する必要がある．

第 8 章

結論

本論文では、画像アプリケーションの開発生産性の課題を解決するため、遺伝的プログラミングにより画像処理と数値演算が混在したプログラムを自動生成する手法について提案した。

第 2 章では、遺伝的プログラミングについて述べた。遺伝的プログラミングは、広く使われているプログラム自動生成フレームワークの一つであり、生物の進化過程における遺伝のメカニズムを応用している点に特徴がある。演算子をノードとした木構造でプログラムを表現し、それを生物個体として見立てて、所望の結果に近いほど適応度が高いと考えて選択し、交叉や突然変異などの遺伝的操作を加えることで新たな次世代の個体を生成して、世代交代を繰り返すことで、最適に近いプログラムを生成していく。課題に対するプログラムの良さは必ずしも単一の目的関数で測ることができるとは限らず、複数の目的関数が互いにトレードオフの関係にある場合も多い。このような多目的最適化問題に対して、パレート最適という概念で最適解を探す、多目的遺伝的アルゴリズムが提案されており、遺伝的プログラミングにも応用が可能である。

第 3 章では、遺伝的プログラミングを用いた画像処理プログラムの自動生成の従来手法と課題について述べた。画像処理プログラムは画像データだけを扱えば良いとは限らず、パラメータや特徴量の抽出のために数値データの処理も必要である。しかし、遺伝的操作で互換性のない入出力を持つノードが接続されると実行できないプログラムができてしまうため、対策が必要である。また、サイズ制約は専門家でないアプリケーション開発者にとって難しい。

第 4 章では、この問題に対し、Image-Value ペアという概念を導入することを提案した。遺伝子型では単一データ型、表現型では複数データ型、という二重性を実現し、致死遺伝子が発生する無効接続の問題を原理的に解消できることについて述べた。Image-Value ペアを処理するプログラムである Image-Value ツリーの評価では、表現型に移行する際にイントロンを削除し、計算量が大幅に削減することができるため、提案手法では個体サ

イズに制限を設けることを避けられる。

第 5 章で、提案手法の効果を見るため、花写真画像からの背景除去フィルタの自動生成課題に適用し、実験を行った。遺伝的プログラミングフレームワークには、多目的遺伝的アルゴリズムの代表的手法である NSGA-II を基礎として、遺伝的プログラミングで一般的に用いられる木構造型遺伝子と木構造向けの遺伝的操作を適用して用いた。注目部分と背景の処理をそれぞれ別に PSNR で評価したスコアを用い、複数のトレーニングデータに対する平均性能と最低性能を同時に向上させる 2 目的最適化問題とした。自動生成されたプログラムは、未知画像に対する処理を従来手法と比べた場合、平均性能はほぼ同等だが最低性能を 15dB から 25dB に大幅に向上させることができた。また、数値データの処理ができるため、従来手法で固定数値パラメータを使用した場合に比べて平均性能も向上させることができた。初期乱数を変えた 3 回の試行をしたところ、それぞれで獲得したプログラムの骨格部分が類似していることを確認した。これは、提案手法が再現性を持っていることを示している。

第 6 章では、数値データを計算できる特徴を活用して、花写真画像分類フィルタの自動生成を行った。さらに、分類グループごとに専用に自動生成した背景除去プログラムと組み合わせ、全体として強化された背景除去システムを構築することを提案し、単純な背景除去フィルタシーケンスに比べて 18% 高い性能を示した。分類器の代表的手法である SVM はサンプル数を多くする必要があるが、同じ条件では分類器を構成できなかった。

第 7 章では、個体サイズの増加に関する課題と、Image-Value ペアの拡張について議論を行った。提案手法によりサイズ制限をしなくても PTYPE サイズを抑えて画像処理計算量を低く保つことができるが、GTYPE としてはブロー状態にあり、依然としてサイズ抑制は重要であることを述べた。しかし、適切なサイズは事前にわからないため、効果的なサイズ抑制手法の開発が必要である。Image-Value ペアの拡張は、組になるデータ型を増やせば可能であるが、イントロンが増加し、遺伝的操作の効果が低下し、ブローがしやすくなる可能性について述べた。

今後は、トレーニングデータ数、個体数、世代数を大きくした場合の効果とともに、計算速度を向上する方法についても検討を行いたい。また、顔認識のような、より本格的な主アルゴリズムのための前処理生成手法として提案手法を導入した場合について、適用方法の変化やシステムの性能向上について検討を行いたい。

付録 A

Image-Value ツリー評価の最適化

Image-Value ツリーの評価方法は第 4 章で述べたが，遺伝的プログラミングが数十万もの解候補を試行していくことおよび画像データの大きさのため，現実に行うためには演算量とメモリ使用量の最適化が必要となる．本章では，Image-Value ツリーの評価部分の実装について，まずその課題について明らかにし，そして課題を解決するための実装の最適化として，共通部分の再利用，ノードの評価順序，並列計算，について順に述べる．

A.1 Image-Value ツリー評価の課題

遺伝的プログラミングによる画像処理プログラムの自動生成は，演算量とメモリ使用量が非常に大きいという本質的特徴を持つ．これは，画像処理の特徴および遺伝的プログラミングが非常に多くのプログラムを実際に実行しながら評価していくという手法であることが理由である．

■演算量 一回の自動生成の演算量 L_{GP} は，非終端ノード 1 つあたりの平均演算量 $\overline{L_{NT}}$ ，個体あたりの平均非終端ノード数 $\overline{N_{tree}}$ ，サンプル数 K ，世代あたりの個体数 M ，および世代数 G を用いて，次の式で見積もることができる．

$$L_{GP} = \overline{L_{NT}} \times \overline{N_{tree}} \times K \times M \times G. \quad (A.1)$$

表 A.1 に，各ノードの演算時間の相対的な大きさを示した．これらの値は本研究において行った多数の実験を通じて取得したデータに基づく結果であり，第 5 章の第 5000 世代のものではない．また，画像サイズ，利用した PC や，ライブラリのバージョン，コンパイラによっても異なるため，あくまで傾向を見るためのものである．

また，表 A.2 に，背景除去プログラムの自動生成（第 5 章）のある試行について，第 5000 世代の最良個体 (Best) および第 5000 世代に含まれる全個体 (All) について，含ま

表 A.1: Image-Value フィルタの相対的実行時間.

Node	Time	Node	Time
AvgPixel	383	Binarize	2413
MinPixel	1551	BinarizeInv	1802
MaxPixel	876	Add	137
VNegate	8	AlgebraicProd	801
VAdd	3	AlgebraicSum	6153
VAlgebraicProd	4	BoundedProd	2529
VAlgebraicSum	12	Diff	138
VDiff	4	LogicalProd	195
VLogicalSum	8	LogicalSum	230
VLogicalProd	6	Subtract	155
VSubtract	2	HighPass	77705
Dilate3x3	1708	LowPass	99026
Erode3x3	2281	ThresholdV	170
Median3x3	5359	ThresholdInvV	186
Laplacian3x3	3598	BinarizeV	196
Sharpen3x3	5709	BinarizeInvV	225
Smooth3x3	6266	AddV	2335
SobelX3x3	11950	ModAvgV	577
SobelY3x3	11021	MultiplyV	1188
Sobel3x3	16498	SubtractV	1478
Negate	431	FillV	658
ProjectionX	3786	Select	45
ProjectionY	4524	SelectV	45
		Average	5488

れるノード数を種類ごとに示した. 例えば, Binarize フィルタは, 最良個体に 54 個含まれており, 最良個体を含む全個体の合計では 3605 個含まれている.

ノードの種類ごとの出現頻度はあまり偏りはないため $\overline{L_{NT}} = 5488$ とし, $\overline{N_{NT}} = 167213/100 = 1672.1$, $K = 5$, $G = 5000$, $M = 100$ を用いて計算すると, $L_{GP} = 2.2939 \times 10^{13}$ となる. Time 1 単位を 100 ナノ秒で実行できるコンピュータを用いた場合, およそ 26.5 日間かかる計算量となる. 遺伝的プログラミング全体では, 選択, 遺伝的操作, などの処理時間が加わる.

■メモリ使用量 ある世代の個体集団 $\mathbf{P}$ を一個体ずつ逐次的に評価する場合に必要なメモリ容量 $S(\mathbf{P})[\text{bytes}]$ は, Image-Value ペア 1 つあたりのメモリ容量 S_{pair} , 事例サンプル数 K , 個体数 M , および評価中に必要となる一時的な記憶領域数 $R(\mathbf{T})$ を用いて,

$$S(\mathbf{P}) = S_{\text{pair}} \times (2K + M + \max_{\mathbf{T} \in \mathbf{P}} R(\mathbf{T})) \quad (\text{A.2})$$

表 A.2: 第 5000 世代の最良個体 (Best) および全個体 (All) に含まれる Image-Value フィルタの種類別個数.

Node	Best	All	Node	Best	All
Gray0	313	26557	Binarize	54	3605
Green0	365	26404	BinarizeInv	62	3596
Red0	349	26368	Add	54	3764
Blue0	374	26777	AlgebraicProd	58	3801
Subtotal	1401	106106	AlgebraicSum	50	3716
AvgPixel	44	3581	BoundedProd	46	3722
MinPixel	49	3667	Diff	50	3716
MaxPixel	43	3324	LogicalProd	56	3895
VNegate	53	3504	LogicalSum	50	3502
VAdd	47	3722	Subtract	45	3712
VAlgebraicProd	51	3582	HighPass	50	3784
VAlgebraicSum	56	3754	LowPass	37	3429
VDiff	40	3663	ThresholdV	42	3718
VLogicalSum	52	3782	ThresholdInvV	50	3374
VLogicalProd	42	3750	BinarizeV	53	3389
VSubtract	40	3545	BinarizeInvV	43	3666
Dilate3x3	45	3657	AddV	49	3760
Erode3x3	46	3553	ModAvgV	40	3610
Median3x3	46	3712	MultiplyV	64	3668
Laplacian3x3	47	3525	SubtractV	58	3637
Sharpen3x3	54	3621	FillV	48	3603
Smooth3x3	54	3823	Select	48	3632
SobelX3x3	45	3644	SelectV	43	3571
SobelY3x3	42	3502	Subtotal	2245	167213
Sobel3x3	48	3598	Total	3646	273319
Negate	55	3665			
ProjectionX	52	3600			
ProjectionY	44	3569			

と表すことができる. ただし, $\mathbf{T}$ は $\mathbf{P}$ に含まれる個体を表す. また, K の係数 2 は入力と出力の合計を意味している.

S_{pair} は, 画像データに必要なメモリ容量 $S_I[\text{bytes}]$, 数値データに必要なメモリ容量 $S_V[\text{bytes}]$, およびログなどを含むオーバーヘッド ϵ を用いて,

$$S_{\text{pair}} = S_I + S_V + \epsilon \quad (\text{A.3})$$

と計算される.

評価中に必要となる一時的な記憶領域数 $R(\mathbf{T})$ は, 個体の木構造の形状とノードの評価順序に依存する関数で, メモリ使用量に大きく影響を与える. Image-Value ツリーは, PTYPE 表現を葉から根に向かって順に処理していくことで評価する. そのためには,

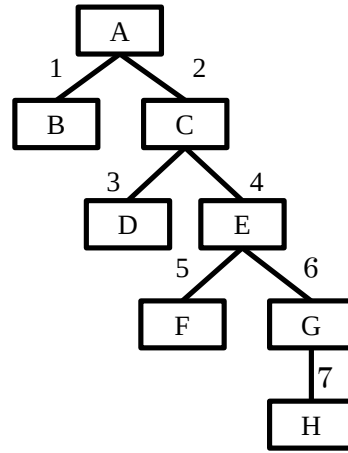


図 A.1: 偏ったノード配置を持つ木の例.

Image-Value ツリーの PTYPE のノードを根から再帰的にサブツリーを評価していく方法や、全てのノードをトポロジカルソートして葉から根に向かう方向で辿る方法がある。しかし、2 以上の入力を持つノードは、入力エッジにつながるすべての部分木の評価を終えてからでないと自身の評価ができない。同時に必要な記憶領域の理論的最小値は、(フィルタの入力数の最大値) + (フィルタの出力数の最大値) であるが、辿る順序によっては中間結果の記憶領域が非常に大きくなる可能性がある。

例えば、図 A.1 のような Image-Value ツリーを評価する場合を考える。エッジ 1 側の部分木から評価した場合、エッジ 2 側の部分木の評価が終わるまで、エッジ 1 のデータをメモリ中に保持しておく必要がある。しかし、エッジ 2 側の部分木を先に行えば、エッジ 1 側の中間結果を記憶する必要がない。Image-Value ツリー中の全終端ノードが同じ深さに揃っている**平衡木**であれば、ノード数 N_{tree} を用いて $R(\mathbf{T}_i) = \log_2 N_{\text{tree}}$ 程度だが、図 A.1 のように極端に偏っている場合は入力数の最大値 N_{in} を用いて $R(\mathbf{T}_i) = N_{\text{tree}}/N_{\text{in}}$ 程度になることがある。Image-Value ツリーは数千のノードからなる場合もあるため、深刻な問題となりうる。

第 5 章で用いたパラメータを代入すると、 $S(\mathbf{P}) = 4248 \times (5 + 100 + 2733/4) = 3348486[\text{bytes}] = 3.19[\text{MiB}]$ となる。ただし、画像サイズ $m = 80, n = 53$ と数値に C 言語の double 型を用い、簡単化のため ϵ は 0 とすると、 $S_{\text{pair}} = S_I + S_V = (80 \times 53 \times 1) + 8 = 4248[\text{bytes}]$ と計算される。表 A.2 の全個体のノード数を個体数で割った平均値 $\overline{N_{\text{tree}}} = 2733$ 、および入力数の最大値 $N_{\text{in}} = 4$ を用いた。この程度では問題にはならないが、画像を VGA サイズ (640×480) に変更すると $200[\text{MiB}]$ 程度が必要となり、メモリ使用量がコンピュータのキャッシュ量を超えてしまうため、演算時間にも悪影響を及ぼしてくる。

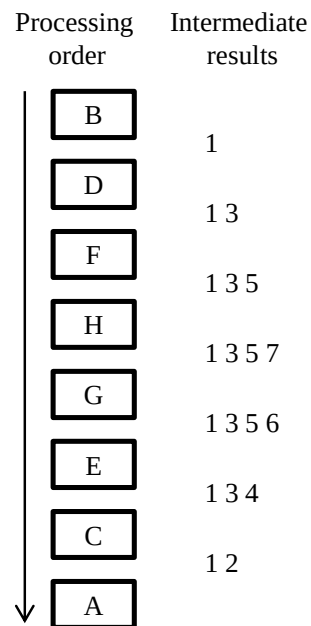


図 A.2: 一般的なトポロジカルソートを用いた場合のノードの実行順序.

■**ディスク I/O** 遺伝的プログラミングを用いた自動生成には長い時間が必要であるため、停電などのトラブル、OSリブート、あるいはプログラムのバグによるクラッシュが発生して意図せず自動生成が中断されてしまうことが考えられる。そのような場合でも、自動生成を最初からやり直すことがないよう、世代ごとに結果を記録していくチェックポイント機能を備えることが望ましい [26].

また、自動生成終了後に結果を分析する場合にも、中間データの保存は必要である。

各世代で保存しておくデータとして、少なくとも Image-Value ツリーの S 式と適応度を含む必要がある。そのうち支配的となる S 式は、表 A.2 から世代に含まれるノード数 273319 と、フィルタ名の括弧を含む文字数 10 とすると、 $273319 \times 10 = 2733190[\text{bytes}] = 2.61[\text{MiB}]$ 程度が必要である。初期世代のノード数を 10000 とし、その後線形に増加するものと仮定すると、5000 世代では、 $5000 \times 10 \times (10000 + 273319)/2 = 7082975000 = 6.60[\text{GiB}]$ の容量が必要となる。

A.2 メモリ使用効率を最適化する評価順序

中間記憶が少なくなるような順序で評価を行うため、提案手法では、Sarkar-Serrano-Simons アルゴリズム [72] を適用し、入力につながる部分木のうち、部分木のサイズが大きなものから順に評価するようにした。

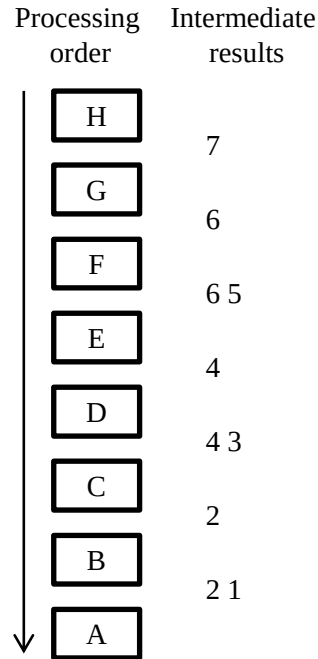


図 A.3: Sarkar-Serrano-Simons 法による効率的なノードの実行順序.

単純に左側からの深さ優先探索によって順序を決めた場合は図 A.2 のようにノード数に比例した数になってしまうが Sarker-Serrano-Simons アルゴリズムで修正されたトポロジカルソートを用いた場合, 図 A.3 のようになり理論的最小値が得られる.

この対策は, 主にメモリ使用量の削減に効果があるが, キャッシュ利用効率が高くなるため演算時間の短縮にも効果がある.

A.3 並列計算の利用

各個体とトレーニング事例サンプルとの組み合わせは各々独立に実行できるので, マルチコア CPU やコンピュータクラスタを利用して並列処理を行うことで実行時間を短縮できる. 並列度 P は, 集団の個体数 M , 事例サンプル数 K を用いて,

$$P = M \times K \quad (\text{A.4})$$

と計算される. 実際に実行可能な並列度は計算資源の量, 特に CPU の個数で決定される. CPU の個数 N_{CPU} を用いると, 実行可能な並列度 P' は,

$$P' = \min(P, N_{\text{CPU}}) \quad (\text{A.5})$$

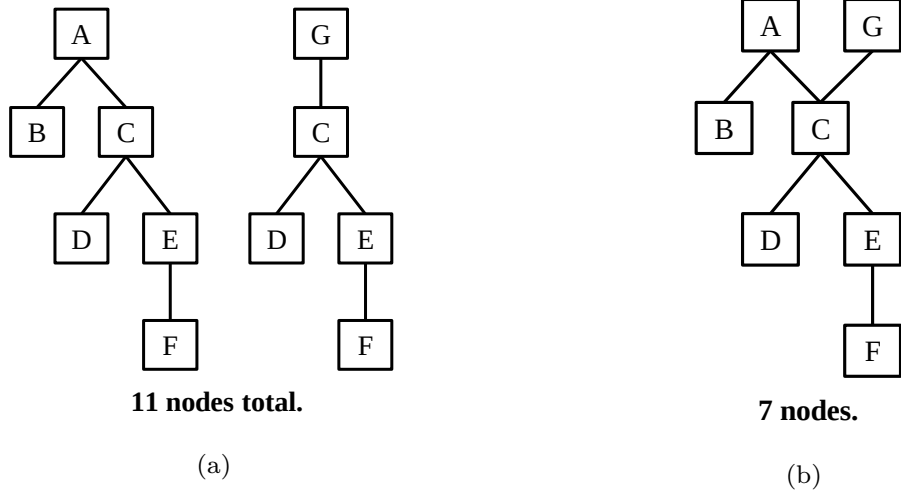


図 A.4: Image-Value グラフへの変換. (a) 統合前の Image-Value ツリー群. (b) 統合後の Image-Value グラフ.

となる. この場合の計算時間は,

$$L_{\text{GP,parallel}} = \overline{L_{\text{NT}}} \times \overline{N_{\text{tree}}} \times K \times M \times G/P' \quad (\text{A.6})$$

$$= L_{\text{GP}}/P', \quad (\text{A.7})$$

メモリ使用量は,

$$S_{\text{parallel}}(\mathbf{P}) = S_{\text{pair}} \times (K + M + \max R(\mathbf{T}) \times P') \quad (\text{A.8})$$

となる.

並列計算は計算時間の削減に効果があるが, メモリ使用量は並列した数に比例して増加する.

A.4 個体間共通部分の計算の統合

同世代の個体集団に含まれる Image-Value ツリーは, 共通する部分木を多数持っているため, 演算結果を共有することで全体として計算量の削減が期待できる. そこで, 共通部分を持つ複数の Image-Value ツリーを統合し, **Image-Value グラフ (Image-Value Graph, IV Graph)** に変換する. Image-Value グラフは, 無閉路有向グラフ (Directed Acyclic Graph, DAG) に分類されるグラフ構造である.

DAG 化した場合の計算時間は, グラフに統合されたツリーの数 U , Image-Value グラフの個数 M_{graph} を用いて,

$$L_{\text{GP,graph}} = \overline{L_{\text{NT}}} \times \overline{N_{\text{graph}}} \times K \times M_{\text{graph}} \times G. \quad (\text{A.9})$$

として計算できる．また，並列度 P_{graph} は

$$P_{\text{graph}} = M_{\text{graph}} \times K, \quad (\text{A.10})$$

並列実行した場合の計算時間 $L_{\text{GP,graph,parallel}}$ は，

$$L_{\text{GP,graph,parallel}} = L_{\text{GP,graph}} / \min(N_{\text{CPU}}, P_{\text{graph}}), \quad (\text{A.11})$$

となる．

すべての Image-Value ツリーを統合した場合 $M_{\text{graph}} = 1$ となるが，並列度に対する寄与がなくなってしまうため，世代に含まれる Image-Value ツリーを複数のグループに分割して複数の Image-Value グラフを作成するのが良い． M_{graph} の値は計算時間 $L_{\text{GP,graph,parallel}}$ が最も小さくなるようにバランスさせることが望ましいが，含まれる Image-Value フィルタの数は個体で大きく異なる場合があるため，経験的には，Image-Value グラフに含まれるノード数の合計が 5,000~10,000 程度になるようにグループ化した場合が効果が高かった．

メモリ使用量については，DAG 化により複数の出力を持つノードを作り出すので，中間結果の最大値は統合した Image-Value フィルタの数程度増える．

関連研究として，Machado らが GTYPE としても常時 DAG として個体データを持つことを提案しているが [73]，本論文中では GTYPE は通常の木構造を用い，評価の際に PTYPE に変換してから DAG 化を行っている．

A.5 保存データの圧縮

世代ごとにディスクに保存するデータは，データ圧縮を行った．S 式は，同じ単語が繰り返し含まれるテキストデータであるため，データ圧縮が有効に働きやすい．第 6 章のサブ分類器の自動生成に必要なディスク容量について，圧縮なしと圧縮ありの場合で比較したものを，表 A.3 に示す．ディスクに保存されるものには，入力として，パラメータ設定ファイル，事例データ画像が，出力として世代チェックポイントが，それぞれ含まれる．チェックポイントは独自構文のテキストファイルで，1 つの世代に含まれる全個体の S 式，適応度，ランクの他，データ解析用に親個体，交叉点，突然変異点，を格納している．

圧縮なし (Uncompressed) は，一つのサブ分類器につき約 1,066~5,249[MiB] のディスク容量を必要としているのに対し，圧縮あり (Compressed) は，同じデータを gzip 形式で圧縮したもので，約 122~748[MiB] と約 87% 程度の容量削減に成功している．圧縮はファイルに書き込む前にメモリ上で行われる．保存するデータ量の減少に伴うディスク I/O の削減効果が大きいので，圧縮に伴う実行時間の増加は無視できる．

表 A.3: サブ分類器の自動生成に必要なとなったディスク容量. Uncompressed は圧縮なし, Compressed は圧縮あり.

Classifier	Uncompressed [MiB]	Compressed [MiB]	Ratio [%]
A0	3023.6	385.1	12.7
A1	4497.1	585.0	13.0
A2	4072.4	527.6	13.0
A3	2610.9	329.0	12.6
A4	2171.3	269.6	12.4
B0	1066.0	121.7	11.4
B1	3987.3	541.6	13.6
B2	5248.9	713.7	13.6
B3	2854.0	362.7	12.7
B4	3092.2	413.0	13.4
C0	1754.5	222.1	12.7
C1	2055.0	259.3	12.6
C2	1944.4	241.7	12.4
C3	1741.7	216.8	12.4
C4	1402.4	170.3	12.1

付録 B

本論文で用いた Image-Value フィルタ

本論文で用いる Image-Value フィルタとその内部フィルタを表 B.1 および表 B.2 に示す。画像処理フィルタ、画像特徴を数値化するフィルタ、数値演算フィルタ、数値をパラメータとして画像を操作するフィルタ、選択フィルタ、などがある。

表 B.1: 利用可能な IV フィルタ. 表中の番号は, 表 B.2 中の番号に対応する.

(a) Numeric operations using image parameters.

Name	f_I	f_V
AvgPixel(x_1)	101	202
MinPixel(x_1)	101	203
MaxPixel(x_1)	101	204

(b) Numeric operations using numeric parameters.

Name	f_I	f_V
VNegate(x_1)	101	205
VAdd(x_1)	101	206
VAlgebraicProd(x_1)(x_2)	101	207
VAlgebraicSum(x_1)(x_2)	101	208
VDiff(x_1)(x_2)	101	209
VLogicalSum(x_1)(x_2)	101	210
VLogicalProd(x_1)(x_2)	101	211
VSubtract(x_1)(x_2)	101	212

(c) Image operations using image parameters.

Name	f_I	f_V
Dilate3x3(x_1)	102	201
Erode3x3(x_1)	103	201
Median3x3(x_1)	104	201
Laplacian3x3(x_1)	105	201
Sharpen3x3(x_1)	106	201
Smooth3x3(x_1)	107	201
SobelX3x3(x_1)	108	201
SobelY3x3(x_1)	109	201
Sobel3x3(x_1)	110	201
Negate(x_1)	111	201
ProjectionX(x_1)	112	201
ProjectionY(x_1)	113	201

(cont.)

Name	f_I	f_V
Binarize(x_1)	116	201
BinarizeInv(x_1)	117	201
Add(x_1)(x_2)	118	201
AlgebraicProd(x_1)(x_2)	119	201
AlgebraicSum(x_1)(x_2)	120	201
BoundedProd(x_1)(x_2)	121	201
Diff(x_1)(x_2)	122	201
LogicalProd(x_1)(x_2)	123	201
LogicalSum(x_1)(x_2)	124	201
Subtract(x_1)(x_2)	125	201

(d) Image operations using numeric parameters.

Name	f_I	f_V
HighPass(x_1)(x_2)	126	201
LowPass(x_1)(x_2)	127	201
ThresholdV(x_1)(x_2)	128	201
ThresholdInvV(x_1)(x_2)	129	201
BinarizeV(x_1)(x_2)	130	201
BinarizeInvV(x_1)(x_2)	131	201
AddV(x_1)(x_2)	132	201
ModAvgV(S_1)	133	201
MultiplyV(x_1)(x_2)	134	201
SubtractV(x_1)(x_2)	135	201
FillV(x_1)(x_2)	136	201

(e) Selection operators.

Name	f_I	f_V
Select(x_1)(x_2)	137	210
SelectV(x_1)(x_2)(x_3)(x_4)	138	213

表 B.2: 内部フィルタ.

(g) f_I

No.	f_I
101	$I' = I^1$
102	$I'_{i,j} = \max_{m,n=-1,0,1} \{I^1_{i+m,j+n}\}$
103	$I'_{i,j} = \min_{m,n=-1,0,1} \{I^1_{i+m,j+n}\}$
104	$I'_{i,j} = \text{median}_{m,n=-1,0,1} \{I^1_{i+m,j+n}\}$
105	$I' = I^1 * \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$
106	$I' = I^1 * \begin{pmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{pmatrix}$
107	$I' = I^1 * \begin{pmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{pmatrix}$
108	$I' = I^1 * \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$
109	$I' = I^1 * \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$
110	$I' = \frac{1}{2} (\text{No.108} + \text{No.109})$
111	$I'_{i,j} = 255 - I^1_{i,j} $
112	$I'_{i,j} = \frac{1}{n} \sum_{y=1}^n I^1_{i,y}$
113	$I'_{i,j} = \frac{1}{m} \sum_{x=1}^m I^1_{x,j}$
116	$I'_{i,j} = \begin{cases} 255 & I^1_{i,j} > t \\ 0 & I^1_{i,j} \leq t \end{cases} * 1$
117	$I'_{i,j} = \begin{cases} 0 & I^1_{i,j} > t \\ 255 & I^1_{i,j} \leq t \end{cases} * 1$
118	$I'_{i,j} = I^1_{i,j} + I^2_{i,j}$
119	$I'_{i,j} = \frac{1}{255} (I^1_{i,j} I^2_{i,j} + 1)$
120	$I'_{i,j} = I^1_{i,j} + I^2_{i,j} - \frac{1}{255} (I^1_{i,j} I^2_{i,j} + 1)$
121	$I'_{i,j} = I^1_{i,j} + I^2_{i,j} - 255$
122	$I'_{i,j} = I^1_{i,j} - I^2_{i,j} $
123	$I'_{i,j} = \min(I^1_{i,j}, I^2_{i,j})$
124	$I'_{i,j} = \max(I^1_{i,j}, I^2_{i,j})$
125	$I'_{i,j} = I^1_{i,j} - I^2_{i,j}$
126	*2
127	*3
128	$I'_{i,j} = \begin{cases} I^1_{i,j} & I^1_{i,j} > 255V^2 \\ 0 & I^1_{i,j} \leq 255V^2 \end{cases}$
129	$I'_{i,j} = \begin{cases} 0 & I^1_{i,j} > 255V^2 \\ I^1_{i,j} & I^1_{i,j} \leq 255V^2 \end{cases}$
130	$I'_{i,j} = \begin{cases} 255 & I^1_{i,j} > 255V^2 \\ 0 & I^1_{i,j} \leq 255V^2 \end{cases}$
131	$I'_{i,j} = \begin{cases} 0 & I^1_{i,j} > 255V^2 \\ 255 & I^1_{i,j} \leq 255V^2 \end{cases}$

(cont.)

No.	f_I
132	$I'_{i,j} = I^1_{i,j} + 255V^2$
133	$I'_{i,j} = I^1_{i,j} + 255V^2 - \frac{1}{mn} \sum_{x=1,y=1}^{m,n} I^1_{x,y}$
134	$I'_{i,j} = I^1_{i,j} \times 255V^2$
135	$I'_{i,j} = I^1_{i,j} - 255V^2$
136	$I'_{i,j} = 255V^2$
137	$I^1 = \begin{cases} I^1 & V^1_{i,j} \geq V^2 \\ I^2 & V^1_{i,j} < V^2 \end{cases}$
138	$I^1 = \begin{cases} I^1 & V^3_{i,j} \geq V^4 \\ I^2 & V^3_{i,j} < V^4 \end{cases}$

(h) f_V

No.	f_V
201	$V' = V^1$
202	$V' = \frac{1}{255} \frac{1}{mn} \sum_{x=1,y=1}^{m,n} I^1_{x,y}$
203	$V' = \frac{1}{255} \min_{x=1\dots m, y=1\dots n} I^1_{x,y}$
204	$V' = \frac{1}{255} \max_{x=1\dots m, y=1\dots n} I^1_{x,y}$
205	$V' = 1 - V^1$
206	$V' = V^1 + V^2$
207	$V' = V^1 \times V^2$
208	$V' = V^1 + V^2 - V^1 \times V^2$
209	$V' = V^1 - V^2 $
210	$V' = \max(V^1, V^2)$
211	$V' = \min(V^1, V^2)$
212	$V' = V^1 - V^2$
213	$V' = \begin{cases} V^1 & V^3 \geq V^4 \\ V^2 & V^3 < V^4 \end{cases}$

I' is an output. (I^n, V^n) is an n-th input.
 $I^n_{i,j}$ is a pixel (i, j) of the image I^n .

*1 t is determined by the Otsu's method [74].

*2 Apply DFT to I^1 , cut off elements which are less than the maximum frequency multiplied by V' , and apply IDFT to obtain I' . The pixel values in I' will be normalized in a range of 0 and 255.

*3 Apply DFT to I^1 , cut off elements which are more than the maximum frequency multiplied by V' , and apply IDFT to obtain I' . The pixel values in I' will be normalized in a range of 0 and 255.

謝辞

本論文は、筆者が社会人学生として北海道大学大学院情報科学研究科メディアネットワーク専攻博士後期課程在学中に、情報メディア学講座情報メディア環境学研究室において行った研究をまとめたものです。

本研究に関して終始あたたかく御指導、御鞭撻を頂きました本学山本強教授に心より感謝いたします。

論文の執筆にあたって具体的に親身に御指導頂きました本学土橋宜典准教授に深謝いたします。

本論文を御精読頂き有用な御助言、御指導を頂きました本学荒木健治教授、長谷山美紀教授、ならびに坂本雄児教授に、心より感謝いたします。

本論文の執筆にあたっては、本学の同窓であり親友でもある田森秀明博士に、いつも推敲や有用な指摘をして頂きました。深く感謝いたします。

本研究を社会人学生として行うにあたり、時間と費用の両面で多大なる御支援をいただいた、株式会社フィックスターズおよび同 CEO 三木聡氏に心より感謝いたします。

また、個々には挙げきれませんが多くの方に助けて頂き、本論文を書き上げることができました。感謝いたします。

そして、本論文の提出を最後まで、本当に長い間辛抱強く見守り支えてくれた、妻和代と愛娘紡季に深く感謝いたします。ありがとう。

参考文献

- [1] Jean Philippe Farrugia, Patrick Horain, Erwan Guehenneux, and Yannick Alusse. GPUCV: A framework for image processing acceleration with graphics processors. In *2006 IEEE International Conference on Multimedia and Expo, ICME 2006 - Proceedings*, Vol. 2006, pp. 585–588, 2006.
- [2] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, Vol. 1, pp. 321–331, 1988.
- [3] Stanley Osher and James A. Sethian. Fronts propagating with curvature dependent speed: Algorithms based on Hamilton-Jacobi formulations. *Journal of computational physics*, 1987.
- [4] Yuri Boykov and Gareth Funka-Lea. Graph Cuts and Efficient N-D Image Segmentation. *International Journal of Computer Vision*, Vol. 70, No. 2, pp. 109–131, November 2006.
- [5] Haifeng Chen and Peter Meer. Robust fusion of uncertain information. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, Vol. 35, pp. 578–586, 2005.
- [6] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, Vol. 1, , 2001.
- [7] David G. Lowe. Object recognition from local scale-invariant features. *Proceedings of the Seventh IEEE International Conference on Computer Vision*, Vol. 2, pp. 1150–1157 vol. 2, 1999.
- [8] Gabriella Csurka, Christopher R. Dance, Lixin Fan, Jutta Willamowski, and Cédric Bray. Visual Categorization with Bags of Keypoints. In *ECCV International Workshop on Statistical Learning in Computer Vision*, Prague, 2004.
- [9] Quoc V. Le, Marc’Aurelio Ranzato, Rajat Monga, Matthieu Devin, Kai Chen, Greg S. Corrado, Jeff Dean, and Andrew Y. Ng. Building high-level features

- using large scale unsupervised learning. In *International Conference in Machine Learning*, p. 38115, 2011.
- [10] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, Vol. 20, pp. 273–297, 1995.
- [11] 安居院 猛, 中嶋 正之. 画像工学の基礎. 昭晃堂, 1986.
- [12] 尾上 守男 (編). 画像処理ハンドブック. 昭晃堂, 1987.
- [13] 江尻 正員. 工業用画像処理. 昭晃堂, 1988.
- [14] 阿刀田 央一, 岡野 公一, 富澤 眞樹, 田村 仁. プログラミックフィルタ : 高性能低水準画像処理用非線形フィルタ作成のための一方式. 電子情報通信学会論文誌. D-II, 情報・システム, II-情報処理, Vol. 79, No. 6, pp. 1034–1045, 6 1996.
- [15] 坂上 勝彦, 田村 秀行. 処理モジュールの構造的知識を利用した画像処理プログラム自動生成システム. 情報処理学会論文誌, Vol. 26, No. 4, pp. 652–661, July 1985.
- [16] 松山 隆司, ビンセント ハング. 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, September 1985.
- [17] 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, February 1986.
- [18] 野村 浩, 長谷川 純一, 鳥脇 純一郎. 画像処理手順自動構成システム IMPRESS における特徴点抽出手順自動構成法の実現. 情報処理学会研究報告コンピュータビジョンとイメージメディア (CVIM) , Vol. 1988, No. 34(1988-CVIM-054), pp. 1–8, May 1988.
- [19] 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, Vol. J82-D2, No. 11, pp. 1982–1989, November 1999.
- [20] 濱田 敏弘, 清水 昭伸, 長谷川 純一, 鳥脇 純一郎. 画像処理エキスパートシステム IMPRESS における少数の設計標本からの手順構成に関する検討. 電子情報通信学会論文誌 D, Vol. J84-D2, No. 9, pp. 2029–2039, 2001.
- [21] 寺野 隆雄. エキスパートシステムはどうなったか? 計測と制御, Vol. 42, No. 6, p. 458, June 2003.
- [22] Toyoaki Nishida. Artificial intelligence research in the second half century. *Journal of Information Processing and Management*, Vol. 55, No. 7, pp. 461–471, October 2012.
- [23] 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 : 映像情報メディア, Vol. 53, No. 6, pp. 888–894, 6 1999.

-
- [24] 長尾 智晴. 進化的画像処理. 昭晃堂, 2002.
- [25] 土居 意弘, 山本 強. 遺伝的アルゴリズムによるフィルタツリー型画像処理の自動生成. In *DIA2006*, 2006.
- [26] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1 edition, 1992.
- [27] 土居 意弘, 土橋 宜典, 田森 秀明, 山本 強. 画像・数値ペアフィルタシーケンスの事例ベース自動生成. 画像電子学会誌 (ビジュアルコンピューティング論文特集号), Vol. 43, No. 3, pp. 309–317, 2014.
- [28] Munehiro Doi, Yoshinori Dobashi, Hideaki Tamori, and Tsuyoshi Yamamoto. Example-based automatic generation of image filters and classifiers based on image-value pairs. In *Proceedings of the 2014 3rd International Conference on Image, Vision and Computing*, No. IV036, 2014.
- [29] Munehiro Doi, Yoshinori Dobashi, Hideaki Tamori, and Tsuyoshi Yamamoto. Example-based automatic generation of image classifiers based on image-value trees. In *Proceedings of the 2015 Joint Conference of the International Workshop on Advanced Image Technology and the International Forum on Medical Imaging in Asia*, No. 273, 2015.
- [30] John H Holland. Building blocks, cohort genetic algorithms, and hyperplane-defined functions. *Evolutionary computation*, Vol. 8, No. 4, pp. 373–391, 2000.
- [31] 伊庭 斉志. 進化論的計算手法. オーム社, 2005.
- [32] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. October 1989.
- [33] 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, September 1997.
- [34] (社) 電気学会進化技術応用調査専門委員会. 進化技術ハンドブック 第 I 巻 基礎編. 近代科学社, 2010.
- [35] Dirk Thierens and David Goldberg. Elitist recombination: an integrated selection recombination GA. *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence*, pp. 508–512, 1994.
- [36] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transaction on Evolutionary Computation*, Vol. 6, No. 2, pp. 182–197, 2002.
- [37] Alex Fukunaga, Andre Stechert, and Darren Mutz. A Genome Compiler for High Performance Genetic Programming. *UNIVERSITY OF WISCONSIN*, pp. 86 –

- 94, 1998.
- [38] 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, pp. 17–26, 2001.
 - [39] Jilian F. Miller. An empirical study of the efficiency of learning boolean functions using a Cartesian Genetic Programming approach. *Proceedings of the Genetic and Evolutionary Computation Conference*, Vol. 2, pp. 1135–1142, 1999.
 - [40] Terence Soule and James A. Foster. Removal bias: a new cause of code growth in tree based evolutionary programming. In *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360)*, pp. 781–786, 1998.
 - [41] Mark Wineberg and Franz Oppacher. The benefits of computing with introns. pp. 410–415, July 1996.
 - [42] William B. Langdon, Terence Soule, Riccardo Poli, and James A. Foster. The Evolution of Size and Shape. In *Advanced in Genetic Programming 3*, chapter 8. 1999.
 - [43] PLM Wong. Removing redundancy and reducing fitness evaluation costs in genetic programming. 2008.
 - [44] Stephen Dignum and Riccardo Poli. Operator equalisation and bloat free GP. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Vol. 4971 LNCS, pp. 110–121, 2008.
 - [45] Riccardo Poli. A simple but theoretically-motivated method to control bloat in genetic programming. *Genetic Programming*, Vol. 2610, pp. 204–217, 2003.
 - [46] Edwin de Jong, Richard Watson, and Jordan Pollack. Reducing bloat and promoting diversity using multi-objective methods, 2001.
 - [47] Wolfgang Banzhaf. Genetic programming for pedestrians. *Proceedings of the 5th International Conference on Genetic Algorithms*, p. 628, 1993.
 - [48] Julian Miller. What bloat? Cartesian Genetic Programming on Boolean problems. *2001 Genetic and Evolutionary Computation Conference Late Breaking Papers*, pp. 295–302, 2001.
 - [49] Qingfu Zhang and Hui Li. MOEA/D: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation*, Vol. 11, No. 6, pp. 712–731, 2007.
 - [50] Aimin Zhou, Bo-Yang Qu, Hui Li, Shi-Zheng Zhao, Ponnuthurai Nagaratnam

- Suganthan, and Qingfu Zhang. Multiobjective evolutionary algorithms: A survey of the state of the art, 2011.
- [51] Altannar Chinchuluun and Panos M. Pardalos. A survey of recent developments in multiobjective optimization, 2007.
- [52] J. David Schaffer. Multiple objective optimization with vector evaluated genetic algorithms. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pp. 93–100, 1985.
- [53] Carlos M. Fonseca and Peter J. Fleming. Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. *Genetic Algorithms: Proceedings of the Fifth International Conference*, pp. 416–423, 1993.
- [54] N. Srinivas and Kalyanmoy Deb. Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms. In *Evolutionary Computation*, Vol. 2, pp. 221–248. 1994.
- [55] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. SPEA2: Improving the strength pareto evolutionary algorithm. Technical report, Computer Engineering and Networks Laboratory (TIK), 2001.
- [56] Abdullah Konak, David W. Coit, and Alice E. Smith. Multi-objective optimization using genetic algorithms: A tutorial. *Reliability Engineering & System Safety*, No. 91, pp. 992–1007, 2006.
- [57] Carlos A. Coello Coello. A Comprehensive Survey of Evolutionary-Based Multiobjective Optimization Techniques. *Knowledge and Information Systems*, Vol. 1, pp. 269–308, 2013.
- [58] 電子情報通信学会. 多目的最適化. 電子情報通信学会『知識の森』, 3-1, S3 群 – 4, p. 15. 電子情報通信学会, 2012.
- [59] Kalyanmoy Deb and Himanshu Jain. An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving Problems With Box Constraints. *IEEE Transactions on Evolutionary Computation*, Vol. 18, No. 4, pp. 577–601, August 2014.
- [60] 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: 映像情報メディア, Vol. 59, No. 11, pp. 1687–1693, 11 2005.
- [61] 白川 真一, 荻野 慎太郎, 長尾 智晴. Genetic Image Network による画像変換の自動構築. 情報処理学会論文誌. 数理モデル化と応用, Vol. 48, No. 19, pp. 117–126, 12 2007.
- [62] 白川 真一, 中山 史朗, 矢田 紀子, 長尾 智晴. Genetic Image Network に基づく画

- 像分類アルゴリズムの自動構築. 人工知能学会論文誌, Vol. 25, No. 2, pp. 262–271, 2010.
- [63] 中山 史朗, 白川 真一, 矢田 紀子, 長尾 智晴. 弱識別器に Genetic Image Network を用いたアンサンブル画像分類法. 電気学会論文誌C (電子・情報・システム部門誌), Vol. 131, No. 5, pp. 958–965, 2011.
- [64] Krzysztof Krawiec and Bir Bhanu. Coevolution and Linear Genetic Programming for Visual Learning. In *Genetic and Evolutionary Computation – GECCO-2003*, Vol. 2723, pp. 332–343, July 2003.
- [65] Simon Harding, Vincent Graziano, Jürgen Leitner, and Jürgen Schmidhuber. MT-CGP: Mixed Type Cartesian Genetic Programming. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference - GECCO '12*, p. 751, New York, New York, USA, July 2012. ACM Press.
- [66] David J Montana. Strongly typed genetic programming. *Evolutionary computation*, Vol. 3, No. 2, pp. 199–230, 1995.
- [67] David Andre and Astro Teller. A study in program response and the negative effects of introns in genetic programming. In *Proceedings of the 1st annual conference on genetic programming*, pp. 12–20, 1996.
- [68] Telecommunication Standardization Sector of ITU(ITU-T). *Objective perceptual assessment of video quality: Full reference television*. International Telecommunication Union (ITU), 2004.
- [69] E.M. Kasprzak and K.E. Lewis. Pareto analysis in multiobjective optimization using the collinearity theorem and scaling method. *Structural and Multidisciplinary Optimization*, Vol. 22, No. 3, pp. 208–218, 2001.
- [70] Chih-Chung Chang and Chih-Jen Lin. LIBSVM. *ACM Transactions on Intelligent Systems and Technology*, Vol. 2, No. 3, pp. 1–27, April 2011.
- [71] Stefan Bleuler, Martin Brack, Lothar Thiele, and Eckart Zitzler. Multiobjective genetic programming: reducing bloat using SPEA2. *Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No.01TH8546)*, pp. 536–543, 2001.
- [72] Vivek Sarkar, Mauricio J. Serrano, and Barbara B. Simons. Register-sensitive selection, duplication, and sequencing of instructions. In *Proceedings of the 15th international conference on Supercomputing - ICS '01*, pp. 277–288, New York, New York, USA, June 2001. ACM Press.
- [73] Penousal Machado and Amílcar Cardoso. Speeding up Genetic Programming.

In *Proceedings of the Second International Symposium on Artificial Intelligence, Adaptive Systems (CIMAFA - 99)*, 1999.

- [74] Nobuyuki Otsu. A Threshold Selection Method from Gray-Level Histograms, 1979.

学術研究論文等研究業績一覧

学会誌

- 土居 意弘, 土橋 宜典, 田森 秀明, 山本 強, ” 画像・数値ペアフィルタシーケンスの事例ベース自動生成,” Journal of the Institute of Image Electronics Engineers of Japan, Vol. 43, No. 3, pp. 309-317, 2014.

国際会議

- M. Doi, Y. Dobashi, H. Tamori, and T. Yamamoto, ”Example-Based Automatic Generation of Image Filters and Classifiers based on Image-Value Pairs,” 2014 3rd International Conference on Image, Vision and Computing, No. IV036, 2014.
- M. Doi, Y. Dobashi, H. Tamori, and T. Yamamoto, ”Example-Based Automatic Generation of Image Classifiers Based on Image-Value Trees,” The 2015 Joint Conference of the International Workshop on Advanced Image Technology and the International Forum on Medical Imaging in Asia, No. 273, 2015.

学術講演

- 土居 意弘, 山本 強, ” 知識フィードバックによる GA 自動画像処理シーケンス生成に関する一検討,” DIA2007, 2007
- 土居 意弘, ” 遺伝的アルゴリズムによるフィルタツリー型画像処理の自動生成,” DIA2006, 2006

索引

B

bloat → ブロート

C

Cartesian Genetic Programming 33
CGP → Cartesian Genetic Programming
child → 子
chromosome 23
constant → 定数

D

DAG → 無閉路有向グラフ
directed acyclic graph → 無閉路有向グラフ

E

early convergence 27
edge → エッジ
Elitist Recombination 27
ER 27

F

Fast Non-Dominated Sorting 28
fitness → 適応度
fitness function → 適応度関数
full 31
function → 関数

G

GA → 遺伝的アルゴリズム
Genetic Algorithm → 遺伝的アルゴリズム
Genetic Programming → 遺伝的プログラミング
genotype 23
GP → 遺伝的アルゴリズム
Graphic Processing Unit 13
grow 31
GTYPE 23

I

Image-Value グラフ 83

Image-Value ツリー 43
Image-Value フィルタ 42
Image-Value ペア 13, 18, 41
individual 23
intron → イントロン
local intron 45
semantical intron 45
structural intron 45

L

lethal gene 23
LGP → Linear Genetic Programming
Linear Genetic Programming 33
Lisp 29
locus 23

M

MGG → Minimal Generation Gap, 27
Minimal Generation Gap 22, 27

N

node → ノード
Non-Dominated Sorting GA II 28
non-terminal node → 非終端ノード
NSGA-II 28

P

parent → 親
phenotype 23
PTYPE 23

R

ramped half and half 31
root → ルート

S

SGA 27
SIMD → Single Instruction Multiple Data

Simple GA 22, 27
Single Instruction Multiple Data 13
string 23

T

terminal node → 終端ノード

あ

遺伝子座 23
遺伝的アルゴリズム 21, 21
遺伝的操作 25
遺伝的プログラミング 16, 21, 28
イントロン 21, 32
意味的イントロン 45
局所的イントロン 45
構造的イントロン 45
エキスパートシステム 14
エッジ 28
エリート戦略 24
親 28

か

関数 29
記号 29
木構造 28
記号列 23
強化型背景除去フィルタ 69
子 28
交叉 25
高速非優越ソート 28
個体 23

さ

シェアリング 25
初期個体群 24
初期収束 27
世代交代の連続化 27
世代交代モデル 22
染色体 23
選択 24

た

多目的遺伝的アルゴリズム	21
多目的遺伝的プログラミング	18
多目的最適化	21, 32, 34
多様性	24, 25
致死遺伝子	23
頂点	→ ノード
定数	29
適応度	16, 21
適応度関数	16, 22, 31
トーナメント方式	24

突然変異	25
------	----

な

ノード	28
終端ノード	28
非終端ノード	28

は

バイナリ記号列	23
パレート最適解	34
ブロート	17, 18, 21, 31, 39

平衡木	80
辺	→ エッジ

ま

マルチコア	13
無閉路有向グラフ	83

ら

ルート	28
ルーレット方式	24