社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

Macaw：遺伝的アルゴリズムを用いた自動プログラム修正の進化過程の可視化ツール

富田　裕也†　肥後　芳樹†　柗本　真佑†　楠本　真二†

† 大阪大学大学院情報科学研究科
〒 565–0871 大阪府吹田市山田丘 1–5
E-mail: †{y-tomida,higo,shinsuke,kusumoto}@ist.osaka-u.ac.jp

あらまし　自動プログラム修正の手法の 1 つに遺伝的アルゴリズムを用いる手法がある．この手法は修正対象のプログラムに対して，すべてのテストケースに成功するプログラムが得られるまでプログラム文の挿入，削除および置換を繰り返し行う．この手法は探索空間が非常に大きいために，解を得るまでに非常に多くの変異プログラムを生成することが多い．そのため，解を得るまでの過程，すなわちプログラムの進化過程の解析が難しい．本研究では，プログラムの進化過程を可視化する手法を提案し，可視化ツールを実装した．さらに，実装したツールが解析の助けになるかを評価するために，遺伝的アルゴリズムに基づいた自動プログラム修正ツールである kGenProg を用いて被験者実験を行った．実験の結果，プログラムの進化過程の可視化は自動プログラム修正ツールの開発者にとって有用であることを明らかにした．

キーワード　自動プログラム修正，デバッグ，可視化，遺伝的アルゴリズム

1. まえがき

ソフトウェア開発においてデバッグは多大な労力を必要とする作業であり，開発工数の半数以上を占めるといわれている [1]．そのため，デバッグの支援はソフトウェア開発の効率化やソフトウェアの信頼性および安全性の向上に有益である．デバッグの労力を削減するために，デバッグの支援に関する研究が数多く行われている [2]〜[5]．

デバッグの支援に関する研究の 1 つに自動プログラム修正がある．自動プログラム修正は修正対象のプログラム，テスト集合および欠陥の所在を入力として，全てのテストケースに成功するプログラムを出力する．自動プログラム修正の手法の 1 つに遺伝的アルゴリズム [6] を用いた手法がある．この手法は，修正対象のプログラムに対して遺伝的アルゴリズムを適用することで欠陥の修正を試みる．

遺伝的アルゴリズムを用いた自動プログラム修正の手法を実装したツールの 1 つに GenProg [7] がある．GenProg およびその関連手法では，欠陥を修正できるまでプログラムを変更し続ける．この手法は解を得る過程（以降，進化過程という）で多くのプログラムを生成する．解を得る過程を解析することで自動プログラム修正ツールの改良やパラメータの調整に関する知見が得られる．しかし，GenProg およびその関連手法では解を得るまでに非常に多くのプログラムを生成するため，プログラムの進化過程の解析が難しい．

この問題を解決するために本研究では，GenProg およびその関連手法の解を得る過程を可視化する手法を提案する．また，実際のソフトウェア開発の過程で生じた欠陥に対して自動プログラム修正を行ったときのプログラムの進化過程を可視化し，本研究で提案する可視化手法が有用なのかを定性的に評価した．

2. 研究背景

2.1 欠陥限局

欠陥限局は欠陥を含むプログラム中の欠陥の位置を推測する手法である [8]．以降，本研究では 1 つでも実行に失敗したテストケースが存在するプログラムを欠陥を含むプログラムとする．

2.2 自動プログラム修正

自動プログラム修正は欠陥を含むプログラムとテスト集合を入力として，修正済みのプログラムを出力する手法である．修正済みのプログラムとは，テスト集合に含まれる全てのテストケースに成功するプログラムである．

2.3 GenProg

自動プログラム修正の手法の 1 つに遺伝的アルゴリズム [6] を用いた GenProg [7] がある．遺伝的アルゴリズムは解の候補を生物の個体に見立て，遺伝的操作と呼ばれる操作を繰り返し行うことで，最適解に近い解（以降，近似解という）を探索するアルゴリズムである．GenProg では個体はプログラム，近似解は修正済みのプログラムとなる．各個体はプログラム文や文字列ではなく，変更を加えた位置と変化の対象となったプログラム文（以降，塩基という）の列で表現される．

GenProg の流れを図 1 に示す．まず，GenProg は欠陥を含むプログラム（以降，初期個体という）に対して推定した欠陥箇所に変更を加え，個体を複数生成する．次に，生成した個体の評価

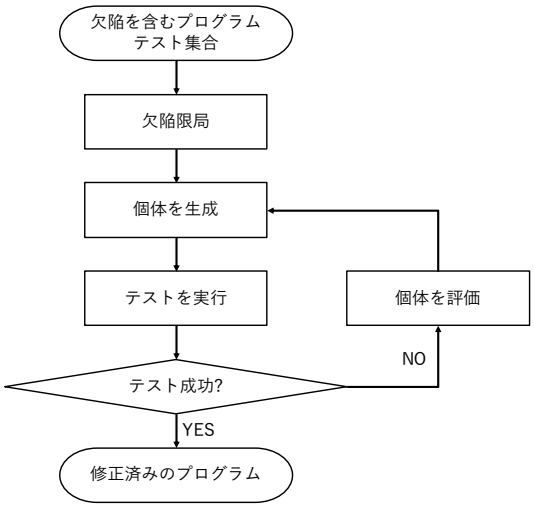


図 1: GenProg の流れ

を行う．個体の評価には適応度という値を用いる．適応度は個体の良さを表しており，一般的に適応度の計算にはテストの通過率が用いられる．GenProg は最適解が得られるまで個体の生成および評価を繰り返す．1 回の繰り返しのことを世代という．

個体の生成処理で行われる操作を以下に示す．

選択　現世代の個体に対してテストを実行する．それらの個体から成功したテストケースが多い個体を一定数取り出す．取り出された個体は次世代の変異や交叉の対象になる．選択によって，個体の多様性を保つために現世代よりも前の世代で生成された個体が選択されることがある．本研究では現世代よりも前の世代で生成された個体を選択することをコピーという．

変異　選択によって取り出された個体の欠陥箇所に変更を加え，個体を生成する．加える変更はプログラム文の挿入・削除・置換のいずれかである．挿入や置換は代入文や if 文などのプログラム文を用いる．挿入や置換で用いるプログラム文は初期個体に含まれるプログラム文である．

交叉　前世代の 2 つの個体から，各々の個体の塩基を持つ個体を生成する．代表的な交叉の手法は一点交叉，一様交叉およびランダム交叉である．

2.4　遺伝的アルゴリズムを用いた自動プログラム修正ツールの問題点

遺伝的アルゴリズムに用いた自動プログラム修正ツールのアルゴリズムを改良するためにはプログラムの進化過程を解析する必要がある．しかし，解を探索する過程で非常に多くの個体を生成することが多く，出力されるログが非常に大きくなりやすい．そのため，遺伝的アルゴリズムを用いた自動プログラム修正ツールのプログラムの進化過程は解析が難しい．

3.　提 案 手 法

3.1　概　　要

本研究ではプログラムの進化過程の解析が難しいという問題を解決するために，プログラムの進化過程の可視化をする手法を提案する．提案手法は可視化によってプログラムの進化過程を一目で把握でき，解析が容易にできることを目的としている．

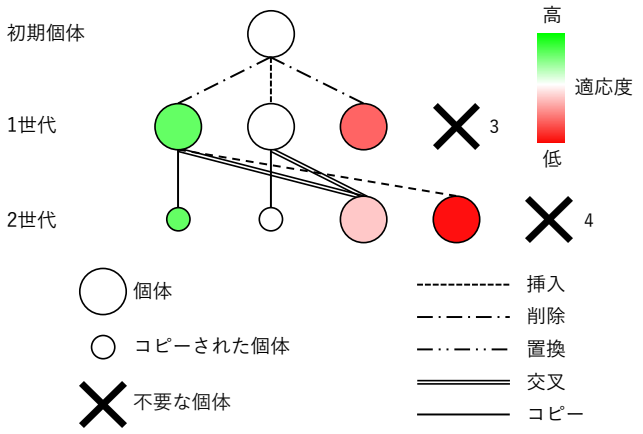


図 2: 可視化のイメージ

3.2　可視化の方針

プログラムの進化過程を図 2 のような木構造で可視化する．ノードは個体に対応しており，横一列に並んでいる個体群はそれらが同世代の個体であることを表す．各個体がどの世代で生成されたのかを明確にするためには，コピーされた個体とその他の操作で生成された個体を異なる方法で表現する必要がある．提案手法では，ノードの大きさでそれらの違いを表現する．生成された個体は円形のノードで表現し，コピーされた個体は一回り小さい円形のノードで表現する．斜め十字のノードはその世代で生成された個体の内，コンパイルに失敗した，またはその世代よりも前の世代で生成された個体と全く同じ個体（以降，不要な個体とする）の集合に対応している．このノードの右に不要な個体の集合の大きさを表示する．円形と一回り小さい円形のノードの色は対応する個体の適応度によって変化する．初期個体の適応度と等しい場合は白色になり，大きい場合は 1 に近いほど緑色に近づき，小さい場合は 0 に近いほど赤色に近づく．

エッジは挿入，削除，置換，コピーおよび交叉を表現する．また，テスト結果やソースコード差分のような各個体の詳細な情報は利用者が必要とする場合に表示する．

3.3　実　　装

プログラムの進化過程の可視化をするツールとして Macaw（Macaw is an ARP Comprehension Assist tool With visualization）を開発した．Macaw は JavaScript で実装しており，Web ブラウザ上で動作する．図 3 は Macaw でオープンソースソフトウェアの欠陥の修正過程を可視化した結果である．

4.　ユースケース

4.1　自動プログラム修正ツールの改良

自動プログラム修正ツールの開発者（以降，開発者という）が自動プログラム修正ツールの操作の変更を行う場面を考える．開発者は Macaw を使うことで変更した操作が変更前よりも有効に働いているか判断できる．例えば，変更した操作に対応するエッジが変更前よりも多くの緑色のノードに繋がっていれば，変更後の操作は変更前よりも優れているといえる．さらに，ノードをクリックすることで差分を表示できるため，開発者はより詳細な分析ができる．

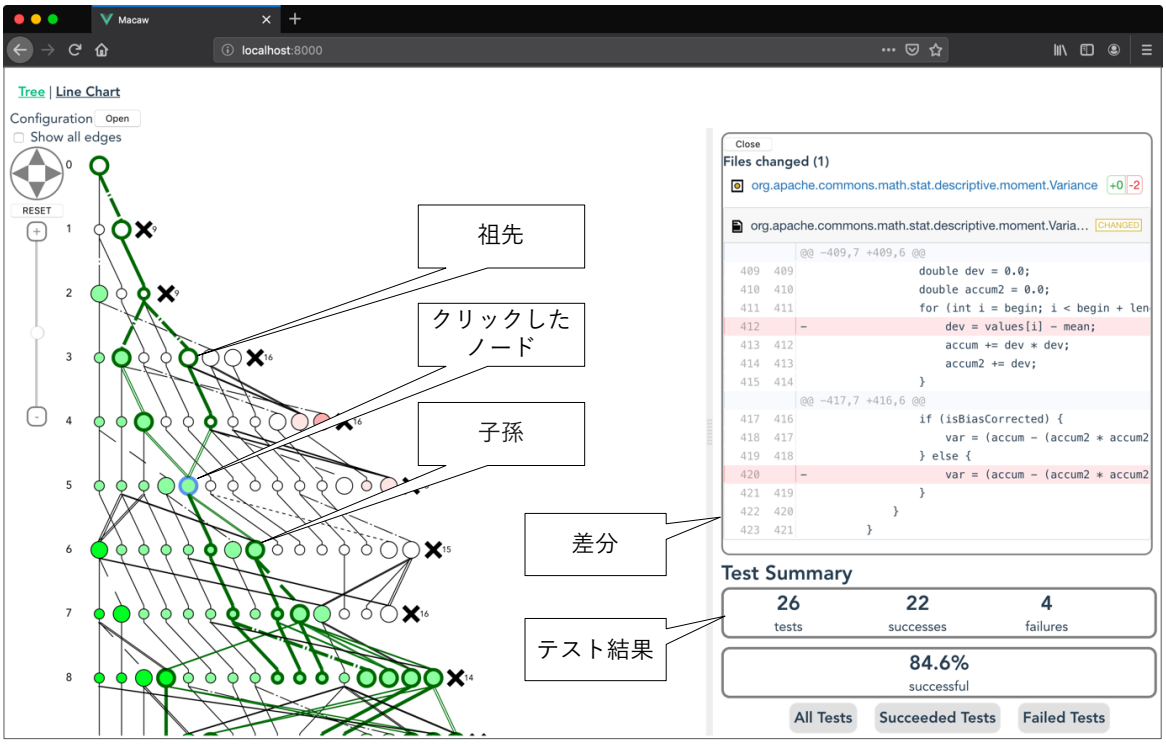


図 3: 可視化の例

4.2 自動プログラム修正ツールのパラメータ調整

自動プログラム修正ツールの利用者（以降，利用者という）が最大世代数，各世代で変異や交叉で生成する個体の数，コピーする個体の数などのパラメータを変えながら自動プログラム修正ツールを実行する場面を考える．例えば，適応度の可視化によって世代を経ても適応度が変化しない結果が得られたとする．このとき，利用者は世代数を増やしても欠陥を修正できないと判断し，最大世代数を減らし各世代において生成する個体の数を増やすようなパラメータの調整ができる．また，適用した操作の可視化によって交叉によって生成された個体の大部分がコンパイルに失敗するという結果が得られたとする．このとき，利用者は修正済みプログラムを得るためには交叉は不要と判断し，交叉で個体を一切生成しないようにパラメータを調整できる．

5. 被験者実験

5.1 被 験 者

本実験の被験者は大阪大学大学院情報科学研究科に所属する教員 2 名，同研究科に所属する修士の学生 5 名，大阪大学基礎工学部情報科学科に所属する学部生 1 名の計 8 名である．また，全ての被験者は kGenProg [9] の開発者である．

5.2 実 験 対 象

本実験の可視化対象は Apache Commons Math というオープンソースソフトウェア（以降，OSS という）の開発過程で生じた欠陥に対して，kGenProg を適用したときのプログラムの進化過程である．これらの欠陥は Defects4j [10] という欠陥のデータセットより取得した．表 1 に示す欠陥に対して kGenProg を適用したときの結果を対象にして実験を行なった．

5.3 実 験 手 順

被験者実験は以下の流れで実施した．

（1）Macaw の説明：Macaw の使い方を被験者に説明する．

（2）操作の練習：簡単なプログラムの欠陥を kGenProg で修正したときの可視化結果に対して Macaw を操作してもらい，ツールの使い方を習得してもらう．

（3）議論：被験者に 5.2 節で述べた実験対象に対して可視化から得られたことについて議論してもらう．各議論は意見が出

表 1: 実験対象にした欠陥の一覧

欠陥 ID	失敗した テストケースの数	最大世代数	変異で生成する 個体の数	交叉で生成する 個体の数	選択する 個体の数	交叉の手法
Math95	1	300	10	0	5	-
	1	40	75	0	5	-
Math43	6	200	10	5	5	ランダム交叉
	6	40	55	5	5	ランダム交叉
Math2	1	400	2	40	20	ランダム交叉
	1	400	2	40	20	一様交叉
	1	400	2	40	20	一点交叉

```
public SumOfLogs() {
  value = 0d;
  n = 0;
+ var = evaluate(values, m, begin, length);
}
```

図 4: 変数を使用したプログラム文が挿入された例

なくなるまで行う．

5.4 実験結果

5.4.1 kGenProg のアルゴリズムの改良案

交叉の親の選択方法 Math43 ではテスト結果が異なる個体を用いた交叉によって，解が得られた．そのため，ランダムに交叉の親を選ぶのではなく，テスト結果が大きく異なる 2 つの個体を交叉の親に選んだ方が交叉の性能が向上するのではないのかという意見が得られた．

適応度の表現 kGenProg は適応度を適切に表現していないため遺伝的アルゴリズムの利点を活かした動作をできていないという意見が得られた．kGenProg では適応度をテストの成功率と定義している．しかし，Defects4J に記録されている多くのバグは失敗するテストケースの数が少ない．そのため，kGenProg は解と入力されたプログラムの間を適応度を適切に表現できず，個体が徐々に改善するという遺伝的アルゴリズムの利点を活かした動作をしていないという意見が得られた．

挿入操作の改良 変数を使用したプログラム文を挿入するときに，その変数の宣言文も同時に挿入する操作を実装すれば，効率的に解を探索できるのではないのかという改良案が得られた．挿入操作は図 4 のように変数を使用する文が挿入することが多い．さらに，他の操作と比べて挿入操作は不要な個体を多く生成している．そのため，挿入によって生成された個体は未定義変数を使用したプログラム文が挿入される可能性が高く，コンパイルが失敗しやすいと被験者が考えた結果得られた改良案である．

選択の改良 不要な個体を多く生成した個体を選択しないという改良案が得られた．実験の結果より不要な個体を多く生成した個体からはまだ発見されていない個体が生成されにくい．そのため，世代が進むにつれて，不要な個体の数が増え，個体の多様性が乏しくなると被験者が考えた結果得られた改良案である．

5.4.2 kGenProg の欠陥

本実験中に 2 つの kGenProg の欠陥を発見した．1 つ目は選択の欠陥である．個体を選択するときに，適応度が同じならば，意図せず古い世代の個体が優先的に選択される欠陥である．2 つ目は交叉の欠陥である．交叉は 2 つの個体を用いるべきである．しかし，1 つの個体から交叉が行われることがあった．図 5 中の赤線で囲まれたノードは交叉の欠陥の例である．

5.5 考　　察

この節では前節で得られた意見が得られた理由を考察する．

5.5.1 kGenProg のアルゴリズムの改良案に関する意見

交叉の親の選択方法 適用した操作の可視化によって，被験者は行われた操作が一目で判断できる．被験者が交叉によって得られた解の親に当たる個体のテスト結果を比べた結果，テスト結果

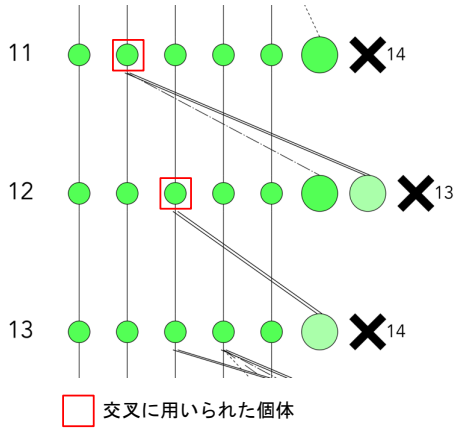


図 5: 交叉に用いる個体が 1 つだけの例

が異なる 2 つの個体を交叉の親に選んだ方が交叉の性能が向上するのではないのかという意見が得られたと考えられる．

適応度の表現 適応度の可視化によって，被験者は適応度の変化を把握しやすくなる．被験者が多くのノードの色が白色であったことを発見し，kGenProg は適切に適応度を表現できていないのではないのかという意見が得られたと考えられる．

挿入の改良 適用した操作の可視化とソースコードの差分によって，被験者は操作によって変化した部分を把握しやすくなる．被験者が挿入によって生成された個体のソースコードの差分を表示すると，変数を使用した文が多く表示されていたことを発見した．その結果，変数宣言文も同時に挿入するという改良案が得られたと考えられる．

選択の改良 被験者がコピーされた個体から生成された個体に注目したところ，複数回コピーされた個体からはまだ発見されていない個体が生成されにくいことを発見した．その結果，複数回コピーされた個体を選択の対象外にするという改良案が得られたと考えられる．

5.5.2 kGenProg の欠陥に関する意見

コピーされた個体を生成された個体よりも一回り小さくした．そのため，個体を選択するときに，適応度が同じならば，古い世代の個体が優先的に選択される欠陥が発見されたと考えられる．適用した操作の可視化によって被験者がどの操作が適用されたか一目で判断できる．そのため，交叉に用いる個体が 1 つだけである欠陥が発見されたと考えられる．

5.5.3 全　　体

kGenProg のアルゴリズムの改良案が得られたことから，プログラムの進化過程の可視化は開発者にとって有用であると結論づけることができる．特に，kGenProg のアルゴリズムの改良案が得られたことから，プログラムの進化過程の可視化は遺伝的アルゴリズムに基づいた自動プログラム修正ツールの改良に繋がるといえる．

6. 性能評価実験

Macaw はインタラクティブなツールである．Macaw の利用者が効率的にプログラムの進化過程を分析するために，描画までにかかる時間やユーザの操作に反応するまでの時間は非常に重

要である．そのため，表 2 に示す性能を持つ計算機を使用して Macaw の性能を評価する実験を行った．

6.1 実験設計

本実験では，Joda-Time，JFreeChart，Apache Commons Math の欠陥に対して表 3 に示すパラメータ設定で kGenProg を適用した結果を使用する．解を発見したとしても kGenProg は停止せず制限時間まで動作し続けるように kGenProg のパラメータを設定した．この設定で各 OSS の欠陥に対して kGenProg を適用したときの結果を表 4 に示す．本実験では 2 種類の時間を計測した．

初期化時間　進化過程を記録したファイルの読込みを開始してから画面が表示されるまでの時間

応答時間　ズームあるいはスクロールをしてから再び画面が描画されるまでの時間

いずれの時間も 5 回ずつ実行し，その平均を計測値とする．また，初期化時間の計測時には以下の時間も計測した．

ファイル読込み時間　入力ファイルの読込みにかかる時間

計算時間　入力ファイルの読込みが終わってから，ノードやエッジの座標，色および形が決定されるまでの時間

描画時間　ノードやエッジの座標，色および形が決定されてから，画面に表示されるまでの時間

実験で用いた計算機の性能およびブラウザを表 2 に示す．

6.2 性能目標

利用者が違和感なく操作ができていると感じることができる時間は最長で 100 ミリ秒，注意を維持できる時間は最長で 10 秒と言われている [11]．これらの値に基づき，Macaw の性能目標として初期化時間を 10 秒，応答時間を 100 ミリ秒に設定した．

6.3 実験結果

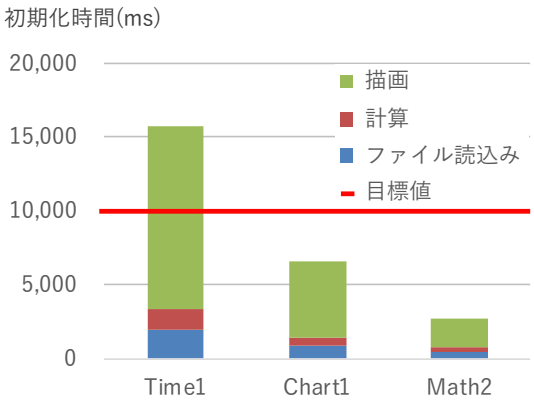
初期化時間および応答時間の計測結果を図 6a，6b に示す．グラフの縦軸はミリ秒単位の時間，横軸は実験対象にした欠陥 ID，赤色の線は目標値を示している．図 6a から Chart1 と Math2 の初期化時間は目標値を達成できていることがわかる．図 6b から Time1 と Chart1 の応答時間は目標値を達成できておらず，再描画に目標値の 3〜12 倍の時間がかかっていることがわかる．以上のことから Macaw はズームやスクロールの実行性能が悪いため，応答時間の改善が必要であることがわかる．

表 2: 計測に用いた計算機の性能

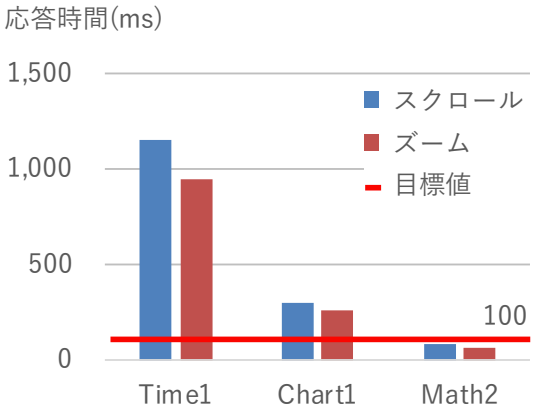
OS	Windows10
CPU	Intel Xeon E5-2603v3 x2
GPU	NVIDIA Quadro K2200
メモリ	32GB

表 3: kGenProg の設定

変異で生成する個体の数	10
交叉で生成する個体の数	5
選択する個体の数	5
交叉の手法	ランダム交叉
制限時間（分）	30



(a) 初期化時間



(b) 応答時間

図 6: 計測結果

6.4 考察

初期化時間に注目すると，描画時間が初期化時間の 70%以上を占めている．その原因として Macaw はノードやエッジを SVG を用いて表現しており，初期化時に全ての SVG 要素の構築および描画をしていることが考えられる．そのため，画面外の SVG 要素の構築を行わないように Macaw を改良すれば，大幅に初期化時間を削減できると考えられる．さらに，同じ理由でズーム・スクロールの応答時間も改善できると考えられる．

7. 妥当性への脅威

7.1 被験者実験

実験対象ソフトウェアは Apache Commons Math のみである．ソフトウェアの欠陥によって，進化過程も変わるため，他のソフトウェアで実験を行えば，異なる意見が得られる可能性がある．

7.2 性能評価実験

実験対象にするソフトフェアや実行時のパラメータによって，

表 4: kGenProg を適用した結果

欠陥 ID	到達世代数	生成した個体の数
Time1	464	6,885
Chart1	186	2,775
Math2	49	930

差分のデータが大きく変化するため，読込み時間が変化する可能性がある．

8. あとがき

本研究では，遺伝的アルゴリズムに基づいた自動プログラム修正の可視化を提案した．提案手法が有用であるか確認するために被験者実験を行った．被験者実験によって，自動プログラム修正ツールのアルゴリズムの改良案が得られた．よって，進化の過程の可視化は自動プログラム修正ツールの開発者にとって有用であると結論づけることができる．

今後の課題として，可視化する情報を増やすことが考えられる．例えば，以下に示す拡張が考えられる．

操作サマリ　操作別にその操作が行われた回数や生成した不要な個体の数を表示する．この拡張によってどの操作が有用かどうか理解しやすくなると考えられる．

個体とその親との差分　利用者が選択した個体とその親に当たる個体との差分，すなわち操作によってプログラムが変更された部分を表示する．この拡張によって行われた操作についてより理解しやすくなると考えられる．

進化の履歴　利用者が選択した個体に対して適用された操作およびプログラムの変更箇所を一覧で表示する．この拡張からどの操作によって解が得られたかがより理解しやすくなると考えられる．

現在，Macaw は kGneProg の進化過程しか可視化できない．他の遺伝的アルゴリズムに基づいた自動プログラミング修正ツールの進化過程を可視化できるようにすることも今後の課題である．また，Macaw の性能を上げることも今後の課題として考えられる．

謝辞　本研究は，日本学術振興会科学研究費補助金基盤研究（B）（課題番号：17H01725）の助成を得て行われた．

文　献

[1] Tom Britton, Lisa Jeng, Graham Carver, Paul Cheak, and Tomer Katzenellenbogen. Reversible Debugging Software: Quantify the time and cost saved using reversible debuggers, 2013.

[2] L. A. Clarke. A System to Generate Test Data and Symbolically Execute Programs. *IEEE Transactions on Software Engineering*, Vol. SE-2, No. 3, pp. 215–222, Sep. 1976.

[3] Carlos Pacheco, Shuvendu K. Lahiri, Michael D. Ernst, and Thomas Ball. Feedback-Directed Random Test Generation. In *Proceedings of the 29th International Conference on Software Engineering*, pp. 75–84, 2007.

[4] Daniele Zuddas, Wei Jin, Fabrizio Pastore, Leonardo Mariani, and Alessandro Orso. MIMIC: Locating and understanding bugs by analyzing mimicked executions. *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, pp. 815–825, Sep. 2014.

[5] James A. Jones and Mary Jean Harrold. Empirical Evaluation of the Tarantula Automatic Fault-localization Technique. In *Proceedings of the 20th ACM/IEEE International Conference on Automated Software Engineering*, pp. 273–282, 2005.

[6] K. F. Man, K. S. Tang, and S. Kwong. Genetic algorithms: concepts and applications [in engineering design]. *IEEE Transactions on Industrial Electronics*, Vol. 43, No. 5, pp. 519–534, Oct. 1996.

[7] Claire Le Goues, Michael Dewey-Vogt, Stephanie Forrest, and Westley Weimer. A Systematic Study of Automated Program Repair: Fixing 55 out of 105 Bugs for $8 Each. In *Proceedings of the 34th International Conference on Software Engineering*, pp. 3–13, 2012.

[8] Rui Abreu, Peter Zoeteweij, Rob Golsteijn, and Arjan J. C. van Gemund. A Practical Evaluation of Spectrum-based Fault Localization. *J. Syst. Softw.*, Vol. 82, No. 11, pp. 1780–1792, Nov. 2009.

[9] Yoshiki Higo, Shinsuke Matsumoto, Ryo Arima, Akito Tanikado, Keigo Naitou, Junnosuke Matsumoto, Yuya Tomida, and Shinji Kusumoto. kGenProg: A High-performance, High-extensibility and High-portability APR System. In *the 25th Asia-Pacific Software Engineering Conference*, pp. 697–698, Dec. 2018.

[10] René Just, Darioush Jalali, and Michael D. Ernst. Defects4J: A Database of Existing Faults to Enable Controlled Testing Studies for Java Programs. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, pp. 437–440, 2014.

[11] P. Pirolli. Powers of 10: Modeling Complex Information-Seeking Systems at Multiple Scales. *Computer*, Vol. 42, No. 3, pp. 33–40, Mar. 2009.