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.

自動プログラム生成における プログラム依存を考慮した交叉法の提案と評価

渡辺　大登† 　柗本　真佑† 　肥後　芳樹† 　楠本　真二†

倉林　利行†† 　切貫　弘之†† 　丹野　治門††

† 大阪大学大学院情報科学研究科
†† 日本電信電話株式会社
E-mail: †{h-watanb,shinsuke,kusumoto}@ist.osaka-u.ac.jp

あらまし　自動でプログラムを生成する技術の実現手法として，遺伝的アルゴリズム（GA）を用いた生成と検証に基づく手法が提案されている．この手法では，テストケースを入力として受け取り，プログラムの改変と評価を繰り返して全テストケースに通過するプログラムの生成を試みる．GA におけるプログラム改変の方法は変異と交叉に大別される．このうち，交叉は過去の改変履歴の一部を再利用する操作であり，一点交叉や一様交叉などの生物を模した方法がある．しかし，これら既存の交叉はプログラム内の依存関係を破壊し，構文的意味的に誤ったプログラムを生成する問題がある．そこで本研究ではプログラムの依存関係を予め解析し，依存関係を破壊しない新たな交叉手法を提案する．評価実験として，プログラミングコンテストの問題を題材に提案手法の効果を確かめた．その結果，コンパイル成功個体数は従来手法が優れていたものの，提案手法は従来手法では生成できないプログラムの生成に成功した．

キーワード　自動プログラム生成，自動プログラム修正，交叉，プログラムスライス

1. はじめに

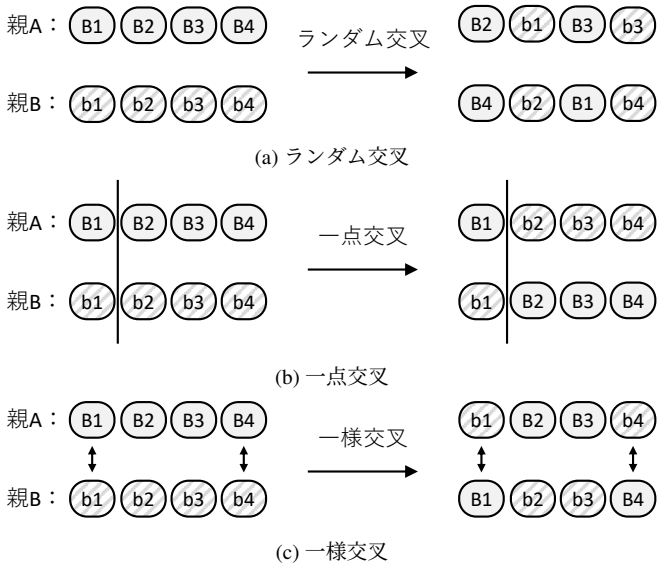
プログラムを開発者の介在なく完全自動で生成することを目指した研究の 1 つに，自動プログラム修正 [1]（Automated Program Repair，APR）と呼ばれる技術を転用した方法が存在する [2]．APR とは，バグを含むプログラムから自動的にバグを取り除く技術である．この APR の分野にブレイクスルーをもたらしたツールとして GenProg [3] がある．GenProg は入力として，バグを含むプログラムとテストケースを受け取り，遺伝的アルゴリズム [4]（Genetic Algorithm，GA）に基づき，入力されたバグを含むプログラムに対して改変（個体の生成）と優秀な個体の選別（個体の選択）を繰り返し探索的にソースコードを全テストケースを通過する，バグのない状態に近づけていく．APR を転用した自動プログラム生成（Automated Program Generation，APG）は，GA に基づく APR に対してバグを含むプログラムの代わりに最低限のコンパイルのみ可能な空プログラムを入力することで実現される．空プログラムは複数のテストが失敗する状態であり，これを複数のバグを含む状態とみなすことで，APR はテストケースを 1 つずつ通過するようにプログラムを進化させていく．

GA によるプログラム改変（個体生成）の方法は変異と交叉に大別される．変異によって巨大な探索空間からテスト通過に寄与する改変法（塩基）を選び出し，選び出された塩基間の組合せを交叉によって探索する．変異は 1 つの個体を親として，新たな塩基を 1 つ生成し親個体へ追加する操作である．この改変法には，プログラム文の挿入や削除，置換がある．交叉は過去に生成された 2 個体を両親として，両親の持つ塩基を組み替えることで複数の個体を生成する．具体的な交叉法としては，塩基列内のある場所を乱択によって決定し，その場所より後方の全塩基を入れ替える一点交叉や，塩基列の対応する場所ごとに乱択によって塩基を入れ替える一様交叉がある．

GenProg にはバグ修正に多くの時間を要するという課題がある [5]．この理由は，GenProg のアルゴリズムにおいて乱択を用いる箇所が多いことにある．この課題に着目したツールとしては，予め用意した修正パターンを適用する PAR [6] や Relifix [7]，多目的遺伝的アルゴリズムを利用する ARJA [8] や ARJAe [9] などが存在するが，いずれも個体選択や変異に着目したツールであり，交叉の改善には取り組んでいない．

本研究では APR を転用した APG における探索速度の向上を目的とした交叉の改善について考える．GenProg が用いる一点交叉や一様交叉の課題として，プログラムの制約条件を破壊してしまうことが挙げられれる．プログラムは構文的制約や意味的制約に代表される様々な制約条件を持つ．例えば，`i = a + 1;` という文をプログラムに挿入するためには，挿入箇所より前で `a` を定義する必要がある．しかし，既存の交叉ではこれらの制約条件を加味せず乱択によって新しい個体を生成するため，この条件を充足しない個体が生成されうる．

本稿では，プログラムが持つ制約条件を活用したプログラム



(a) ランダム交叉

(b) 一点交叉

(c) 一様交叉

図 1: 交叉の例

生成に特化した交叉を提案する．プログラムの制約を考慮した交叉によって，APR の巨大な探索空間から解が含まれない空間のみを削減し探索の高速化を目指す．評価実験ではプログラムコンテストの問題を題材として，提案する交叉と既存の交叉が生成した解候補のコンパイル成功率と両親に対して上位の解であるかを比較した．その結果，コンパイル成功個体数は従来手法である一点交叉が優れていたものの，提案手法は従来手法では生成できない両親の上位となる個体の生成に成功した．

2. 準　　備

2.1 遺伝的アルゴリズムを用いた自動プログラム生成

遺伝的アルゴリズム（GA）とは，生物の進化過程を模したメタヒューリスティクスアルゴリズムである．ループ（世代）を重ねるごとに個体を求める解へと少しずつ近づけていく．各世代においては，一定数の解候補（個体）に対して変形を加え，変形された個体がどの程度解へ近づいたかを評価し，次世代へ用いる個体を選択する．GA の主要部分となる変形は個体の持つ遺伝子に対して行われ，その方法には変異と交叉の 2 種類が存在する．

変　　異

生物の進化における突然変異を模した方法である．現存する個体から親となる 1 個体を選び出し，その個体に対して僅かな変形を加え新たな個体を生成する．

交　　叉

生物の進化における交配を模した方法である．現存する個体から両親となる 2 個体を選び出し，両親に過去に加えられた変形（塩基）を組み合わせて新たな個体を生成する．

交叉の具体的な手法として以下の 3 手法を挙げる．

ランダム交叉

両親の全ての遺伝子をランダムに並び替え，半分を取り出す手法（図 1a）

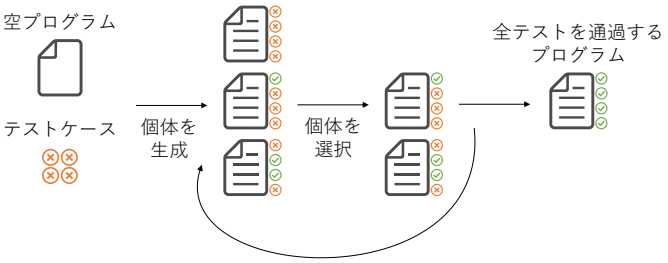


図 2: 遺伝的アルゴリズムを用いた自動プログラム生成の流れ

一 点 交 叉

両親の遺伝子上の場所を乱択によって 1 つ選び，その場所より後ろを入れ替える手法（図 1b）

一 様 交 叉

各遺伝子ごとに，独立に 1/2 の確率で入れ替えるかを決定する手法（図 1c）

次に，APG に対する GA の適用について説明する．GA はメタヒューリスティクスであり，その適用においては個体の遺伝子設計と選択に用いる個体評価の方法を定義する必要がある．GenProg では，前者をプログラムへの編集，後者を入力されたテストケースの通過数としている．図 2 に GenProg を用いた APG の流れをに示す．入力は最低限のコンパイルのみ可能な空プログラムとテストケースの集合であり，出力は入力された全テストケースに通過するプログラムである．変異や交叉によってプログラムに僅かな改変を加えることで新たな個体を生成し，個体のテスト通過数によって優秀な個体を選択する．図では，4 つのテストケースが入力され，3 つの個体が生成された状態を示している．生成された 3 個体の通過テスト数は上から 0，1，2 である．この通過テスト数に基づき，選択と淘汰される個体が決定される．図では，最上部に位置する通過テスト数 0 の個体が淘汰され，それ以外の 2 個体が選択された．GA ではこのような処理のループによって，最終的に全テストを通過するプログラムを生成する．

2.2 既存手法の課題

本節では，プログラミングコンテスト AtCoder[1]で過去に開催された AtCoder Beginner Contest（ABC）102 の A 問題[2]を題材として既存手法の課題を説明する．図 3 に APG の入力と生成個体の具体的な例を示す．図では 3 つのテストケースと最低限のコンパイルのみ可能なプログラムを APG に入力している．GA で生成される個体は図中の V1 から V4 の 4 つの個体が示すように，塩基の列からなる遺伝子を持つ．塩基や遺伝子の設計は様々なものが考えられるが，ここでは塩基はプログラムへの操作とその位置から構成される．図中の個体 V1 は B1 のみからなる遺伝子を持ち，その塩基 B1 は 2 行目を `return n` に置換するプログラムへの編集を意味している．個体 V1 はこの遺伝子の適用の結果，2 つ目のテストを通過するように進化している．

ここで，GA の処理中に個体 V1 と V2 が生成されたとする．

(注1)：https://atcoder.jp
(注2)：https://atcoder.jp/contests/abc102/tasks/abc102_a

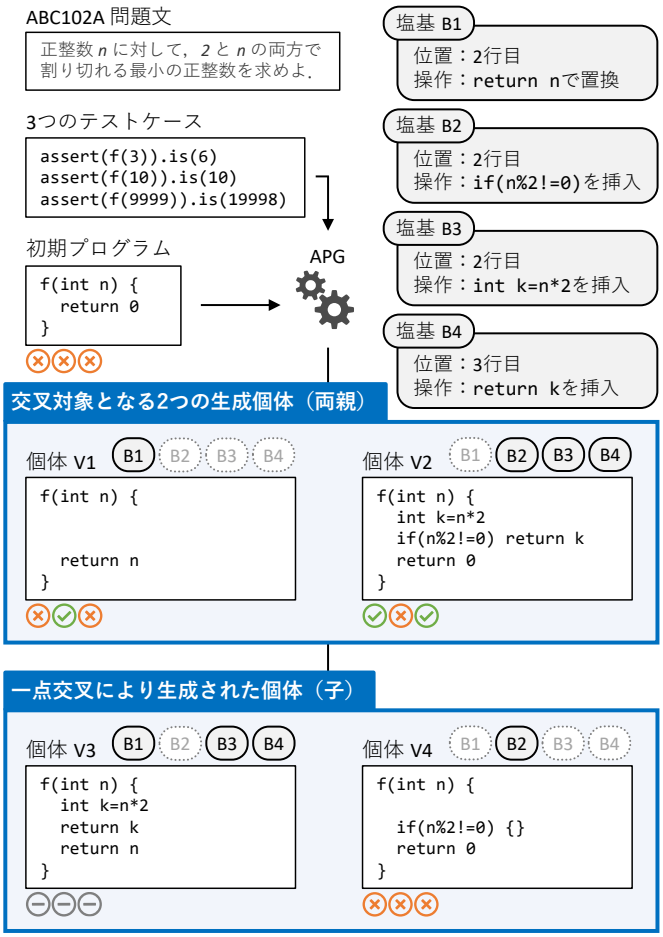


図 3: 生成個体の具体例と既存交叉の課題

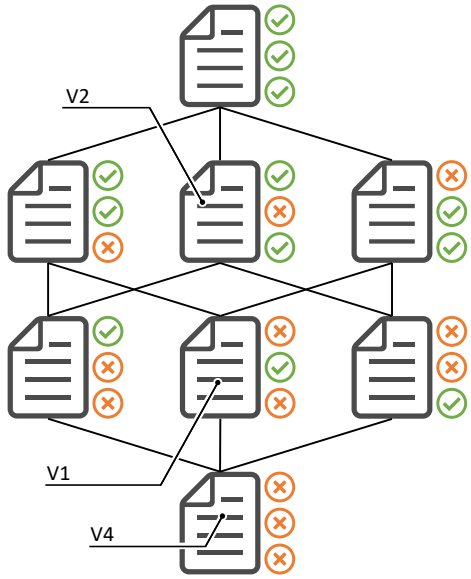


図 4: テスト数 3 のときの全個体の関係

個体 V1 は前述のとおり，塩基 B1 を持ち 2 つ目のテストに通過している．また，V2 は V1 の失敗する 1 つ目と 3 つ目のテストに通過している．V1 と V2 はテスト通過の観点から互いの失敗テストを補い合う相補的な個体と理解できる．テスト数 3 のときの全個体の関係を図 4 に示す．図より V1 と V2 は上下関係にないことが分かる．このような相補的な個体を優先的に交叉することで APG の効率が向上する [10]．よって，V1 と V2 を両親とする交叉による新たな個体（子）の生成を考える．図 3 の下方に一点交叉による子の例として V3 と V4 を示す．V3 と V4 は一点交叉によって両親となった V1，V2 が持つ 2 つ目以降の塩基が入れ替えられ生成された．V3 はコンパイル失敗，V4 はコンパイルには成功するものの全テストに失敗している．従来の交叉はプログラムの構文や意味を利用せず，乱択によってのみ個体を生成している．このような交叉は効率的とはいえない．プログラムの構文や意味，テストの実行時に得られる情報を利用して，コンパイル成功やテスト通過観点から両親の上位を生成することを目的とした新たな交叉を考える必要がある．

2.3 プログラムスライス

プログラムスライス [11] とは，プログラム内の各文の依存関係を明らかにする技術である．スライスには大きくスタティックスライス [12] とダイナミックスライス [13] が存在するが，本稿では前者に着目する．スタティックスライスはプログラム内のある文の実行に影響を与える可能性のある全ての文を抽出する技術である．スライス起点を通過テストの通過する `return` 文としてスライス技術を使えば，通過テストの実行に必要な文のみを抽出できる．

3. 提 案 手 法

本研究では，プログラムスライスを用いた遺伝子の依存を保持する交叉を提案する．プログラムスライスの利用によって，依存の保持だけでなく，コンパイルやテスト通過に不要な塩基の除外という副次的な効果も期待できる．提案手法は既存の交叉手法と同様に，両親となる 2 個体を入力として受け取り，それらから生成された複数の子を出力する．ただし，両親が相補的な個体であることを前提としている．両親が相補的な個体でない場合，子は生成されない．提案手法のアルゴリズムをアルゴリズム 1 に示す．提案手法は以下の 4 つのステップから構成される．

- Step-1：片方の親のみが通過するテストの抽出
- Step-2：プログラムスライス起点の決定
- Step-3：スライスによる依存塩基の抽出
- Step-4：抽出された塩基の結合

Step-1：片方の親のみが通過するテストの抽出

両親の通過テストを比較し，片方の親のみが通過するテストを抽出する．生成する子はこのステップで抽出されたテストに通過することが期待される．

Step-2：プログラムスライス起点の決定

親となる各個体について，Step-1 で抽出された各テストごとにプログラムスライスの起点となる文を決定する．スライス起点は通過テストの実行に影響のある最小限の文の集合が得られる箇所が望ましい．よって，通過テストの実行経路に含まれる，`return` 文もしくは `throw` 文をプログラムスライスの起点とする．`return` 文が存在しない場合は，通過テストの実行経路に含まれる最後に実行された文をスライス起点とする．このようにして，Step-1 で抽出したテストと 1 対 1 に対応するプログラムスライスの起点を決定する．

Step-3：利用する塩基の抽出

Step-2 で得られた各スライス起点からプログラムスライスを得る．プログラムスライスによって得られた文に対応する塩基を交叉に利用する塩基として抽出する．また，抽出した塩基は抽出前の順に並び替える．例えば，塩基の列 (a, b, c, d, e) から (a, b, d) を抽出する場合，その順番は (a, b, d) の 1 通りとなる．

Step-4：抽出された塩基の結合

各親個体から抽出された塩基を 1 つに結合し遺伝子を作成する．結合時に重複する塩基は取り除く．重複排除の目的は，変数定義などの重複によるコンパイルの失敗を防ぐことにある．結合した塩基から出力となる子を生成できる．

アルゴリズム **1** Linkage Crossover

Input:

Parent genes G_A and G_B

Passed tests $Passed_A$ and $Passed_B$

Output:

Child Genes G

function MAKE-GENE($G_A, G_B, Passed_A, Passed_B$)
  $PassedOnlyA \leftarrow Passed_A \setminus Passed_B$
  $PassedOnlyB \leftarrow Passed_B \setminus Passed_A$
  $SelectedA \leftarrow$ SELECT-GENES($G_A, PassedOnlyA$)
  $SelectedB \leftarrow$ SELECT-GENES($G_B, PassedOnlyB$)
  $G \leftarrow$ MARGE-GENES($SelectedA, SelectedB$)
  return G

function SELECT-GENES(G, T)
  $CoveredReturnStmt \leftarrow$ COVERAGE-ANALYSIS(G, T)
  $DependentGenes \leftarrow Slice(G, CoveredReturnStmt)$
  return $DependentGenes$

4. 実　　験

提案手法の評価のため，提案手法を GenProg の Java 実装である kGenProg [14] に実装し実験を行った．比較対象として，2.1 節で説明したランダム交叉，一点交叉，一様交叉の 3 手法を用いる．

4.1 実験題材の作成

提案手法や従来の交叉手法の動作のためには，入力となる 2 個体からなる両親が必要である．この実験題材となる両親の生成のために，プログラミングコンテスト AtCoder[3]で過去に開催された AtCoder Beginner Contest（ABC）のうち ABC101 から ABC180 までの 100 点問題 80 問を用いた．この 80 問に対して，kGenProg を実行し GA 中に生成された個体のうち，相補的な関係にある個体対を全て収集した．kGenProg の実行に必要な入力を表 1 に示す．実験題材の生成のために，kGenProg に必要なその他のパラメータは既定値[4]を用いた．

表 1: kGenProg への入力

項目	値
入力問題	ABC101~ABC180 100 点問題
問題数	80
乱数シード	0~1 （＝ 2 試行）
制限時間	1 試行あたり 1 時間
最大世代数	無制限
プログラムへの操作	挿入のみ
終了条件	正解個体の発見・時間切れ

上記 160 試行から 1,486 対の相補的な個体対が得られた．この 1,486 個体対を両親として，従来の交叉 3 種と提案手法を用いて子の生成を行う．

4.2 評 価 指 標

本実験では大きく 2 種類の評価指標を用いる．交叉によって生成された子自体の評価と親子の比較による評価である．前者には，コンパイル可能な子の個数と生成プログラムの行数を用いる．コンパイル可能な子を多く生成できるほど，よい交叉手法といえる．生成プログラムの行数を比較する目的は，提案手法による不要な塩基の除外効果を確認することにある．

次に，後者について説明する． まず，テスト通過の比較による 2 つの個体間での関係を定義する．

- 下位：相手の通過テストに 1 つ以上失敗し，相手の全失敗テストに失敗
- 同値：通過テストと失敗テストが等しい
- 上位：相手の全通過テストに通過し，相手の失敗テストを 1 つ以上通過
- 相補：相手の通過テストに 1 つ以上通過し，相手の失敗テストに 1 つ以上通過

この関係から，交叉により生成された子個体と親個体の取りうる関係は表 2 に示した ${}_4\mathrm{H}_2 = 10$ 通りへ分類できる．ただし，本実験では親個体を相補的な個体と限定しているため，表中の-で示した 4 種の関係となる子個体は生成されない．

親子の比較による評価では，各交叉手法により生成された子を表 2 の 6 種の関係へ分類し，その個体数を比較する．以下で，各関係の解釈を順に述べる．

上位&上位

両方の親に対して優れている個体を表す．提案手法の生成目標の個体である．

上位&相補

一方の親に対して上位であり，他方に対して相補的な個体を表す．個体の多様性向上の観点から重要な個体であるが，提案手法の生成目標ではない．

相補&相補

両方の親に対して相補的な個体を表す．上位&相補と同じく，重要な個体ではあるが，生成目標ではない．

相補&同値

一方の親に対して相補的であり，他方に対して同値な個体を表す．上位&相補や相補&相補と同じく重要な個体ではあるが，生成目標ではない．

相補&下位

一方の親に対して同値，他方に対して下位の個体を表す．交

(注3)：https://atcoder.jp

(注4)：https://github.com/kusumotolab/kGenProg/tree/v1.8.0#options

叉による上位個体の生成に失敗したことを表す．しかし，生成された場合でも GA に悪影響を与えないと考えられる．片方の親に対して下位であるため，GA 中の選択処理によって淘汰され次世代で利用されないからである．

下位&下位

両方の親に対して下位の個体を表す．相補&下位と同じく，交叉失敗と理解できる．

4.3 実験結果

4.1 節で得られた 1,486 個体対を両親として，4 種の交叉法それぞれを用いて 1 組の両親に対し 10 個体の子を試みた．つまり，1 つの交叉法あたりの生成個体数は 14,860 個体となる．

各交叉法と生成した子の内訳を表 3 に示す．表は個体数とコンパイル可能数に占める割合を示している．コンパイル可能な個体は一点交叉を用いた場合に最も多く得られた．その一方で，本研究の目的となる上位&上位や GA に良い影響を与える上位&相補，相補&相補は提案手法での生成が最も多かった．また，交叉失敗と理解できる相補&下位や下位&下位の個体も提案手法において最も多く生成された．

各交叉法ごとの個体の内訳を観察する．従来手法 3 種においては生成個体の傾向は似通っており，相補&同値が最も多くの割合を占めている．提案手法ではこの傾向は見られず，下位&下位が最も多かった．

コンパイル可能な個体におけるプログラム行数の平均を表 4 に示す．平均行数が最も小さいのはランダム交叉であった．しかし，両親の上位互換となる上位&上位や上位&相補に着目すると提案手法の平均行数が最も短かった．

5. 考　　察

5.1 提案手法の効果

提案手法は従来手法と比較して GA に良い影響を与えるといえる．提案手法では僅かではあるが，従来手法では生成できなかった上位&上位となる個体を生成している．また，GA 中の個体の多様性を向上させる上位&相補や相補&相補となる個体も最も多く生成したことも理由である．交叉失敗と理解できる相補&下位や下位&下位も提案手法での生成が最も多かったが，これらの個体は必ず淘汰されるため，GA に対して悪影響を与えることはない．

また，提案手法の副次的な作用である不要な塩基の排除も確認できた．表 4 より，提案手法により生成された個体のプログラム行数が短いことから，この効果が分かる．

表 2: 両親に対する子の関係

		親 B に対して			
		上位	相補	同値	下位
親 A に対して	上位	上位&上位	上位&相補	-	-
	相補	上位&相補	相補&相補	相補&同値	相補&下位
	同値	-	相補&同値	-	-
	下位	-	相補&下位	-	下位&下位

```
int error1(int n){
  return n++;
  int k = 0; // compile error
}
```

図 5: コンパイル失敗例：return 文の後に文が存在

```
int error2(int n){
  int k = 0;
  if(n % 2 != 0){
    int k = 3; // compile error
  }
  return k;
}
```

図 6: コンパイル失敗例：変数宣言の重複

5.2 提案手法のコンパイル失敗原因

提案手法のコンパイル失敗個体の原因について調査した．調査の結果，プログラムスライスから対応する塩基を抽出するのみではコンパイル可能な個体の生成には不十分であると分かった．これらを原因とするコンパイルエラーを防ぐためには，提案手法における塩基の結合時により厳密な解析を行う必要がある．具体的なコンパイル失敗の原因とその例を以下で説明する．

return 文の後に文が存在

Java では `return` 文の後に他の文が存在する場合，コンパイルエラーとなる．この具体例を図 5 に示す．このような個体が生成された原因は提案手法の遺伝子結合の処理にある．このコンパイルエラーを回避するには，遺伝子結合時に `return` 文をブロックの最下部に配置させるなどの工夫が必要である．

変数宣言の重複

コンパイルエラーの原因として，変数宣言の重複が存在した．この具体例を図 6 に示す．提案手法では，このコンパイルエラーを回避するために，重複塩基の排除を行っていた．しかし，適用の位置のみが異なる塩基は排除していなかったため，このコンパイルエラーが発生した．

6. 妥当性の脅威

提案手法の実装には Java で実装された APR ツール kGenProg を用いた．Java 以外のプログラム言語，特にスクリプト言語を用いた場合，得られる実験結果が結果が異なる可能性がある．また，GenProg など他のツールを用いた実験も妥当性確保に必要である．

実験では交叉対象となる両親を AtCoder の問題 80 問から生成した．この問題以外から交叉の両親を作成した場合，同様の結果が得られるとは限らない．

7. おわりに

本稿では，プログラムスライスを利用した両親の上位となる個体を生成する交叉を提案した．また，評価実験としてプログラミングコンテスト AtCoder の問題 80 問から相補的な個体を

表 3: コンパイル可能な個体数と両親との関係による分類

個体数	コンパイル可能	上位&上位		上位&相補		相補&相補		相補&同値		相補&下位		下位&下位	
ランダム交叉	173	0	(0.0%)	0	(0.0%)	7	(4.0%)	118	(68%)	0	(0.0%)	48	(28%)
一点交叉	12,168	0	(0.0%)	6	(0.049%)	96	(0.79%)	11,850	(97%)	108	(0.89%)	108	(0.89%)
一様交叉	5,333	0	(0.0%)	23	(0.43%)	725	(14%)	3,845	(72%)	366	(6.9%)	374	(7.0%)
提案手法	6,852	4	(0.058%)	54	(0.79%)	1,019	(15%)	1,833	(27%)	1,322	(19%)	2,620	(38%)

表 4: コンパイル可能な個体におけるプログラム行数の平均

個体数	全体の平均	上位&上位	上位&相補	相補&相補	相補&同値	相補&下位	下位&下位
ランダム交叉	26.0	-	-	52.9	17.4	-	43.3
一点交叉	168	-	50.3	55.2	170	177	89.0
一様交叉	132	-	113.4	139	122	188	172
提案手法	36.5	21	28.4	38.3	19.1	37.9	47.6

生成し，提案手法の性質を確認した．コンパイル成功個体数は従来手法である一点交叉が優れていたものの，提案手法は従来手法では生成できない両親の上位となる個体の生成に成功した．また，提案手法によって生成されるプログラムの行数を削減できた．

今後の研究課題として，提案手法が APG の効率へどのような効果をもたらすのか確認することが挙げられる．生成できるプログラムの増減や生成時間を従来手法と比較することで，効果を確認する予定である．また，提案手法の副次的な作用である不要塩基の排除がテストケースへの過剰適合を招く可能性が考えられる．生成されたプログラムがテストケースへの過剰適合 [15] を起こしていないか評価する必要がある．

手法の改善という観点では，5.2 節で述べた提案手法でのコンパイル失敗原因への対応も重要な課題である．また，本稿ではプログラムスライス技術のうち，スタティックスライスを用いたが，ダイナミックスライスの利用も検討している．

謝辞 本研究は JSPS 科研費（JP20H04166，JP21K18302，JP21K11820, JP21K11829, JP21H04877, JP22H03567, JP22K11985）の助成を得て行われた．

文　献

[1] L. Gazzola, D. Micucci, and L. Mariani, “Automatic software repair: A survey,” IEEE Transactions on Software Engineering, vol.45, no.1, pp.34–67, 2017.

[2] 富田裕也，松本淳之介，柗本真佑，肥後芳樹，楠本真二，倉林利行，切貫弘之，丹野治門，“遺伝的アルゴリズムを用いた自動プログラム修正手法を応用したプログラミングコンテストの回答の自動生成に向けて，” 情報処理学会研究報告，第 2020-SE-204 巻，pp.1–8，March 2020.

[3] C.L. Goues, T. Nguyen, S. Forrest, and W. Weimer, “Genprog: A generic method for automatic software repair,” IEEE Transactions on Software Engineering, vol.38, no.1, pp.54–72, 2012.

[4] D.E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, 1st edition, Addison-Wesley Longman Publishing, 1989.

[5] F. Long and M. Rinard, “An analysis of the search spaces for generate and validate patch generation systems,” Proc. International Conference on Software Engineering, pp.702–713, 2016.

[6] D. Kim, J. Nam, J. Song, and S. Kim, “Automatic patch generation learned from human-written patches,” Proc. International Conference on Software Engineering, pp.802–811, 2013.

[7] S.H. Tan and A. Roychoudhury, “relifix: Automated repair of software regressions,” Proc. International Conference on Software Engineering, pp.471–482, 2015.

[8] Y. Yuan and W. Banzhaf, “ARJA: automated repair of java programs via multi-objective genetic programming,” IEEE Transactions on Software Engineering, vol.46, no.10, pp.1040–1067, 2020.

[9] Y. Yuan and W. Banzhaf, “Toward better evolutionary program repair: An integrated approach,” ACM Transactions on Software Engineering and Methodology, vol.29, no.1, pp.1–53, 2020.

[10] H. Watanabe, S. Matsumoto, Y. Higo, S. Kusumoto, T. Kurabayashi, K. Hiroyuki, and H. Tanno, “Applying multi-objective genetic algorithm for efficient selection on program generation,” Proc. Asia-Pacific Software Engineering Conference, pp.515–519, 2021.

[11] M. Weiser, “Program slicing,” IEEE Transactions on Software Engineering, vol.SE-10, no.4, pp.352–357, 1984.

[12] M. Weiser, “Programmers use slices when debugging,” Communications of the ACM, vol.25, no.7, pp.446–452, 1982.

[13] B. Korel and J. Laski, “Dynamic slicing of computer programs,” Journal of Systems and Software, vol.13, no.3, pp.187–195, 1990.

[14] 柗本真佑，肥後芳樹，有馬諒，谷門照斗，内藤圭吾，松尾裕幸，松本淳之介，富田裕也，華山魁生，楠本真二，“高処理効率性と高可搬性を備えた自動プログラム修正システムの開発と評価，” 情報処理学会論文誌，vol.61，no.4，pp.830–841，2020.

[15] E.K. Smith, E.T. Barr, C.L. Goues, and Y. Brun, “Is the cure worse than the disease? overfitting in automated program repair,” Proc. Joint Meeting on Foundations of Software Engineering, pp.532–543, 2015.