修士学位論文

題目

プログラムスライスを用いた交叉による自動プログラム生成の効率改善

指導教員

楠本 真二 教授

報告者

渡辺 大登

令和 5 年 2 月 1 日

大阪大学 大学院情報科学研究科

コンピュータサイエンス専攻

令和 4 年度 修士学位論文

プログラムスライスを用いた交叉による自動プログラム生成の効率改善

渡辺 大登

内容梗概

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 80 問を対象とした適用実験を行った結果，既存の交叉と比べた生成速度の向上を確認した．

主な用語

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

目次

図目次

表目次

1　はじめに

開発者が介在しない完全自動でのプログラム生成（Automated Program Generation，APG）を目指した研究の一つとして，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 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 にブレイクスルーをもたらした手法として，遺伝的アルゴリズム [3]（Genetic Algorithm，GA）に基づく探索的な手法がある．この手法ではバグを含むプログラムとテストケースを受け取り，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バグのない状態に近づけていく．APR を転用した 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 [4] と交叉 [5] 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て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 [6] 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 [7] がある．

GA ベースの APR は一種の探索問題であり，解空間の効率的な探索が重要な課題である [8]．探索の効率化はバグ修正に要する処理時間の削減のみならず，修正可能なバグ種別の拡大に繋がる．APR の探索効率に着目した研究としては，予め用意した修正パターンを適用する PAR [9] や Relifix [10]，多目的遺伝的アルゴリズム [11] を利用する ARJA [12] や ARJAe [13] 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，GA の性能に大きな影響を持つ [14] 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．既存 APR ツール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の破壊が挙げられる．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，挿入箇所より前で変数 `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生成に特化した交叉を提案する．本研究の主な貢献を以下に記す．

- APR を転用した APG における既存交叉の比較：Le Goues らは実際のプロジェクトを題材として APR ツールである GenProg [15] のバグ修正能力を調査し，一様交叉が一点交叉に比べ優れていると述べている [16]．一方で，APR ツールを APG に転用したプログラム生成に対する既存交叉の比較は行われていない．本稿では，プログラミングコンテストで実際に出題された問題 80 問を対象に既存交叉の性能比較を行った．その結果，バグ修正の場合と同じく一様交叉が優れていた．
- プログラムの意味的制約を考慮した交叉の提案：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，生成可能な問題や生成所要時間を既存の交叉と比較した．その結果，提案交叉は既存交叉の中で最も優れた一様交叉と補完関係にあることが分かった．

以降，2 節では GA を用いた APG と既存手法の課題について例を交えて説明し，3 節では提案手法について述べる．4 節では設定した Research Question を説明し，5 節で行った実験と実験結果を示し，Research Question の結論を述べる．6 節では関連研究を示し，7 節では本研究における妥当性の脅威について述べ，最後に 8 節で本研究のまとめと今後の課題について述べる．

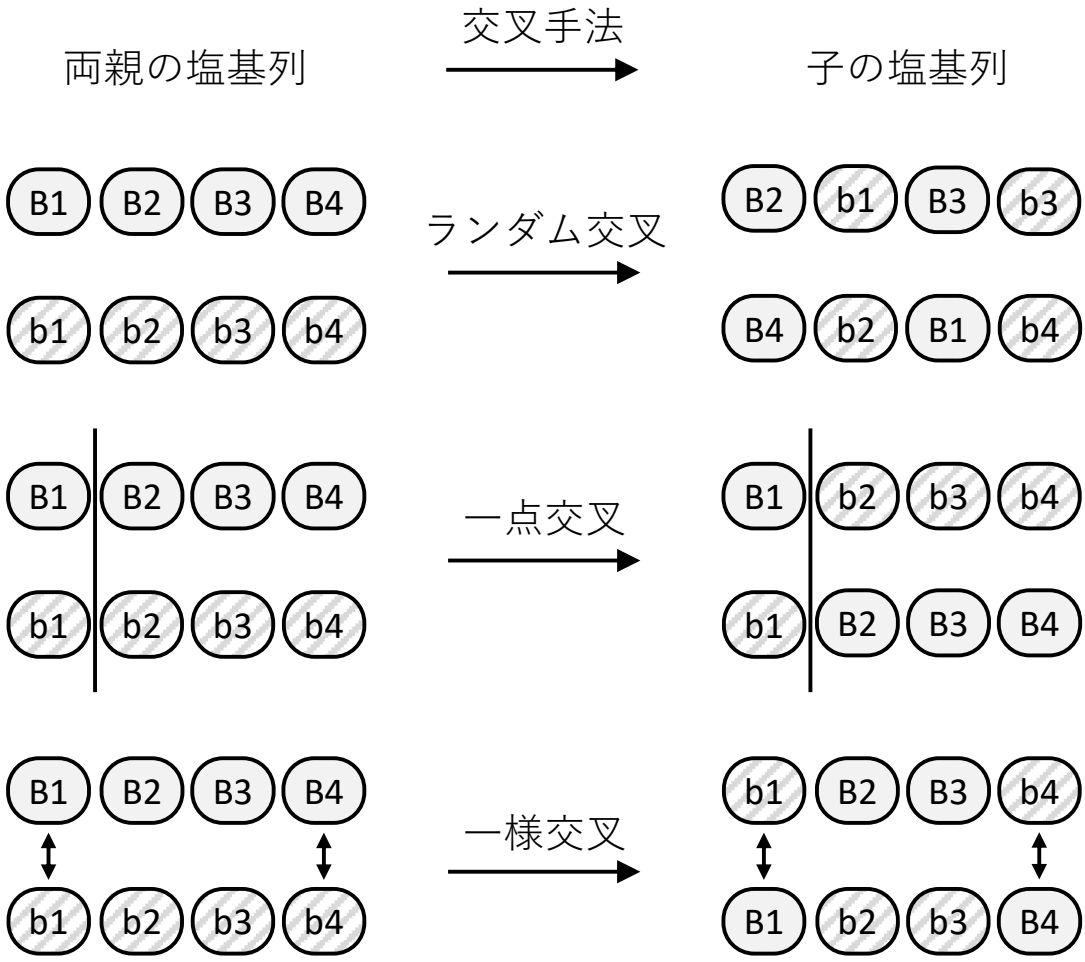

図 1　既存交叉法の例

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 [17] 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．

変異は生物の進化における突然変異を模した個体生成の方法である．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（塩基）を加え新たな個体を生成する．一方で，交叉は生物の進化における交配を模した個体生成の方法である．現存する個体から両親となる 2 個体を選び出

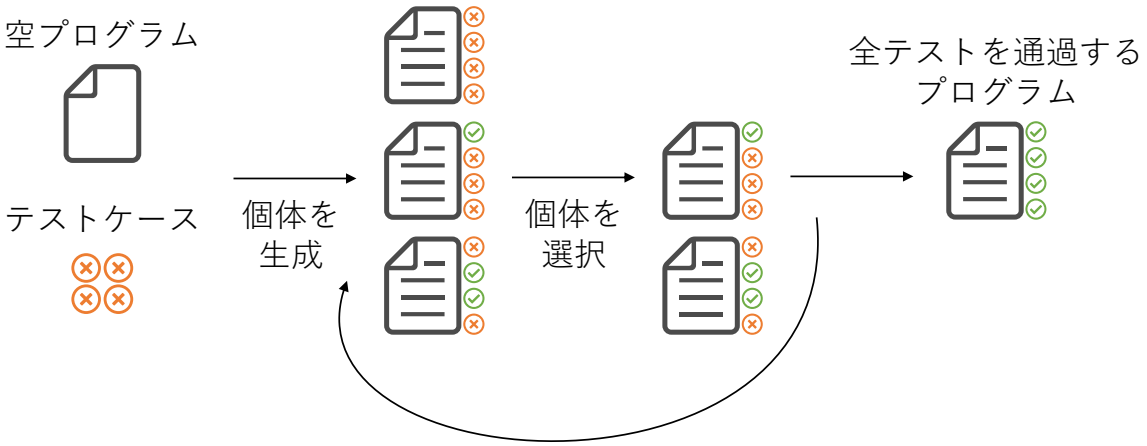

図 2　遺伝的アルゴリズムを用いた自動プログラム生成の流れ

し，両親に過去に加えられた変形を組み合わせて新たな個体を生成する．既存交叉の具体的な手法として図 1 に 3 つの手法を示す．

- ランダム交叉：両親の全ての塩基をランダムに並び替え，半分を取り出す手法
- 一点交叉：両親の遺伝子上の位置を乱択によって 1 箇所選び，その場所より後ろの塩基を全て入れ替える手法
- 一様交叉：各遺伝子上の位置ごとに，独立に 1/2 の確率で塩基を入れ替えるかを決定する手法

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

既存の交叉法の課題として，GA で広く用いられている一般的的な方法のみが利用されており，プログラム生成には特化されていない点が挙げられる．GA は文字通り生物の進化を模倣した手法であり，

遺伝子の長さを常に一定に保つという暗黙の制約がある．2.1 節で挙げた 3 つの交叉手法によって生成される子の遺伝子は常に親個体と等しい．しかし，APG ではこの制約は不要であり，性能の悪化に繋がる場合もある．

この既存交叉の課題について具体的な例を挙げて説明する．図 3 に APG の入力と生成個体の例を示す．図では，プログラミングコンテスト AtCoder [18] で過去に開催された AtCoder Beginner Contest（ABC）102 A 問題 [19] を題材とした．APG 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入力として受け取っている．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．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 [20]．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 [21] 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 [22] とダイナミックスライス [23] 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．

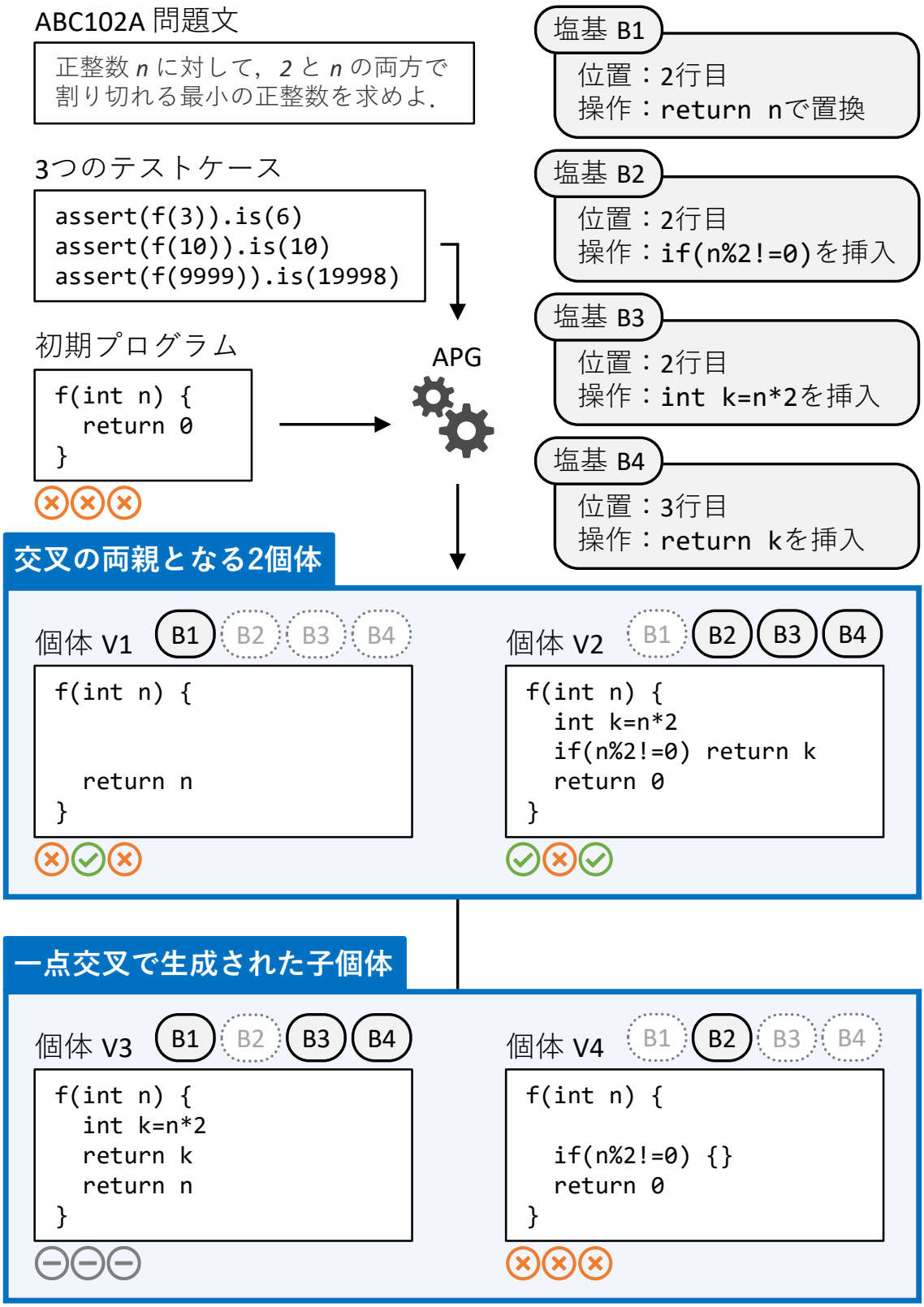

図 3　生成個体の具体例と既存交叉の課題

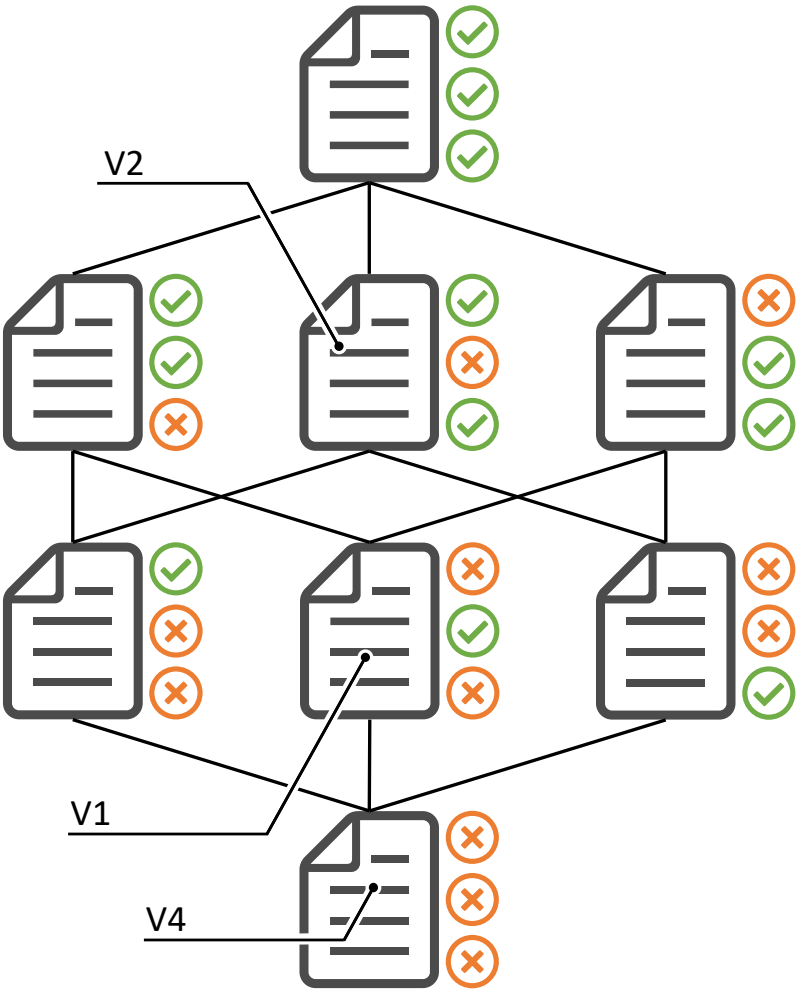

図 4　テスト数 3 のときの全個体の関係

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．例えば，図 3 中の個体 V1 と V2 が交叉の両親の場合，V1 は 2 つ目のテスト，V2 は 1 つ目，3 つ目のテストが抽出される．提案交叉の目標は生成する子がこのステップで抽出された全テストに通過することである．

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

Step-3 で抽出した塩基を 1 つに結合して遺伝子を作成する．結合時に重複する塩基は取り除く．また，変数宣言のブロック最上部への移動や，`return` 文のブロック最下部への移動も行う．これらの目的は，変数名の重複や未定義などを原因とするコンパイルの失敗を防ぐことにある．結合した塩基列から出力となる子を生成できる．

4 Research Question

提案手法の有効性を評価するために，以下の **Research Questions**（**RQ**）を設定する．

RQ1：提案交叉による生成個体にはどのような傾向があるか？

RQ1 では，提案交叉が生成する個体にどのような傾向が見られるのかを既存の交叉法と比較して確認する．各交叉法に対して同じ個体を両親として与え，生成された子個体の性質を比較する．評価の観点としては，生成された子自体の比較と，入力となる両親と生成された子の比較の 2 種類を用いる．本 RQ によって，提案交叉の目的である両親の上位となる個体を生成できているか確認できる．

RQ2：提案交叉は **APG** の効率へどのような影響を与えるか？

RQ2 では，提案交叉が APG の効率へどのような影響を与えるかを調べる．APG ツールへ提案交叉と既存交叉を組み込み，プログラム生成を試みる．評価指標としては，生成に成功した問題数と生成の所要時間を用いる．本 RQ によって，提案交叉が APG の効率を向上しているか確認できる．

5 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 [24] に実装し，2 種類の実験を行った．提案手法で必要となるプログラムスライスの作成にはプログラム解析ツールの TinyPDG [25] を用いた．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．

5.1 実験 1 と実験 2 に共通する実験設定

本節では実験 1，2 に共通する実験設定について説明する．実験設定の一覧を表 1 に示す．kGenProg へ入力する生成目標となる題材としては，AtCoder で過去に開催された AtCoder Beginner Contest（ABC）から ABC101 から ABC180 までの 100 点問題 80 問を用いた．kGenProg への入力となるテストケースは，AtCoder が公開している全テストケースを利用した．このテストケースは入力に対して出力が正しいかを確認するテストのみで構成され，計算量の適切さは考慮されない．また，問題あたりの平均テストケース数は 10.9 個であった．

kGenProg は GA に基づきプログラムを生成するため，生成中の各段階において乱択的要素が含まれる．よって，実験結果の適切な評価のためには，乱数シードを変化させながら複数回ツールを実行する必要がある．具体的な実行回数や乱数シードは各実験により異なる．

kGenProg は再利用に基づく APR ツールであるため，個体の生成時にはプログラム改変の材料とするソースコード片が必要となる．この再利用コードとして，実験題材とした 80 問の正解コード中の全ステートメントを利用した．通常では，APG ツールの利用時に正解コードが再利用コード中に含まれ

表 1　実験設定

項目	値
入力問題	ABC101∼ABC180 100 点問題
問題数	80
乱数シード	各実験により異なる
制限時間	1 試行あたり 1 時間
再利用コード	全問題の正解コード片
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
実験環境	Xeon E5-2630 2CPUs 16GB mem

るとは限らない．しかし，本実験では独立した小さなコードスニペットの再利用のみでプログラムの生成が可能かを確かめるため，全ての正解コードを用いた．この実験設定は実際の APG ツールの利用前提から乖離しているが，全実験において同一の再利用コードを用いたため，実験の平等性は確保されている．その他の GA の動作に必要なパラメータは既定値[*1]を用いた．

5.2 実験 1

実験 1 では，事前に生成した両親を各交叉へ入力して，生成される子の性質について調べる．評価の方法としては，生成された子自体の評価と，入力となる両親と生成された子を比較する評価の 2 種類を用いる．

5.2.1 実験題材の作成

提案手法や従来の交叉手法の動作には，入力する 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を生成するために，5.1 節で説明した実験設定で 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．ここで乱数シードは 0 と 1 の 2 種類とし，1 問題あたり 2 回プログラム生成を試みた．その結果，合計 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を生成する．

5.2.2 評価指標

実験 1 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，優れた交叉手法といえる．生成プログラムの行数を比較する目的は，提案手法の不要な塩基を除外する効果を確認することにある．

次に，後者の評価指標について説明する．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

[*1] https://github.com/kusumotolab/kGenProg/tree/v1.8.0#options

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．GA に重要な個体の多様性向上 [26] 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．

5.2.3 実験結果

5.2.1 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．

表 3 に各交叉法によって生成されたコンパイル可能な個体数を，表 4 に生成した子の内訳とコンパイル可能個体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

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

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 5 に示す．全個体の平均行数が最も小さいのはランダム交叉であった．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．

表 3　コンパイル可能な個体数

交叉手法	コンパイル可能個体数
ランダム交叉	173　(1.2%)
一点交叉	12,168　(82%)
一様交叉	5,333　(36%)
提案手法	6,852　(46%)

表 4　生成された子の両親との関係による分類

交叉手法	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	0	(0.0%)	0	(0.0%)	7	(4.0%)	118	(68%)	0	(0.0%)	48	(28%)
一点交叉	0	(0.0%)	6	(0.049%)	96	(0.79%)	11,850	(97%)	108	(0.89%)	108	(0.89%)
一様交叉	0	(0.0%)	23	(0.43%)	725	(14%)	3,845	(72%)	366	(6.9%)	374	(7.0%)
提案手法	4	(0.058%)	54	(0.79%)	1,019	(15%)	1,833	(27%)	1,322	(19%)	2,620	(38%)

表 5　コンパイル可能な個体の平均プログラム行数

交叉手法	全個体	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

5.3 RQ1：提案交叉による生成個体にはどのような傾向があるか？

RQ1 では交叉による生成個体の性質を比較する．表 4 より提案手法は僅かではあるが，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ている．これらから，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．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 5 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．

RQ1 の結論：提案交叉は既存交叉と比較して，GA によい影響を与える個体を生成している．また，生成個体の行数も既存手法と比較して短かった．

5.4 実験 2

実験 2 では交叉に関して 5 種類の設定でプログラム生成を試み，その結果を比較する．既存の交叉として一点交叉，一様交叉，ランダム交叉と提案交叉を APG ツールに組み込み，プログラム生成を行う．また，上記の交叉手法に加えて，交叉を行わない場合の実験も行う．ここで乱数シードは 0 ～ 9 の 10 種類とし，1 問題あたり 10 回プログラム生成を試みた．実験 2 の目的は，交叉手法が APG の効率へ及ぼす影響を調査することにある．評価指標として，入力した全テストケースを通過する個体の生成数（成功数）とプログラム生成の所要時間（生成時間）を用いる．

5.4.1 実験結果

まず，生成に成功した問題について確認する．図 5 に各手法における生成に成功した問題のベン図を示す．ここで，生成に成功した問題とは全 10 試行のうち 1 回でも入力した全テストケースに通過するプログラムを生成できた問題とする．図中の円は生成に成功した問題の集合を表している．いずれかの手法で成功した問題は 31 問であり，そのうち 5 手法全てで成功した問題は 17 問であった．交叉なし，ランダム交叉，一点交叉の 3 種においてのみ成功する問題は存在しなかった．一様交叉でのみ生成可能な問題は 2 問，提案交叉でのみ可能な問題は 1 問であった．

表 6 に生成に生成した試行数を示す．表の左側の数値は生成成功試行数の合計である．一点交叉の成功試行数が 52 試行と最も少なく，その他の交叉法では大きな違いは見られなかった．

次に，生成時間について確認する．生成成功時に要した時間の箱ひげ図を図 6 に示す．本図では，ひげの長さの上限は四分位範囲の 1.5 倍であり，その上端より大きい値を外れ値とした．横軸は生成所要

時間を表し，短いほどプログラム生成の効率がよいといえる．平均値を比較すると，生成時間が短い順に提案交叉（17.41 分），一様交叉（17.44 分），一点交叉（18.90 分），交叉なし（20.10 分），ランダム交叉（21.24 分）となる．しかし，第三四分位数の比較では，一様交叉が最も短時間でプログラムを生成している．

一様交叉と提案交叉において，さらに詳細な比較を行う．この理由は，図 5 において一様交叉や提案交叉のみ生成に成功した問題が存在したからである．図 7 に一様交叉と提案交叉の両方で成功した 23 問の平均生成時間を示す．図の縦軸は生成所要時間の平均であり，横軸は問題名である．一様交叉が提案交叉より生成時間が短い問題名を橙色で，提案交叉が短い問題を青色でそれぞれ表している．一様交叉での生成時間が短い問題数は 6 問，提案交叉が短い問題は 17 問であった．また，生成時間に 2 倍以上の差が見られた問題に着目すると，一様交叉が短い問題は 1 問のみ，提案交叉が短い問題は 4 問であった．

表 6　生成に成功した試行数

交叉手法	生成成功試行数
交叉なし	57
ランダム交叉	61
一点交叉	52
一様交叉	62
提案手法	60

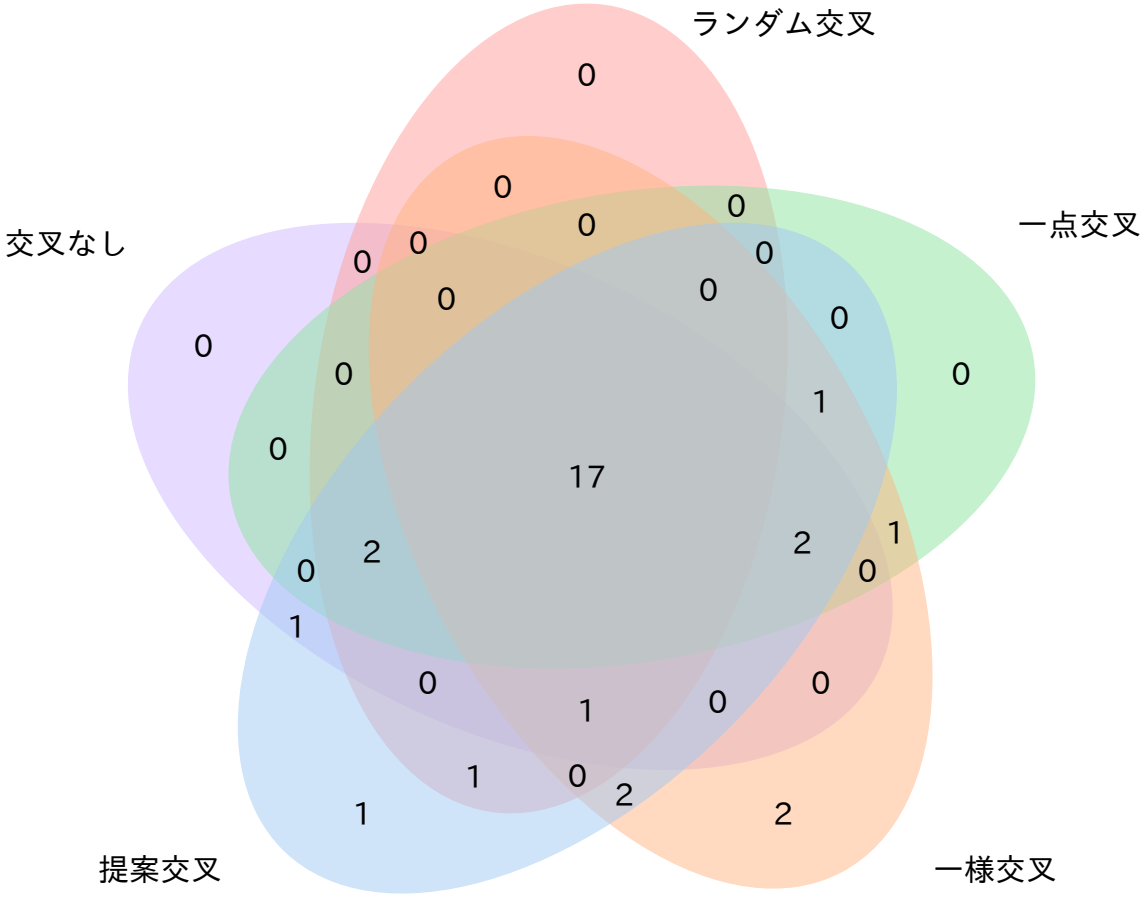

図５　生成に成功した問題のベン図

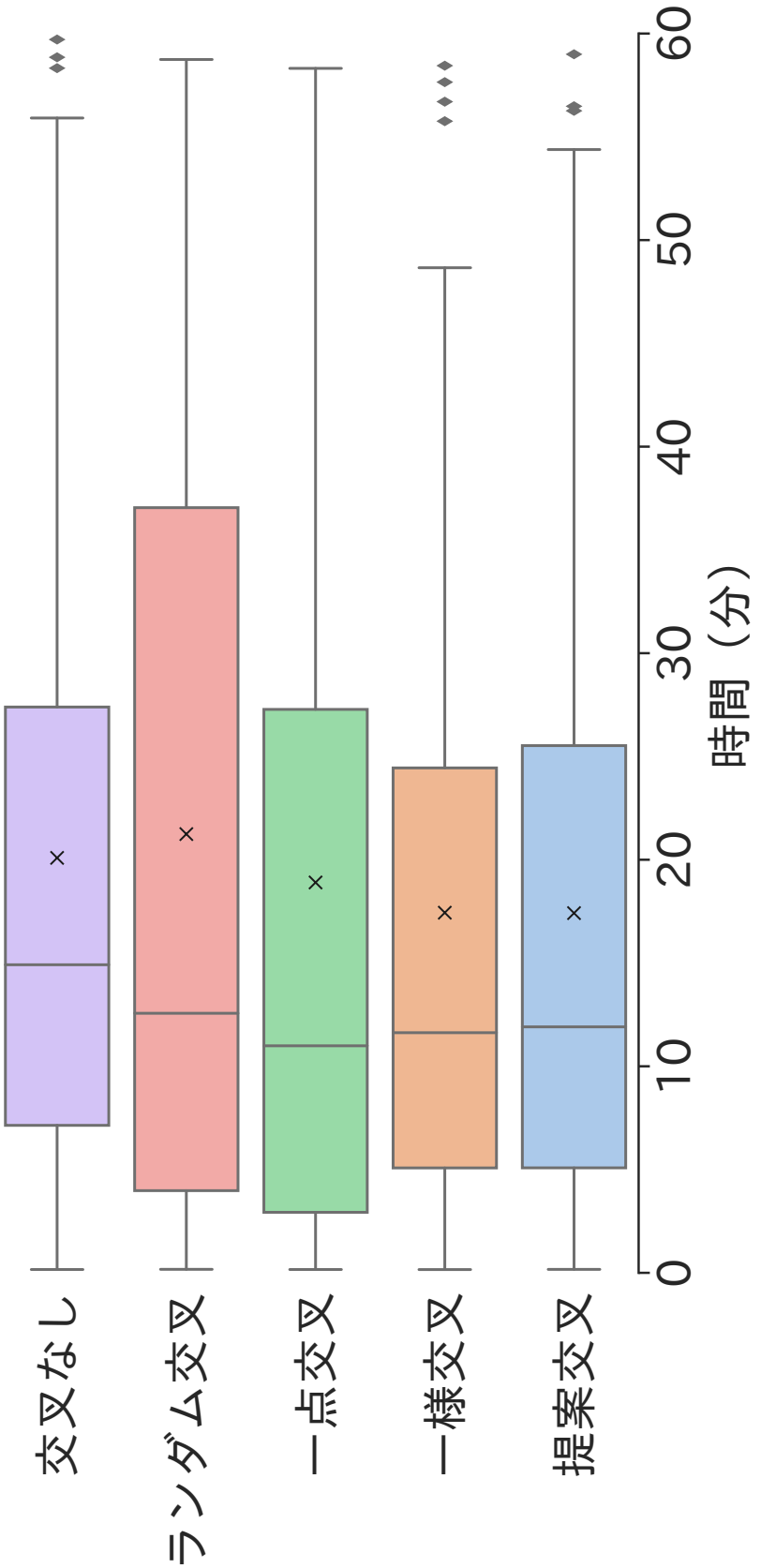

図 6　生成時間の比較

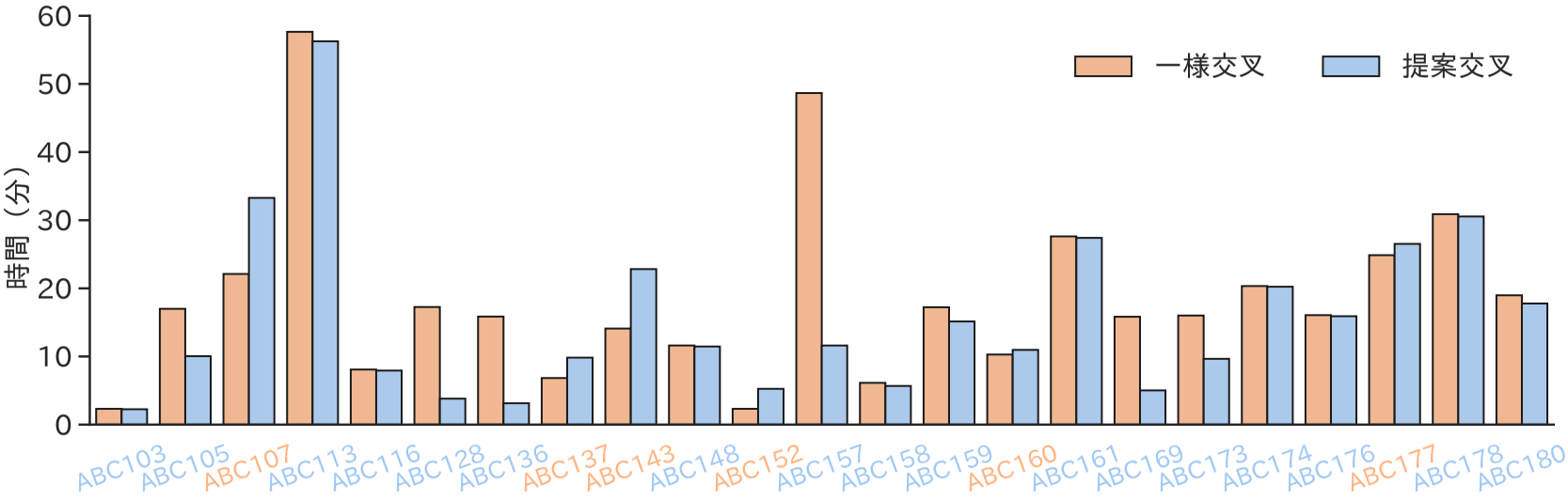

図 7　各問題に対する生成時間の比較

5.5 RQ2：提案交叉は APG の効率へどのような影響を与えるか？

RQ2 では提案交叉が APG に与える影響を確認する．図 5 より，提案交叉で生成可能な問題数は 28 問であり，最も多くの問題を生成可能であった．その一方で，一様交叉のみ生成可能な問題が 2 問存在した．生成時間の比較では，提案交叉は図 6 より生成時間が最も短かった．提案交叉で生成に失敗した問題 2 問を生成可能な一様交叉との時間比較では，提案交叉は共通で成功した 23 問中 17 問において一様交叉より生成時間が短かった．

RQ2 の結論：提案交叉を用いた場合に APG の効率が最も高かった．その一方，一様交叉でのみ生成可能な問題が 2 問存在した．

6 関連研究

6.1 APR 以外の分野における交叉の改善

GA はメタヒューリスティクスであり，その適用において問題固有の知識や制約の利用は必須でない．しかし，それらの知識や制約を活用することで，GA の効率は向上する [27]．このような研究の例として，巡回セールスマン問題 [28] に GA を適用する研究における交叉の改善 [29] や，ニューラルネットワークの遺伝的進化における隠れ層の構造を考慮した交叉 [30] がある．本稿ではこれら既存研究と同じく，問題固有の知識や制約を利用した問題特化の交叉を提案した．

6.2 APR 分野における交叉の改善

APR 分野においても，交叉を改善する研究は多数行われている [31]．Oliveira らは遺伝子の組み換え単位を細粒化した交叉により修正率を 34% 向上させた [32]．また，Le Goues らは個体の遺伝子表現の改善により間接的に既存の交叉を改善した [33]．これら既存手法の主軸は探索空間の拡大，すなわち生成する子個体の種類の増加にある．その一方で，本研究の提案はプログラムの依存解析により両親の上位となる子を生成する交叉であり，探索可能な空間は縮小している．

また先行研究として，著者らの研究グループでは APR を転用した APG の改善手法を提案している [34] が，本研究とは着眼点と手法の両方が異なる．先行研究では，交叉の対象個体を適切に選択するために，GA の評価関数を改善したが，本研究では交叉で生成される子個体が両親の上位互換となるように，プログラムスライスを用いた交叉を提案した．これら 2 手法を組み合わせることで，より効率的なプログラム生成が期待できる．

7　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異なる可能性がある．また，Astor [35] など他の APR ツールを用いた実験も妥当性確保に必要である．

実験題材として AtCoder で実際に出題された問題 80 問から生成した．この題材以外を用いた場合に同様の結果が得られるとは限らない．また，再利用に基づく APR ツールの動作に必須であるプログラムの生成材料として全正解コードを用いたことも妥当性の脅威といえる．通常の APG 利用時を想定した，正解コードを利用しない実験を行った場合，異なる結果が得られる可能性がある．

8　おわりに

本研究では APG の効率向上を目的として，プログラムスライスを利用する交叉を提案した．提案交叉はスタティックスライスを用いたプログラムの文間の依存解析により，個体のコンパイルに必要な文を抽出する．加えて，2 種類の評価実験から提案手法の有効性を確認した．

今後の研究課題として，ダイナミックスライスを用いた手法の改善が挙げられる．本稿ではプログラムスライス技術のうち，プログラムの実行によって得られる情報を利用しないスタティックスライスのみを用いた．その一方で，スライス技術には実行時に得られる情報を用いるダイナミックスライスが存在する．APR では個体の評価時にプログラムを実行するため，オーバーヘッドなく実行時情報を取得できる．この実行時情報の活用により，さらなる手法の改善が期待できる．

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 [36] 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を起こしていないか評価することも重要な課題である．

謝辞

本研究の遂行にあたり，多くの方々にご指導とご支援を賜りました．

終始丁寧かつ暖かなご指導を賜りました，楠本真二教授に心より感謝申し上げます．研究の要所で示唆に富むご助言をいただき，また，研究指導の他にも様々なご支援を頂き大変お世話になりました．

本研究の全過程を通して終始熱心かつ丁寧なご指導を頂きました，柗本真佑助教に心より感謝申し上げます．研究内容に関する議論や論文執筆，研究発表へのご指導など多大なご尽力をいただきました．これらのご指導は研究能力の向上だけでなく，私の人間的な成長にも繋がりました．

ソフトウェア工学講座の肥後芳樹教授には，本研究に関する有益かつ的確なご助言を頂きました．研究の方向性に関するご意見を多数頂き，研究の全体像を客観的に捉え直す助けとなりました．心より感謝申し上げます．

事務補佐員の橋本美砂子氏，前事務補佐員の神谷智子氏には，研究活動を円滑に行うための様々なご支援をいただきました．深く感謝申し上げます．

倉林利行氏をはじめとする日本電信電話株式会社ソフトウェアイノベーションセンタの皆様には，研究内容に対して客観的で多様なご意見をいただきました．また，同じ研究目標に取り組む皆様との意見交換は研究へのモチベーションの向上へ繋がりました．誠にありがとうございました．

楠本研究室の皆様には，常に刺激的かつ活発な議論をいただき，精神的にも支えられました．誠にありがとうございました．

最後に，本研究の遂行中，常に励まし応援頂いた友人や家族へ心より感謝の意を表します．

参考文献

[1] Gazzola, L., Micucci, D. and Mariani, L.: 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, Vol. 2020-SE-204, No. 7, pp. 1–8 (2020).

[3] Goldberg, D. E.: *Genetic Algorithms in Search, Optimization and Machine Learning*, 1st edition (1989).

[4] Schmitt, L. M.: Theory of genetic algorithms, *Theoretical Computer Science*, Vol. 259, No. 1, pp. 1–61 (2001).

[5] Umbarkar, A. J. and Sheth, P. D.: Crossover Operators in Genetic Algorithms: a Review, *ICTACT Journal on Soft Computing*, Vol. 6, No. 1, pp. 1083–1092 (2015).

[6] Poli, R. and Langdon, W. B.: Genetic Programming with One-Point Crossover, in *Proceedings of the Soft Computing in Engineering Design and Manufacturing*, pp. 180–189 (1998).

[7] Syswerda, G.: Uniform Crossover in Genetic Algorithms, in *Proceedings of the International Conference on Genetic Algorithms*, pp. 2–9 (1989).

[8] Long, F. and Rinard, M.: An Analysis of the Search Spaces for Generate and Validate Patch Generation Systems, in *Proceedings of the International Conference on Software Engineering*, pp. 702–713 (2016).

[9] Kim, D., Nam, J., Song, J. and Kim, S.: Automatic Patch Generation Learned from Human-Written Patches, in *Proceedings of the International Conference on Software Engineering*, pp. 802–811 (2013).

[10] Tan, S. H. and Roychoudhury, A.: relifix: Automated Repair of Software Regressions, in *Proceedings of the International Conference on Software Engineering*, pp. 471–482 (2015).

[11] Deb, K., Pratap, A., Agarwal, S. and Meyarivan, T.: A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II, *IEEE Transactions on Evolutionary Computation*, Vol. 6, No. 2, pp. 182–197 (2002).

[12] Yuan, Y. and Banzhaf, W.: ARJA: Automated Repair of Java Programs via Multi-Objective Genetic Programming, *IEEE Transactions on Software Engineering*, Vol. 46, No. 10, pp. 1040–1067 (2020).

[13] Yuan, Y. and Banzhaf, W.: Toward Better Evolutionary Program Repair: An Integrated

Approach, *ACM Transactions on Software Engineering and Methodology*, Vol. 29, No. 1, pp. 1–53 (2020).

[14] Starkweather, T., McDaniel, S., Mathias, K. E., Whitley, L. D. and Whitley, C.: A Comparison of Genetic Sequencing Operators, in *Proceedings of the International Conference on Genetic Algorithms*, pp. 69–76 (1991).

[15] Le Goues, C., Nguyen, T., Forrest, S. and Weimer, W.: GenProg: A Generic Method for Automatic Software Repair, *IEEE Transactions on Software Engineering*, Vol. 38, No. 1, pp. 54–72 (2012).

[16] Le Goues, C., Dewey-Vogt, M., Forrest, S. and Weimer, W.: A systematic study of automated program repair: Fixing 55 out of 105 bugs for $8 each, in *Proceedings of the International Conference on Software Engineering*, pp. 3–13 (2012).

[17] Boussaïd, I., Lepagnot, J. and Siarry, P.: A survey on optimization metaheuristics, *Information Sciences*, Vol. 237, pp. 82–117 (2013).

[18] AtCoder, : `https://atcoder.jp`.

[19] AtCoder Beginner Contest 102 task A, : `https://atcoder.jp/contests/abc102/tasks/abc102_a`.

[20] Watanabe, H., Matsumoto, S., Higo, Y., Kusumoto, S., Kurabayashi, T., Kirinuki, H. and Tanno, H.: Applying Multi-Objective Genetic Algorithm for Efficient Selection on Program Generation, in *Proceedings of the Asia-Pacific Software Engineering Conference*, pp. 515–519 (2021).

[21] Weiser, M.: Program Slicing, *IEEE Transactions on Software Engineering*, Vol. SE-10, No. 4, pp. 352–357 (1984).

[22] Weiser, M.: Programmers Use Slices When Debugging, *Communications of the ACM*, Vol. 25, No. 7, pp. 446–452 (1982).

[23] Korel, B. and Laski, J.: Dynamic slicing of computer programs, *Journal of Systems and Software*, Vol. 13, No. 3, pp. 187–195 (1990).

[24] 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).

[25] TinyPDG, : `https://github.com/YoshikiHigo/TinyPDG`.

[26] Burke, E. K., Gustafson, S. and Kendall, G.: Diversity in Genetic Programming: An Analysis of Measures and Correlation With Fitness, *IEEE Transactions on Evolutionary Computation*,

Vol. 8, No. 1, pp. 47–62 (2004).

[27] Poon, P. W. and Carter, J. N.: Genetic Algorithm Crossover Operators for Ordering Applications, *Computers & Operations Research*, Vol. 22, No. 1, pp. 135–147 (1995).

[28] Applegate, D. L., Bixby, R. E., Chvátal, V. and Cook, W. J.: *The Traveling Salesman Problem A Computational Study*, Princeton University Press (2007).

[29] Larranaga, P., Kuijpers, C. M. H., Murga, R. H., Inza, I. and Dizdarevic, S.: Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators, *Artificial intelligence review*, Vol. 13, pp. 129–170 (1999).

[30] García-Pedrajas, N., Ortiz-Boyer, D. and Hervás-Martínez, C.: An alternative approach for neural network evolution with a genetic algorithm: Crossover by combinatorial optimization, *Neural Networks*, Vol. 19, No. 4, pp. 514–528 (2006).

[31] Ahvanooey, M. T., Li, Q., Wu, M. and Wang, S.: A Survey of Genetic Programming and Its Applications, *KSII Transactions on Internet and Information Systems*, Vol. 13, No. 4, pp. 1765–1794 (2019).

[32] Oliveira, V. P. L., Souza, E. F. D., Le Goues, C. and Camilo-Junior, C. G.: Improved Crossover Operators for Genetic Programming for Program Repair, in *Proceedings of the Search Based Software Engineering*, pp. 112–127 (2016).

[33] Le Goues, C., Weimer, W. and Forrest, S.: Representations and Operators for Improving Evolutionary Software Repair, in *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 959–966 (2012).

[34] 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：自動プログラム生成に対する多目的遺伝的アルゴリズムの導入：相補的な個体選択を目的として, 情報処理学会論文誌, Vol. 63, No. 10, pp. 1564–1573 (2022).

[35] Martinez, M. and Monperrus, M.: Astor: Exploring the Design Space of Generate-and-Validate Program Repair beyond GenProg, *Journal of Systems and Software*, Vol. 151, pp. 65–80 (2019).

[36] Smith, E. K., Barr, E. T., Le Goues, C. and Brun, Y.: Is the Cure Worse Than the Disease? Overfitting in Automated Program Repair, in *Proceedings of the Joint Meeting on Foundations of Software Engineering*, pp. 532–543 (2015).