情報処理学会論文誌 Vol.63 No.10 1564–1573 (Oct. 2022)

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

渡辺 大登[1,a)] 松本 真佑[1,b)] 肥後 芳樹[1] 楠本 真二[1,c)] 倉林 利行[2] 切貫 弘之[2] 丹野 治門[2]

受付日 2021年12月17日, 採録日 2022年7月6日

概要：自動プログラム生成（APG）の実現を目指し，生成と検証に基づく自動プログラム修正（APR）を転用した手法が提案されている．APR はバグを含むソースコードをすべてのテストケースに通過するように全自動で改変する技術である．APR を転用した APG では，初期状態のソースコードを未実装，つまり複数のバグが含まれていると仮定し，ソースコードの改変，評価，選択を繰り返してソースコードを目的の状態に近づけていく．一般的な APR では改変ソースコードの評価指標として，テストケース通過数がよく用いられる．この指標は単一バグの修正を目的とした場合には問題にならないが，複数バグの修正時にはコード評価の表現能力不足という問題につながる．よって，初期状態に複数バグの存在を仮定する APG においては，解決すべき重要な課題である．そこで，本研究では APG の成功率改善を目的とした多目的遺伝的アルゴリズムの適用を提案する．また，多目的遺伝的アルゴリズムによる高い個体評価の表現能力を利用した，相補的なテスト結果の 2 個体を選択的に交叉する手法も提案する．評価実験として，プログラミングコンテストの問題 80 問を題材に提案手法の効果を確かめた結果，成功率の有意な向上を確認した．

キーワード：自動プログラム生成，自動プログラム修正，多目的遺伝的アルゴリズム

Applying Multi-objective Genetic Algorithm to Improve Automated Program Generation by Selecting Complementary Variants

Hiroto Watanabe[1,a)] Shinsuke Matsumoto[1,b)] Yoshiki Higo[1] Shinji Kusumoto[1,c)]
Toshiyuki Kurabayashi[2] Hiroyuki Kirinuki[2] Haruto Tanno[2]

Received: December 17, 2021, Accepted: July 6, 2022

Abstract: Automated program generation (APG) is a concept of automatically making a computer program. Toward this goal, transferring automated program repair (APR) to APG can be considered. APR modifies the buggy input source code to pass all test cases. APR-based APG regards empty source code as initially failing all test cases, i.e., containing multiple bugs. Search-based APR repeatedly generates program variants and evaluates them. Many traditional search-based APR systems evaluate the fitness of variants based on the number of passing test cases. However, when source code contains multiple bugs, this fitness function lacks the expressive power of variants. In this paper, we propose an application of a multi-objective genetic algorithm to APR in order to improve efficiency. We also propose a new crossover method that combines two variants with complementary test results, taking advantage of the high expressive power of multi-objective genetic algorithms for evaluation. We tested the effectiveness of the proposed method on competitive programming tasks. The obtained results showed significant differences in the number of successful trials and the required generation time.

Keywords: automated program generation, automated program repair, multi-objective genetic algorithm

[1] 大阪大学大学院情報科学研究科
Graduate School of Information Science and Technology, Osaka University, Suita, Osaka 565–0871, Japan
[2] 日本電信電話株式会社
Nippon Telegraph and Telephone Corporation, Minato, Tokyo 108–0075, Japan

[a)] h-watanb@ist.osaka-u.ac.jp
[b)] shinsuke@ist.osaka-u.ac.jp
[c)] kusumoto@ist.osaka-u.ac.jp

1. はじめに

人手を介さない完全自動によるプログラムの生成を目指した研究が進められている [6], [30]．その実現手法の 1 つとして，生成と検証に基づく自動プログラム修正 [26]（Automated Program Repair，APR [*1]）を転用した方法がある [31]．APR はバグを含むソースコードと対応するテストケースを入力とし，自動的なバグ箇所の特定 [27]，およびバグ箇所に対するソースコードの改変 [14] を繰り返して，全テストケースを通過する，すなわちバグのないソースコードを生成する．自動プログラム生成（Automated Program Generation，APG）に対する APR の転用においては，初期状態でのソースコードが完全に未実装であり，これを全テストが失敗する，つまり多数のバグを含んだ状態であるととらえることで，テストケースを 1 つずつ通過するよう探索的にソースコードを進化させる．

APR はこの 10 年間で数多くの研究が実施されている [8] が，理論と実用の両面に対して多くの課題が指摘されている．具体的な課題としては，探索空間が巨大であり修正に多くの時間を要する [13]，入力テストケースへの過剰適合が発生する [22]，生成ソースコードの可読性が低く開発者に受け入れられない [18]，複数のバグを含むプログラムの修正が難しい [21]，などがあげられる．先述のとおり，APG では初期のソースコードが多数のバグを含んだ状態であり，APR の APG への転用という観点では，複数バグの修正という課題の解決が必須である．

本稿では複数バグという課題に対し，APR の選択時における評価値の改善について考える．APR では遺伝的アルゴリズム [25]（Genetic Algorithm，GA）などの探索的手法に基づき，ソースコード改変（個体の生成）とテスト実行による評価（個体の検証）を繰り返す．個体の検証では，生成した個体それぞれについて，バグ修正という目的に対する改善度合いを計測し評価値とする．その後，評価値に基づき次の探索に用いる優秀な個体を選択する．

この選択における 1 つの問題は評価値の計算方法にある．多くの APR 技術では評価値をテスト通過数という単一のスカラ値としている [9]．この指標は単一バグを対象とする場合には問題とならないが，複数バグの場合には表現能力の不足という問題につながる．たとえば，ある個体が 3 つのテストケースのうち 2 つを通過していた場合を考える．この個体を oox（テスト 2 つの通過を意味）と表記する．ここで，個体生成で得られた別の 2 つの個体 oxx と xxo におけるテスト通過数は 1 となり完全に同値である．しかし，oox を補完する改変情報を持つという観点では，xxo を優先的に選択するべきである．

本研究では，APR を転用した APG の成功率改善を目的として，APR への多目的遺伝的アルゴリズム [5]（Multi-Objective Genetic Algorithm，MOGA）の適用を提案する．MOGA は，複数の評価関数を用いて生成個体を評価する GA の一種であり，単一のスカラ値と比べて高い表現能力を持つ個体評価が可能となる．また複数評価関数の採用により，局所解の回避も期待できる [10] ため，巨大な探索空間を持つ APG との高い親和性を期待できる．さらに，本研究では MOGA 適用により得られた，相補的なテスト結果を持つ 2 個体を合成する新たな交叉方法も提案する．評価実験では，プログラミングコンテストの問題 80 問を題材として，提案手法と既存の APR ツールの成功率を比較した．その結果，成功率の有意な向上を確認した．

2. 準備

本章では，生成と検証に基づく APR を転用した探索的な APG の流れと，既存手法の課題を具体例を用いて説明する．

2.1 自動プログラム生成の流れ

図 1 に生成と検証に基づく APR を転用した探索的な APG の流れを表す．入力は空プログラムとテストケース，出力は入力された全テストケースを通過するプログラムである．この手法は個体の生成，評価，選択という 3 つの処理（1 世代）の繰り返しからなる．以下では，APR 分野のブレイクスルーとなった GenProg [9] が用いる GA に基づいた手法について説明する．

まず，入力プログラムにわずかな改変を加え，個体を生成する．この個体の生成方法には変異と交叉の 2 種類がある．変異はある 1 つの個体から新しい個体を生成する操作

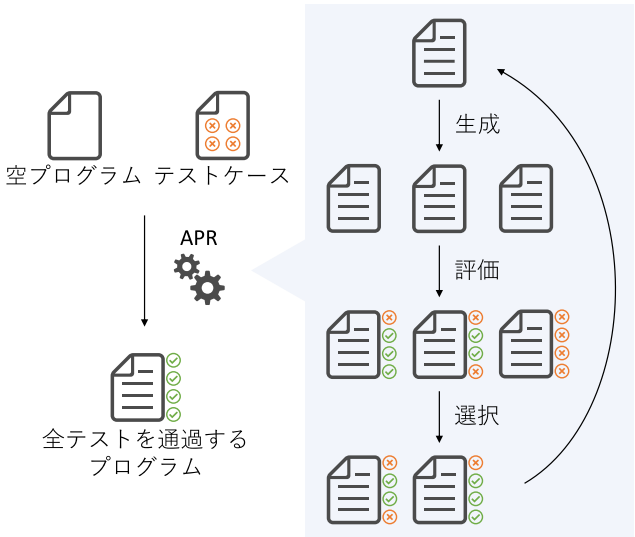


図 1 生成と検証に基づく APR を転用した APG
Fig. 1 Overview of transferring GA-based APR to APG.

[*1] 自動プログラム修正は，生成と検証ベース以外にも意味論ベースの手法も存在するが，本稿では生成と検証ベースの 1 つである遺伝的アルゴリズムを用いた手法を対象とし，簡略化のためにこれを単に APR と略す．

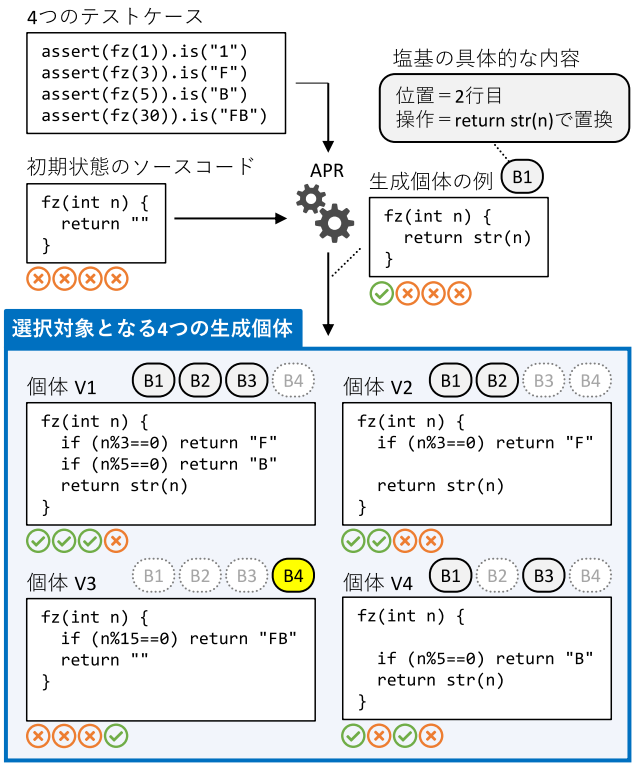


図 2 APR における個体選択の課題
Fig. 2 Challenge in selection phase.

である．変異の操作として最も単純なものはプログラム文の挿入，削除，置換である．交叉は 2 つ以上の個体から新しい個体を生成する操作であり，生成元となる 2 つ以上の個体の改変履歴（遺伝子）を再利用して個体を生成する．GenProg においては，交叉の生成元となる個体は 2 個体であり，それらは乱択によって選ばれる．既存の交叉法として，遺伝子上のある点を交叉点とし交叉点より後ろの遺伝子を交換する一点交叉や，遺伝子の各部分ごとに交換するか否かを定める一様交叉が存在する [1]．次に，生成した各個体がバグ修正という目的にどの程度近づいたかを計測し評価値とする．この評価値はテストケースの実行結果や入力プログラムとの構文や意味的な距離 [4] などから算出する．GenProg は，個体の評価値としてテストケース通過数を用いている．このようにして算出した評価値に基づき次の世代に残すべき個体を選択する．

2.2 既存手法の課題

既存手法の課題を説明するために，FizzBuzz を題材とした具体的なテスト，および生成個体の例を図 2 に示す．図では 4 つのテストケース，および最低限のコンパイルのみが成功する初期状態のソースコードを APR に入力している．APR では図右上に示すように，個々の生成個体が塩基の集合から構成される単一の遺伝子を持つ．塩基と遺伝子は様々な設計が可能であるが，ここでは塩基は位置と操作の 2 つの要素から構成されている．図の右上の個体は，単一の塩基 B1 からなる遺伝子を持っており，その塩基の位置は 2 行目 `return ""`の箇所，塩基の操作内容は `return str(n)` による置換である．この遺伝子を適用した結果，4 つのテストすべてが失敗する状態から，1 つ目のテストが通過する状態に進化している．

ここで，APR 処理の中で 4 つの個体（V1〜V4）が生成されたとする．左上の V1 は最も巨大な遺伝子（つまり多くの塩基）を含んでおり，通過テスト数も 3 個と最も多い．よって V1 は次のソースコードの改変ループに生かすべき，最も良い個体であると解釈できる．APR のような探索問題では，単一個体ではなく複数の個体を選択することが一般的である．これは遺伝子の多様性確保，および局所解を回避するための戦略である．そのために，V1 の次に選択すべき個体を考える必要がある．

APR 研究で広く用いられるテスト通過数というスカラ値を評価値として用いた場合，テスト通過数 2 である V2 と V4 が V1 に続いて選択される．しかしテスト通過の内容を確認すると，この 2 個体は V1 の完全なサブセットとなっており，効率的な選択とはいえない．他方，左下の個体 V3 はテスト通過数自体は 1 と最も少ないものの，他の個体では失敗する 4 つ目のテストを通過している．また，V3 は他の個体にはない特殊な遺伝子 B4 により，FizzBuzz 問題における 15 倍数時の処理を内包している．よって，V1 の次に選択すべき個体は V3 であると結論できる．このように，APR の個体評価においてテスト通過数という評価値はその表現能力が不足しており，個々のテストの成否という情報を個体ごとに比較する必要がある．

3. 提案手法

本稿では APR を転用した APG の成功率改善を目的として，選択と交叉に特化した以下の 3 手法を提案する．

提案 X 多目的遺伝的アルゴリズムの適用
提案 Y 相補的な個体の選択的交叉
提案 Z 交叉によって生成された個体の検証

提案手法は直感的には次のように解釈できる．GA による進化の流れにおいて，MOGA により相補的な個体を選択し（提案 X），その相補的な個体に限定した交叉を適用し（提案 Y），交叉の結果が親の上位であるかを確認（提案 Z）する．3 つの手法はいずれも直前の手法の適用を前提としている．たとえば提案 Y は提案 X の適用を，提案 Z は X と Y 両方の適用を前提としている．

3.1 提案 X：多目的遺伝的アルゴリズムの適用

MOGA とは，複数の評価関数を用いて個体を評価，選択する特殊な GA である．以下では，MOGA に必要な複数の評価関数の設計と個体の選択方法を説明する．

3.1.1 評価関数と個体の評価ベクトル

生成した個体の評価関数について説明する．2.2 節で説明した選択の課題を解決するために，個々のテストの成否を独

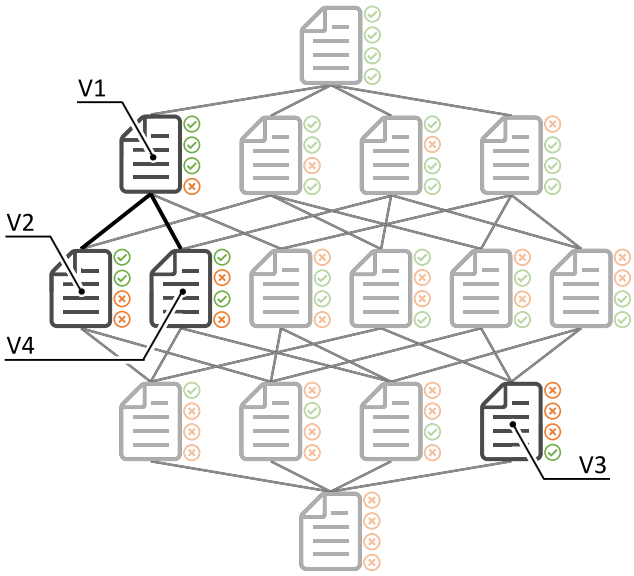


図 3 テスト数 4 のときの全個体のハッセ図と V1 から V4 の関係
Fig. 3 Hasse diagram of all variants with four test cases, and the relation between V1–V4.

立した評価関数とする．つまり，テストが M 個入力されたとき，評価関数は M 個となる．i 個目の評価関数 $E_i(v)$ は個体 v が i 個目のテストを通過すれば 1 を，失敗すれば 0 を返す．個体 v の評価ベクトルは $(E_1(v), E_2(v), \cdots, E_M(v))$ となる．M 個の評価関数の値がすべて 1 となる個体が生まれたとき，すなわち，評価ベクトルのすべての成分が 1 のとき，すべてのテストケースを通過する個体の生成を意味する．

3.1.2 個体の優越と順序関係の定義

個体 v_a，v_b に関して，$E_i(v_a) < E_i(v_b)$, $\forall i \in \{1, \ldots, M\}$ のとき，v_b は v_a を優越するという．また，このとき，$v_a \prec v_b$ として個体の順序関係 $\prec$ を定義する．このとき，$\prec$ は半順序関係であり，個体の集合は束[*2]となる．

3.1.3 次世代へ残す個体の選択

提案する選択では，各個体の評価ベクトルを元に次の世代に残す個体を決定する．具体的には，パレートランキング法 [7] により変異による生成個体と交叉による生成個体の和集合に対しランク付けを行い，個体数が上限となるまで，ランクに応じて個体を選択する．ここで，個体 v のランクは v が属する世代中の $v \prec v'$ となる個体 v' の個数 +1 とする．

図 3 に $M = 4$ での存在しうる全個体の関係をハッセ図 [3] で示す．本図では $v_a \prec v_b$ かつ $v_a \prec v_c \prec v_b$ なる v_c が存在しない場合のみ v_a から v_b に上向きに線を引いた．図中の最下段の個体は初期個体を含むすべてのテストケースに失敗した個体を表し，最上段の個体はすべてのテストを通過する生成目標の個体である．本図の個体がすべて同一世代中に存在する場合，各個体のランクは，最上段の個体が 1，2 段目の 4 つの個体が 2，3 段目の 6 つの個体が 4，4 段目の 4 つの個体が 8，最下段の個体が 16 となる．

図 2 に示した個体のランクは V1 と V3 が 1，V2 と V4 が 2 となる．V1 と V3 は図 3 において，上部に個体が存在しないが，V2 と V4 の上部には V1 が存在するからである．このランクによって，V3 を V2，V4 に優先して選択できる．

3.2 提案 Y：相補的な個体の選択的交叉

本研究では，前節で述べた MOGA の適用に加えて交叉の改善も行う．従来手法において交叉の対象となる 2 個体は乱択によって選ばれる．しかし，複数バグの存在を前提としたとき，乱択による選択には有効性の低い優越関係にある個体間での交叉が起こりうるという課題が存在する．

まず，優越関係にある 2 個体の交叉が有効でないことを例をあげて説明する．図 2 に示した V1 と V2 の交叉による新たな個体の生成を考える．このとき，生成された個体が V1 より多くのテストを通過する可能性は低いと考えられる．これは，2.2 節で述べたように，V2 が V1 のサブセットだからである．

乱択によって交叉個体を選択する場合，優越関係にある個体間の交叉が発生し探索効率の悪化を招く．この課題を解決するために，提案 Y では交叉対象の個体を優越関係にない個体（相補的な個体）に限定する．具体的には，ランク 1 の個体群から，評価ベクトルが一致しない 2 組の個体に対してのみ交叉を行う．このような交叉により，2 個体の通過テストすべてを通過する個体の生成が期待できる．

図 2 に示した 4 個体の場合，ランク 1 の個体は V1 と V3 であり，これら 2 個体の評価ベクトルは不一致である．よって，これらに対してのみ交叉を行う．

3.3 提案 Z：交叉によって生成された個体の検証

一般的な GA に基づいた APR では，交叉によって生成された個体群は変異による個体群と同等の評価関数で評価される．すなわち，両方の個体群はテスト通過数の改善のみで優劣が付けられる．

提案 Z では交叉で生成した個体に対して，新たな検証を追加する．個体 v_a，v_b から交叉により個体 v_c が生成されたとき，$v_a \prec v_c$ かつ $v_b \prec v_c$ を満たす場合のみ，生成した個体 v_c を 3.1.3 項で述べた次の世代への選択処理の対象とする．この理由は前節で述べたように，交叉は対象となる 2 個体の通過テストすべてを通過する個体，すなわち，それら 2 個体を優越する個体の生成を目標とするからである．V1 と V3 の交叉により生成された個体は，4 つすべてのテストを通過する場合に限り次の世代への選択対象となり，ランク付与が行われる．

[*2] Lattice

表 1 実験設定
Table 1 Experimental settings.

項目	値
実験題材	ABC101～ABC180 100 点問題
問題数	80
乱数シード	1～20（= 20 試行）
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
終了条件	正解個体の発見・時間切れ
実験環境	Xeon E5-2630 2CPUs 16 GB mem

4. 実験

本章では，提案手法を既存の APR ツールである kGenProg [24] に実装して，その効果を確認する．拡張前の kGenProg を従来手法として，kGenProg に提案 X，提案 X と Y，すべての提案（XYZ）をそれぞれ実装し，これら 4 種での比較を行う．本実験の目的は，提案 X，Y，Z による APG の成功率の変化を確認することである．評価指標として，全テストを通過する個体の生成率（成功数）とプログラム生成の所要時間（生成時間）を用いる．

4.1 実験設定

実験設定の一覧を**表 1** に示す．実験の題材として，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 問を用いる．従来手法と提案手法はともに GA に基づきプログラムを生成するため，個体の生成や選択などの各段階において乱択的要素が含まれる．よって，乱数シードに 1 から 20 までの 20 個を設定してプログラムの生成を試みる．

従来手法，および提案手法の拡張元である kGenProg は再利用に基づく APR ツールであり，個体の生成時にはプログラム改変の材料となるソースコード片が必要である．この再利用ソースコード片として，実験題材 80 問の正解コードに含まれる全ステートメントを利用した．これにより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生成が可能かを確かめる．なお，再利用コードには正解コードを構成する全ステートメントが含まれるため[*4]，十分な時間をかければ，理論的には正解コードを生成できる．両ツールに入力するテストケースは AtCoder 社の公開するテストケースを利用した．テストケースは入力に対して出力が適切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．また問題あたりのテストケースの平均は 10.9 個である．その他の GA の動作に必要なパラメータは kGenProg バージョン 1.8.0 の既定値[*5]を用いた．

[*3] https://atcoder.jp/
[*4] 通常，APG の利用時には必ずしも正解コードが存在するとは限らないが，本実験では両ツールに同じ再利用コード片を入力するため，平等な実験設定といえる．

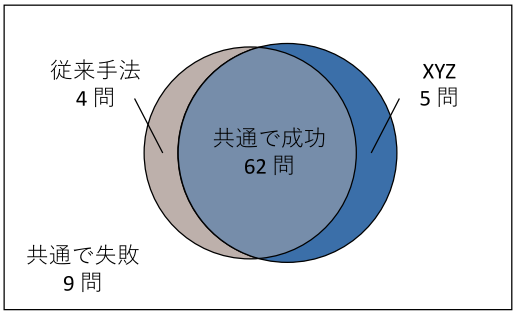


図 4 生成に成功した問題数のベン図
Fig. 4 Venn diagram of successful tasks.

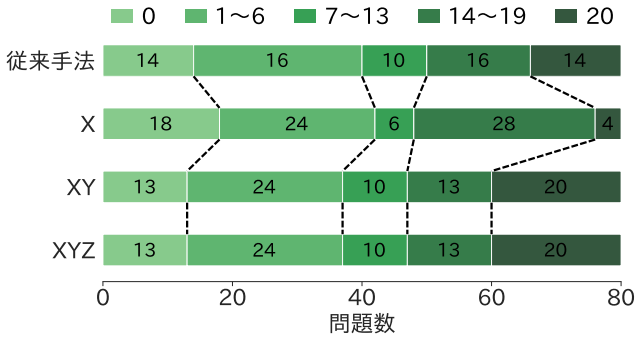


図 5 成功試行数に対する問題数の比較
Fig. 5 Comparison of tasks by successful trials.

4.2 実験結果

4.2.1 成功数の比較

従来手法と提案 XYZ それぞれに対する生成に成功した問題数を**図 4** に示す．ここで，生成に成功した問題とは全 20 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．図中の左側の円は従来手法で生成に成功した問題の集合を表し，右側の円は提案 XYZ で生成に成功した問題の集合を表す．従来手法と提案 XYZ の両方で生成に成功した問題は 62 問，従来手法でのみ生成に成功した問題は 4 問，提案 XYZ でのみ生成に成功した問題は 5 問，両者ともに生成できなかった問題は 9 問であった．両者で生成可能な問題の傾向は強く似通っている．これは変異による個体生成処理に変化がないことが原因と考えられる．

成功試行数に対する問題数を**図 5** に示す．本図では，各問題を成功試行数によって凡例に示す 5 グループへ分割した．たとえば，左端の試行数 0 のグループは 20 試行中で 1 度も生成に成功しなかった問題数を，右端の試行数 20 のグループは全試行で生成に成功した問題数を表す．図より，提案 X のみを含む場合に成功試行数 0 の問題の増加や成功試行数 20 の問題の減少が読み取れる．また，従来手法と

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

比較して提案 XY や XYZ では成功試行数 20 の問題が増加していることが分かる．

成功試行数の変化を定量的に観察するために，生成に成功した試行数の合計を**表 2** に示す．表では，各問題における正解コードに分岐が必要か否かで成功試行数を分割した．分岐が不要な問題では，提案手法による成功試行数への大きな増加は読み取れず，提案 X のみを含む場合には成功数が 50 減少した．その一方，分岐が必要な問題では，成功数の大幅な増加が読み取れる．また，提案 Z は分岐が必要な問題においてのみ成功試行数が増加し，その差は 8 試行であった．

従来手法と提案 XYZ について，各問題の代表値を生成に成功した試行数として Wilcoxon の符号順位検定により有意差を検定したところ，$p = 2.21 \times 10^{-4}$ であり，有意差が認められた．また，その効果量は $r = 2.92 \times 10^{-1}$ であった．

表 2 生成に成功した試行数の合計
Table 2 The number of successfully generated trials.

分岐	従来手法	提案 X	提案 XY	提案 XYZ
不要	528	478	535	535
必要	202	221	268	276
合計	730	699	803	811

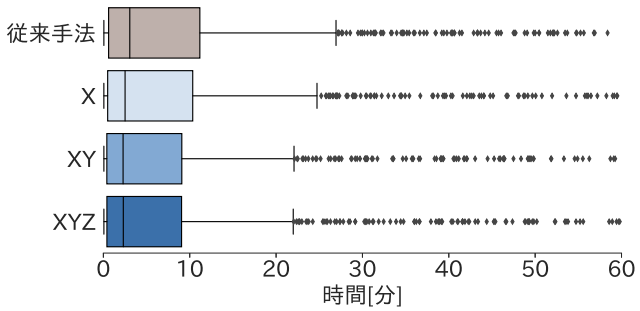


図 6 生成時間の比較
Fig. 6 Comparison of generating time.

4.2.2 生成時間の比較

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，全 1,600 試行のうち生成に成功した試行のみを抽出している．また，ひげの長さの上限を四分位範囲の 1.5 倍とし，上端より大きい値を外れ値とした．横軸は生成時間を表し，短いほどプログラム生成の効率が高いといえる．図より，提案 XY と XYZ はほとんど同等であり，XYZ，XY，X，従来手法の順に生成時間が短い傾向を確認できる．

各問題に対する生成時間を確認するために，従来手法と提案 XYZ における共通で成功した 62 問それぞれに対する生成時間を**図 7** に示す．図の横軸は問題名であり，縦軸は成功時の平均生成時間である．横軸の問題名において，提案 XYZ の生成時間が従来手法より短い問題を青字で表している．提案 XYZ がより短い時間で生成に成功した問題は 42 問，従来手法がより短い時間で生成に成功した問題は 20 問であった．

従来手法と提案 XYZ 間で成功時の生成時間に関して有意差検定を行った．代表値を平均値として Wilcoxon の符号順位検定により有意差を検定したところ，$p = 2.83 \times 10^{-3}$ であり，有意差が認められた．

5. 考察

5.1 提案 X の効果

提案 X の効果を従来手法との比較により考察する．まず，成功試行数について考える．図 5 より，X における成功試行数 0 の問題の増加，成功試行数 20 の問題の減少が読み取れる．さらに表 2 より，この減少は分岐が不要な問題において発生したと理解できる．提案 X による分岐を必要としない問題における成功試行数の減少原因を考察する．

提案 X により成功試行数が減少した例として ABC116 をあげる．ABC116 は直角三角形の 3 辺の長さを入力としその面積を出力する，分岐が不要な問題である．正解となる擬似コードを**図 8** に，提案 X を含む APG が出力した疑似コードを**図 9** に示す．提案 X によって，不要な分岐の

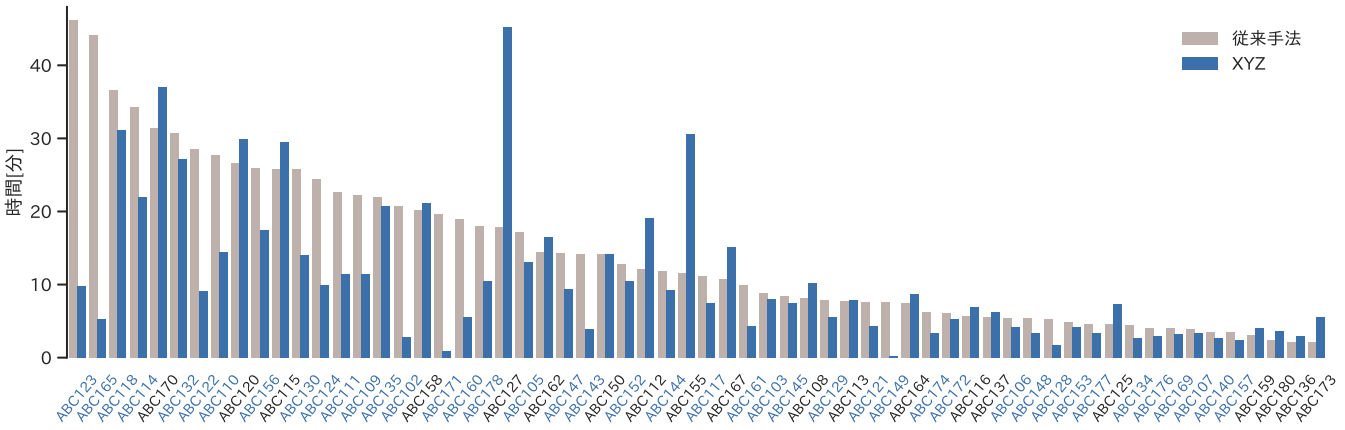


図 7 各問題に対する生成時間の比較
Fig. 7 Comparison of generation time for each task.

```
int abc116(int a, int b, int c) {
  return a * b / 2;
}
```

図 8 ABC116 の正解コード
Fig. 8 Correct code of ABC116.

```
int abc116(int a, int b, int c) {
  if (a >= 30) {
    return a * b / 2;
  }
  if (a < 30) {
    return a * b / 2;
  }
  return 0;
}
```

図 9 提案 X を含む APG の出力
Fig. 9 Output of APG impremented the propose A.

ために一部のテストケースのみ通過する個体も種類が減少しないように選択される．たとえば，図 9 の 1 つ目の `if` 分岐のみを持つ個体や，2 つ目の `if` 分岐のみを持つ個体，あるいは，不要な分岐を多重に含む個体もそれらの個体を優越する個体が生成されるまで淘汰されない．これは，提案 X の目的である個体の多様性向上の結果である．しかし，この戦略は分岐が不要な問題に対しては有利には働かず，成功試行数を減少させたと考えられる．

次に，提案 X のオーバヘッドがもたらす生成時間への影響を考える．提案 X の優越関係による半順序を利用した選択の時間計算量は $O(n^2)$ であり，従来手法の選択 $O(n \log n)$ と比較して計算量が大きい．しかし，図 6 より，X では生成時間がわずかに減少している．APG では，個体評価時にテストケース通過の可否を得るために個体をビルドしテストを実行する．このビルドとテスト実行にかかる時間など，APG の他の段階にかかる処理時間が提案 X と比較して十分に大きいため，生成時間が増加しなかったと考えられる．

5.2 提案 Y の効果

提案 Y の効果を従来手法，提案 X，提案 XY の比較から考察する．生成に成功した試行数の観点では，表 2 より分岐の有無にかかわらず従来手法と提案 X と比較して成功試行数が増加した．個体の多様性を維持（提案 X）したうえで，相補的な 2 個体を選択的に交叉する提案 Y が有効に働いたといえる．つまり，提案 Y による正解コードに必要な分岐のマージや不要な交叉の抑制により試行数が向上した．また，5.1 節で述べた提案 X の課題も提案 Y によって解決している．生成時間についても，図 6 から効率が向上しているといえる．

5.3 提案 Z の効果

提案 Z の効果を提案 XY，提案 XYZ の比較から考察する．表 2 より，生成に成功した試行数は分岐が不要な問題では改善が見られず，分岐が必要な問題では試行数がわずかに増加した．生成時間の観点からも，提案 XY と大きな差はない．これらから，提案 Z による生成効率の向上は見られなかったといえる．この理由は実験題材の単純さにある．

交叉による生成個体が全テストに通過する場合，提案 Z は実行されない．全テストに通過する個体が生成されたとき，その個体を出力し GA が終了するためである．交叉の生成元となる 2 個体は提案 Y より異なるランク 1 の個体ゆえ，提案 Z が実行されるには題材の複雑さが必要である．しかし，本実験で用いた題材では，そのような複雑な問題はわずかであり，全体としては影響を観測できなかったと思われる．

5.4 短時間で生成可能な問題で改善幅が小さい原因

図 7 より，従来手法において生成時間が短い問題ほど提案手法による効率の向上が得られない傾向が読み取れる．この原因は MOGA による高い個体評価能力が短時間で生成可能な問題では生かされなかったからと考えられる．生成に時間のかからない問題の一例として，正解プログラムが持つコード片の個数が少ない問題をあげる．必要コード片の少ない問題はテストケースの観点数が必然的に少ない．また，1 度の操作ですべてのテストに通過する個体を生成できることもある．たとえば，図 8 で示した ABC116 の必要コード片は `return a*b/2` のみであり，この 1 つのコード片を挿入できれば全テストに通過する個体を生成できる．このような問題では評価値に高い表現性は不要であり，従来手法と提案手法の差がつかなかったと考えられる．

6. 関連研究

機械による全自動でのプログラム生成は，コンピュータサイエンス分野における 1 つの大きな夢である．この問題は自動プログラミング [20] と呼ばれ，数多くの研究者がその実現に取り組んできた [15], [16]．自動プログラミングの実現手段は，以下の要素の組合せで大別できる．

- 機械に対して「何をして欲しいか」というオラクル・要求をどう表現するか（自然言語の文章 [12]，DSL [19]，入出力例 [17] など）
- どのような手段でプログラムを作成するか（翻訳ベース [12]，探索ベース [23] など）

本研究で取り組んだ APG は遺伝的プログラミング [11] とも呼ばれており，テストスイートをオラクルとした探索ベースの自動プログラミング手法であるといえる．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 [31]．本研究とは提案

手法そのものが異なっており，GA の評価関数の改善が本研究の主眼であるのに対し，先行研究の提案は，生成対象となるプログラム構造の推論と再利用ステートメントの前処理である．

Yuan らは MOGA を適用した APR ツールとして ARJA を提案している [28]．ARJA はテスト通過数とパッチ行数の 2 つの評価関数を持つ．パッチ行数を新たに評価関数として導入した目的は，生成コードの可読性向上にある．他方で，本研究では可読性という別軸の評価関数を導入するのではなく，全テストケース通過という 1 つの目的を部分問題へ分割することで多目的化[*6] [10] し，MOGA を適用している．この観点から，提案手法と ARJA は MOGA 適用の意図が異なる．

また，個体評価の表現力向上に取り組んだツールとして，ARJAe [29] と 2Phase [2] がある．ARJAe は ARJA の拡張であり，ARJA の持つ 2 つの評価関数に加えて，テストケースの期待値と出力値の差を新たな評価関数として MOGA を適用している．2Phase もテスト通過数および，期待値と出力値の差から評価値を計算するが，ARJAe とは異なり，それら 2 つの加重平均を評価値とする単目的 GA を用いている．これらのツールと提案手法の違いは個体評価の方向性にある．ARJAe や 2Phase は絶対評価によって個体を評価する一方で，提案手法は相対評価による評価の改善に取り組んでいる．図 2 で説明した個体選択における課題は，個体の絶対評価では解決できず，相対評価によって解決を試みた点でこれらの手法と提案手法は異なる．

7. 妥当性の脅威

4 章での実験について，妥当性の脅威が存在する．内的妥当性への脅威として，プログラムの生成材料として正解コードを利用したことがあげられる．4.1 節で述べたように，平等性を確保するため比較対象のツールに同じ生成材料を与えたが，実際の APG 利用時を想定した正解コードを利用しない実験では異なる結果が得られる可能性がある．

外的妥当性への脅威としては，実験題材や利用ツールが実験結果に影響を与えている可能性がある．提案手法はテストケースの粒度や適切さに影響を受けるため，AtCoder 以外の題材での実験を行う必要がある．また，提案手法を kGenProg に実装し実験を行ったが，GenProg など他のツールを用いた実験も必要である．

8. おわりに

本稿では，個々のテスト結果を独立した評価関数とする MOGA を適用した自動プログラム生成手法を提案した．また，プログラミングコンテストの問題 80 問を題材とした評価実験を行い，本手法の有効性を確認した．

今後の課題として，実験結果のさらなる分析があげられる．提案手法において生成時間が著しく悪化した ABC127 など各問題の性質が提案手法に与える影響を分析する必要がある．また，MOGA 適用により局所解の回避という効果も得られるが，この効果が提案手法によって得られたかは確認できていない．この確認のために，生成プログラムが入力テストケースに対して過剰適合しているか分析が必要である．手法の改善という観点では，テスト通過の可否に加え，コード行数やテスト時間などの指標を新たな評価関数として組み込むことも検討している．実験においては AtCoder の最も簡単な 100 点問題を題材としたが，より複雑な題材への適用実験や正解コードを用いない実際の APG 利用時を想定した実験も今後の課題である．

謝辞 本研究の一部は，JSPS 科研費（JP20H04166，JP21H04877，JP21K11829）による助成を受けた．

[*6] Multiobjectivizing

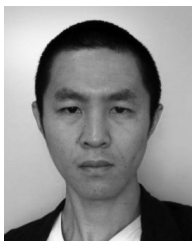
参考文献

[1] Ahvanooey, M.T., Li, Q., Wu, M. and Wang, S.: A Survey of Genetic Programming and Its Applications, *KSII Trans. Internet and Information Systems*, Vol.13, No.4, pp.1765–1794 (2019).
[2] Bian, Z., Blot, A. and Petke, J.: Refining Fitness Functions for Search-Based Program Repair, *Proc. International Conference on Software Engineering Workshops*, pp.1–8 (2021).
[3] Birkhoff, G.: *Lattice Theory*, Vol.25, American Mathematical Society (1940).
[4] D'Antoni, L., Samanta, R. and Singh, R.: Qlose: Program Repair with Quantitative Objectives, *Proc. International Conference on Computer Aided Verification*, pp.383–401 (2016).
[5] Deb, K., Pratap, A., Agarwal, S. and Meyarivan, T.: A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II, *IEEE Trans. Evolutionary Computation*, Vol.6, No.2, pp.182–197 (2002).
[6] Desai, A., Gulwani, S., Hingorani, V., Jain, N., Karkare, A., Marron, M. and Roy, S.: Program Synthesis using Natural Language, *Proc. International Conference on Software Engineering*, pp.345–356 (2016).
[7] Fonseca, C.M. and Fleming, P.J.: Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization, *Proc. International Conference on Genetic Algorithms*, pp.416–423 (1993).
[8] Gazzola, L., Micucci, D. and Mariani, L.: Automatic Software Repair: A Survey, *IEEE Trans. Software Engineering*, Vol.45, No.1, pp.34–67 (2017).
[9] Goues, C.L., Nguyen, T., Forrest, S. and Weimer, W.: GenProg: A Generic Method for Automatic Software Repair, *IEEE Trans. Software Engineering*, Vol.38, No.1, pp.54–72 (2012).
[10] Knowles, J.D., Watson, R.A. and Corne, D.W.: Reducing Local Optima in Single-Objective Problems by Multi-objectivization, *Proc. International Conference on Evolutionary Multi-Criterion Optimization*, pp.269–283 (2001).
[11] Koza, J.R. and Poli, R.: *Genetic Programming*, pp.127–164 (2005).
[12] Liu, H., Shen, M., Zhu, J., Niu, N., Li, G. and Zhang, L.: Deep Learning Based Program Generation From Re-

quirements Text: Are We There Yet?, *IEEE Trans. Software Engineering*, Vol.48, No.4, pp.1268–1289 (2022).
[13] Long, F. and Rinard, M.: An Analysis of the Search Spaces for Generate and Validate Patch Generation Systems, *Proc. International Conference on Software Engineering*, pp.702–713 (2016).
[14] 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).
[15] O'Neill, M. and Spector, L.: Automatic programming: The open issue?, *Genetic Programming and Evolvable Machines*, Vol.21, No.21, pp.251–262 (2020).
[16] O'Neill, M., Vanneschi, L., Gustafson, S. and Banzhaf, W.: Open Issues in Genetic Programming, *Genetic Programming and Evolvable Machines*, Vol.11, No.3–4, pp.339–363 (2010).
[17] Parisotto, E., Mohamed, A., Singh, R., Li, L., Zhou, D. and Kohli, P.: Neuro-Symbolic Program Synthesis, *Proc. International Conference on Learning Representations*, pp.1–15 (2017).
[18] Qi, Z., Long, F., Achour, S. and Rinard, M.: An Analysis of Patch Plausibility and Correctness for Generate-and-Validate Patch Generation Systems, *Proc. International Symposium on Software Testing and Analysis*, pp.24–36 (2015).
[19] Rabinovich, M., Stern, M. and Klein, D.: Abstract Syntax Networks for Code Generation and Semantic Parsing (2017).
[20] Rich, C. and Waters, R.: Automatic Programming: Myths and Prospects, *Computer*, Vol.21, No.8, pp.40–51 (1988).
[21] Saha, S., Saha, R.K. and Prasad, M.R.: Harnessing Evolution for Multi-Hunk Program Repair, *Proc. International Conference on Software Engineering*, pp.13–24 (2019).
[22] Smith, E.K., Barr, E.T., Goues, C.L. and Brun, Y.: Is the Cure Worse Than the Disease? Overfitting in Automated Program Repair, *Proc. Joint Meeting on Foundations of Software Engineering*, pp.532–543 (2015).
[23] Spector, L.: Autoconstructive Evolution: Push, PushGP, and Pushpop, *Proc. Genetic and Evolutionary Computation Conference*, pp.137–146 (2001).
[24] 枩本真佑，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] Weimer, W., Nguyen, T., Le Goues, C. and Forrest, S.: Automatically Finding Patches Using Genetic Programming, *Proc. International Conference on Software Engineering*, pp.364–374 (2009).
[26] Wen, M., Chen, J., Wu, R., Hao, D. and Cheung, S.: Context-Aware Patch Generation for Better Automated Program Repair, *Proc. International Conference on Software Engineering*, pp.1–11 (2018).
[27] Wong, W.E., Gao, R., Li, Y., Abreu, R. and Wotawa, F.: A Survey On Software Fault Localization, *IEEE Trans. Software Engineering*, Vol.42, No.8, pp.707–740 (2016).
[28] Yuan, Y. and Banzhaf, W.: ARJA: Automated Repair of Java Programs via Multi-Objective Genetic Programming, *IEEE Trans. Software Engineering*, Vol.46, No.10, pp.1040–1067 (2020).
[29] 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).
[30] Zhang, S. and Sun, Y.: Automatically Synthesizing SQL Queries from Input-Output Examples, *Proc. International Conference on Automated Software Engineering*, pp.224–234 (2013).
[31] 富田裕也，松本淳之介，枩本真佑，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).

渡辺 大登

2021 年大阪大学基礎工学部情報科学科卒業．同年より同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博士前期課程在学中．自動プログラム生成に関する研究に従事．



枩本 真佑　（正会員）

2010 年奈良先端科学技術大学院大学博士後期課程修了．同年神戸大学大学院システム情報学研究科特命助教．2016 年大阪大学大学院情報科学研究科助教．博士（工学）．エンピリカルソフトウェア工学の研究に従事．

肥後 芳樹　（正会員）

2002 年大阪大学基礎工学部情報科学科中退．2006 年同大学大学院博士後期課程修了．2007 年同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助教．2015 年同准教授．2022 年同教授．博士（情報科学）．ソースコード分析，特にコードクローン分析，リファクタリング支援，ソフトウェアリポジトリマイニングおよび自動プログラム修正に関する研究に従事．電子情報通信学会，日本ソフトウェア科学会，IEEE 各会員．

楠本 真二 （正会員）

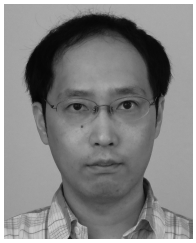
1988 年大阪大学基礎工学部卒業．1991 年同大学大学院博士課程中退．同年同大学基礎工学部助手．1996 年同講師．1999 年同助教授．2002 年同大学大学院情報科学研究科助教授．2005 年同教授．博士（工学）．ソフトウェアの生産性や品質の定量的評価に関する研究に従事．電子情報通信学会，IEEE，IFPUG 各会員．

倉林 利行 （正会員）

2012 年慶應大学理工学部システムデザイン工学科卒業．2014 年同大学大学院理工学研究科総合デザイン専攻博士前期課程修了．同年日本電信電話（株）入社．2017 年度コンピュータサイエンス領域奨励賞（情報処理学会）．2020 年度 IEEE Computer Society Japan Chapter SES Young Researcher Award．2021 年度山下記念研究賞（情報処理学会）．主にプログラム自動生成に関する研究開発に従事．

切貫 弘之 （正会員）

2015 年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博士前期課程修了．同年日本電信電話（株）入社．2015 年度コンピュータサイエンス領域奨励賞（情報処理学会）．2018 年度山下記念研究賞（情報処理学会）．主にリポジトリマイニング，ソースコード解析，ソフトウェアテストに関する研究開発に従事．



丹野 治門 （正会員）

2007 年電気通信大学電気通信学部情報工学科卒業．2009 年同大学大学院電気通信学研究科情報工学専攻博士前期課程修了．同年日本電信電話（株）入社．2020 年電気通信大学大学院情報理工学研究科情報・ネットワーク工学専攻博士後期課程修了．2008 年未踏ユース・スーパークリエータ認定（情報処理推進機構）．2009 年山内奨励賞（情報処理学会）．2013 年山下記念研究賞（情報処理学会）．2016 年企業賞（情報処理学会）．2016 年社長表彰（日本電信電話）．2018 年優秀発表賞，学生奨励賞（日本ソフトウェア科学会）．2020 年企業・ポスター賞（情報処理学会）．ソフトウェアテスト，デバッグに関する研究開発に従事．