自動プログラム修正技術の性能評価 ―九州大学の基幹教育データを用いた事例研究―

中村 司[1,a)]　亀井 靖高[1,b)]　近藤 将成[1,c)]　鵜林 尚靖[1,d)]

概要：現在，様々な自動プログラム修正技術が提案されているが，複数の技術を比較するために利用されるデータセットは限られている．そのため，複数の自動プログラム修正技術を比較した結果がどれほど一般性のあるものなのかは明らかになっていない．そこで，データセットを新たに用意して既存の自動プログラム修正技術を適用し，自動プログラム修正技術の性能を比較した事例を増やしたいと考えた．本研究では，文法エラーを修正する技術として DeepFix と DrRepair の 2 つを取り上げ，九州大学で収集されたデータセットを利用しそれらの修正性能を評価する．さらに，それぞれの技術によってバグを修正できたプログラム群を比較し，それぞれの技術が利用した手法の差が結果に現れているのかを調査する．調査の結果，修正率では DrRepair が大きく上回っていたものの，DeepFix でのみバグを修正できたプログラムと DrRepair でのみバグを修正できたプログラムがともに存在することが明らかになった．また，これら 2 つのプログラム群をソースコードメトリクスの観点から比較したが，大きな差は見られなかった．

キーワード：自動プログラム修正，ソースコードメトリクス，C 言語

1. はじめに

ソフトウェア開発において最もコストのかかる作業の 1 つはデバッグであると言われている．デバッグにかかるコストは開発コスト全体の 50%に及ぶこともあるが，この理由は主に，バグを特定し除去するためには人の手による尽力が必要となるためである [6]．そのため，デバッグにかかるコストを少しでも軽減するための取り組みとして，自動プログラム修正技術に関する研究が現在盛んに行われている．自動プログラム修正技術とは，人の手ではなくコンピュータによってプログラムのエラーを解消することを目指した技術の総称である．

自動プログラム修正技術の実用化に向け，現在様々な手法が提案されている．例えば，DeepFix はバグ修正前後のプログラムを Seq2Seq モデルに学習させ，C 言語の文法エラーに特化した修正を行う [8]．DeepFix を提案した Gupta らの研究では，インド工科大学で収集されたバグありプログラムの 27%を部分的に，19%を完全に修正することに成功している．また，DrRepair はバグ修正前後のプログラムだけでなく，プログラムをコンパイルした際のエラーメッセージも利用してバグ修正用モデルの学習を行う [13]．ソースコードやエラーメッセージに登場するシンボルの関係性をグラフで表現し学習モデルに与えることで，学習モデルが扱うことのできる情報を増やすことができる．DrRepair を提案した Yasunaga らの研究では，Gupta らと同じデータセットを DrRepair の評価に利用し，68.2%という修正率を記録している．

DeepFix を提案する際，Gupta らはインド工科大学によって収集されたプログラムを利用して Seq2Seq モデルの学習や評価を行った．このデータセットはプログラミング初学者向けの講義のなかで学生が提出したソースコードを収集したものであり，DrRepair を始め様々な自動プログラム修正技術の評価に利用されている [1], [9] ．しかし，このようにプログラミング初学者によるソースコードを十分に収集したデータセットで，広く利用されているものは他にない．そのため，インド工科大学とは別の条件で収集されたデータを利用して自動プログラム修正技術の比較を行った際，同様の結果が得られるかは明らかになっていない．

そこで，Gupta らによるデータセットと同様な，プログラミング初学者の学生によるプログラムを収集したデータセットを新しく用意し，既存の自動プログラム修正技術の性能について先行研究と同様の実験結果が得られることを

[1] 九州大学
Kyushu University
[a)] t.nakamura@posl.ait.kyushu-u.ac.jp
[b)] kamei@ait.kyushu-u.ac.jp
[c)] kondo@ait.kyushu-u.ac.jp
[d)] ubayashi@ait.kyushu-u.ac.jp

確かめたいと考えた．Gupta らによるデータセットとともに新規のデータセットを利用することで，自動プログラム修正技術の性能評価に関する一般性を高めることができる．

本研究では，九州大学の学生によって書かれたプログラムに対する各自動プログラム修正技術の性能について，次のような RQ に基づいて調査を行った．

RQ1 各自動プログラム修正技術のいずれかのみによって修正できるプログラムは存在するのか．また，存在するならばそれらに含まれるエラーはどのようなものか．

RQ2 各自動プログラム修正技術によって修正できるプログラム群にはソースコードメトリクスの観点における違いはあるのか．

また，本研究による貢献は次の通りである．

- プログラミングを専門としているわけではない学生によって書かれたプログラムをデータセットとして使用し，先行研究とは異なる条件で各自動プログラム修正技術の性能を評価した．そして，新しく用意したデータセットでも修正率では DrRepair が DeepFix を大きく上回ることが確かめられた．

- 各自動プログラム修正技術によってバグを修正できたプログラム群にはどのような違いがあるのかを分析した．そして，DrRepair でのみ修正できたプログラム群には，学習データセットを生成する際に埋め込むバグの内容が反映されていることが考えられた．

2. 背景

2.1 自動プログラム修正

自動プログラム修正とは，主に機械学習技術を利用し，人の手によらず自動的にプログラム中のバグを修正することを目指した研究分野である．

デバッグにかかるコストを軽減するため，自動プログラム修正に関する研究は近年盛んに行われている．自動プログラム修正の中には，論理エラーを修正するための試みと，文法エラーを修正するための試みがある．論理エラーとは，間違った分岐条件などにより想定と異なる動作をしてしまうエラーである．また，文法エラーとは，変数宣言の欠如のような，プログラミング言語の持つ文法規則に従っていないことによるエラーである．

本研究では，文法エラーの修正に着目する．文法規則のようなプログラミング言語特有の性質は，特にプログラミング初学者にとっては，誤りやすく訂正しにくいものである [5]．そのため，文法エラーを自動的に修正することにより，デバッグにかかるコストをより大きく軽減することができると考えられる．

本研究では，文法エラーを修正対象とした自動プログラム修正技術を 2 つ比較した．以下でそれらを紹介する．

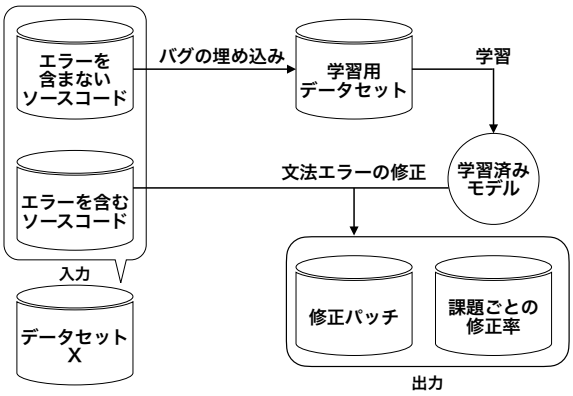

図 1 DeepFix がバグを修正する過程

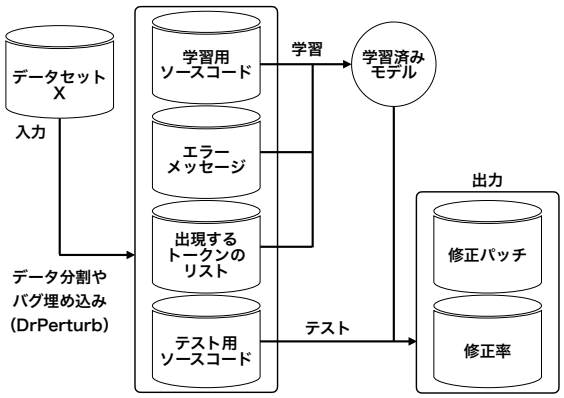

図 2 DrRepair がバグを修正する過程

DeepFix． DeepFix は，2017 年に Gupta らによって提案された, C 言語の文法エラーを修正することに特化した技術である [8]．その動作の大まかな流れを**図 1** に示す．DeepFix は，C 言語で書かれたソースコードにバグを埋め込み，それらをトークン列に変換し，Seq2Seq モデル [11] の学習に利用する．そのため，DeepFix の実行には，修正の対象となるソースコードだけでなく，学習用データセットを生成するためのエラーのないソースコードも必要となる．そして，エラーを含むソースコードに対して正しいトークン列を推論することで，文法エラーの修正を行う．

DeepFix は，「多くの場合，プログラムのエラーは，エンジニアの不注意や言語に対する知識不足などによって生じる文法エラーである」という構想のもとで開発されたものであるため，論理エラーは修正の対象としていない．しかし，文法エラーに関していえば，提案論文中の実験において全体の 27 %のプログラムを完全に，19 %のプログラムを部分的に修正することに成功している．

DrRepair． DrRepair は，2020 年に Yasunaga らによって提案された自動プログラム修正技術である [13]．特徴としては，バグを含むプログラムとその修正結果のペアだけでなく，そのバグに関するコンパイルメッセージも学習に利用されているという点が挙げられる．その動作の大まかな流れを**図 2** に示す．

DrRepair も，入力データセット中の正しいプログラムにバグを埋め込むことで学習用のデータセットを作成する．ここには DrPerturb という手順が取り入れられている．DrPerturb では，学習に利用できるデータが限られてしまうという多くの自動プログラム修正技術が抱える問題を軽減するために，事前に学習させたバグ埋め込みモデルを利用する．このモデルは，初学者，経験を積んだ開発者，自動生成によってそれぞれ書かれたプログラムのバグを学習しており，それと類似した種類および頻度のバグを埋め込む．そして，バグを埋め込んだプログラムをコンパイルした際のメッセージと，そのプログラム中に出現するトークンをまとめたファイルも生成し，修正に利用する．

DrRepair はソースコード中に出現するシンボル（変数や型名など）とコンパイルメッセージ中に出現するシンボルの関係性を学習するため，プログラム－フィードバックグラフというものを定義し利用している．これは，ソースコードやコンパイルメッセージ中に出現するシンボルをそれぞれ頂点とし，同じものを指す頂点を辺で結んだグラフである．DrRepair の学習モデルは Seq2Seq モデルをベースとし，Graph Attention 層を通じてプログラム－フィードバックグラフを取り込んでいる．これにより，バグの修正に関係するトークンを特定しやすくなり，従来の Seq2Seq モデルや AST モデルよりも多くの情報を伝播させることが可能となっている．

2.2 関連研究

自動プログラム修正技術に関する関連研究．自動プログラム修正技術に関する研究は現在盛んに行われており，本研究で取り上げたもの以外にも様々な手法が提案されている．

DeepFix を提案した Gupta らは，その後も自動プログラム修正の研究を続けており，2019 年には RLAssist を発表している [7]．RLAssist はプログラミング初心者のサポートを目的としており，DeepFix と同様に C 言語の文法エラーを修正する．ソースコードの学習には A3C（Asynchronous Advantage Actor-Critic）による強化学習を採用しており，DeepFix よりも柔軟に修正を行うことができる．さらに，人間や別のエージェントによるデモンストレーション [2] を取り込んだ場合には，より高い性能を発揮できることも示されている．しかし，DeepFix が 1 行単位でバグを修正するのに対し，RLAssist はトークン単位で修正する．このことから，プログラミング初心者にありがちな誤りである「変数の宣言忘れ」を解決することが難しいと考え，本研究では取り扱わなかった．

DrRepair を提案した Yasunaga らも，その後も自動プログラム修正に関する研究をつづてており，2021 年には Break-It-Fix-It を提案している [14]．それまでの多くの研究では，自動プログラム修正に利用するモデルのための学習データセットは，バグのない正しいソースコードにバグを埋め込むことで修正前後のソースコードのペアを生成していた．しかし彼らは，このようにして生成されるバグと実際のバグが必ずしも近しいものであるとは限らないと考え，新しい手法として Break-It-Fix-It を提案した．Break-It-Fix-It では，まず既存の手法で事前学習したバグ修正モデルを利用してバグ埋め込みモデルのための学習データセット（バグ修正後のプログラムとバグ修正前のプログラムのペア）を生成する．そして，そのデータを学習したバグ埋め込みモデルを利用してバグ修正モデルのための学習データセット（バグ埋め込み後のプログラムとバグ埋め込み前のプログラムのペア）を生成する．このように，バグ修正モデルとバグ埋め込みモデルの学習を交互に繰り返すことで，2 つのモデルの性能をより高めることができる．その結果，DeepFix や DrRepair を上回る修正率を記録している．しかし，Break-It-Fix-It はラベルづけの難しいデータセットを利用するために設計されたものであるため，修正履歴を辿ることでラベルづけできるデータセットを利用した，本研究では扱わなかった．

自動プログラム修正技術の性能評価に関する関連研究．本研究のように既存の自動プログラム修正技術を別のデータセットを用いて実行し，その性能を評価した研究もある．

Yi らは，生徒にあわせたフィードバックの自動生成が難しいことを ITSP[*1] の課題であるとし，自動プログラム修正技術を利用してこの課題を解決することに取り組んだ [15]．この研究の予備実験として，彼らは代表的な自動プログラム修正技術として GenProg[12]，AE，Angelix，Prophet[10] の 4 つを挙げ，これらが学生によって書かれたプログラムのバグを修正できるのかを確認した．ここでは，Gupta らが DeepFix の評価に利用したもの [8] と同じデータセットが利用された．この予備実験の結果，彼らは 4 つの自動プログラム修正技術を併用し，このデータセットから取り出したエラーの 31%に対して修正パッチを出力することができたと報告している．しかし，この結果は経験を積んだ開発者によるプログラムを修正した際の結果よりも劣るものであり，その原因として学生によるプログラムがより大規模かつ複雑な修正を要求することを挙げている．彼らの研究は，テストスイートに対する論理エラーを修正することを目指したものであるという点で，文法エラーに着目した本研究とは異なっている．

2.3 研究の動機

2.1 節で紹介した自動プログラム修正技術たちの性能は，共通してインド工科大学によるデータセットを利用して評価されている．しかし，評価に利用されているのはこのデータセットのみであるため，九州大学によるデータセットを用いた追実験を行い，各技術の性能を比較し，先行研

[*1] intelligent tutoring system for programming の略．

表 1 各データセットに含まれるソースコードの件数

データセット	問題数	エラーあり	エラーなし	合計
IITK	93	6,978	46,500	53,478
ED	47	3,157	7,339	10,496
PS	57	1,551	3,834	5,385
LA	21	1,861	3,936	5,797

表 2 主要なエラーメッセージとその主な原因

ID	エラーメッセージ	主な原因
e_1	expected declaration or statement	閉じ括弧の不足
e_2	expected identifier	閉じ括弧の過多
e_3	undeclared	変数や関数の宣言がされていない
e_4	expected misc. tokens	その他のトークンや宣言の不足

究で行われた比較の一般性を高めたいと考えた．これにより，最新の技術が他のデータセットにおいてもより高い性能を発揮することや，それぞれの技術にもう一方の技術より優れた点がないのかを確認し，今後の自動プログラム修正技術の開発に向けた指針を提供できる．

そこで，次のような RQ を設定し，調査を行った．

RQ1 各自動プログラム修正技術のいずれかのみによって修正できるプログラムは存在するのか．また，存在するならばそれらに含まれるエラーはどのようなものか．

RQ2 各自動プログラム修正技術によって修正できるプログラム群にはソースコードメトリクスの観点における違いはあるのか．

3. データセット

本研究では，次に示す 4 つのデータセットを使用した．各データセットに含まれるソースコードの件数など，数値データについては**表 1** に示す．

3.1 IITK(Indian Institute of Technology Kanpur)

DeepFix を提案した論文 [8] で用いられたデータセットである．インド工科大学カーンプル校において Prutor[3] と呼ばれる個別指導システムを用いて収集されたものであり，C 言語プログラミングの入門用講義で課された 93 の課題と，それに対する学生の解答をまとめたものである．

3.2 ED(Education), PS(Pharmaceutical Science), LA(Law)

九州大学で開講された基幹教育科目の講義によるものであり，C 言語プログラム初学者向けの課題に対する解答として学生が書いたソースコードから構成されている．また，学生の所属は，ED データセットでは教育学部，PS データセットでは薬学部，LA データセットでは法学部が主となっている．他学部の学生によって書かれたソースコードも含まれていたが，そのような学生はどのデータセットにおいても学生全体の 10 %未満であり，本研究では無視できるものとした．九州大学では，TERA-TERM を通じて接続できる作業サーバーが学生に提供されている．そのサーバーには GCC がインストールされているため，学生はコンパイラをインストールすることなくコンパイル作業に取り組める．本データセットは，そのサーバーから SFTP（SSH File Transfer Protocol）を通じてコンパイル時のログを収集したものである [4]．

DeepFix を提案した論文では，データセット中に多く見られたエラーメッセージとその原因がまとめられており，それらのみでエラーメッセージの大半を成していると述べられていた．それらのエラーメッセージと，対応する原因を日本語に訳したものを，**表 2** に示す．九州大学で収集されたデータセットについてもここに記されたエラーの件数を調査したところ，**表 3** のようになった．ただし，これら 4 種類のいずれにも属さないエラーも存在していたため，エラー e_1〜e_4 の件数の合計は全体の合計と一致しない．これにより，九州大学で収集されたデータセットについても，インド工科大学で収集されたデータセットと同様に，変数宣言の欠如（e_3）や括弧の閉じ忘れ（e_1）が多いことがわかる．また，これらの他に，本来あるはずのないトークンが含まれているというエラーも多く見られた．

4. 実験と結果

手順 1. 各データセットに対して DeepFix による修正を適用した．学習データセットの生成には DeepFix の提案論文 [8] で用いられていたものと同様のバグ埋め込みを使用し，学習データセットの生成に利用したものと同じデータセットのソースコードに修正を適用した．この際，IITK データセットにおいてはトークン数が 100 以上 400 以下のもの，九州大学で収集された 3 つのデータセットにおいてはトークン数が 25 以上 400 以下のソースコードに限定して，学習データの生成に利用した．この選別は，DeepFix の提案論文で述べられていた実験の設定を踏襲したものであるが，九州大学で収集された 3 つのデータセットについてはデータの大きさを確保するために下限を調整した．

手順 2. 各データセットに対して DrRepair による修正を適用した．このとき，DrRepair では入力データセットによる学習を行う前に Codeforces[*2] で収集されたデータセットによる事前学習を行うことができるが，DeepFix と比較する際にモデルのアーキテクチャによる違いのみを分析したいと考え，事前学習は行わない設定で実行した．

手順 3. DeepFix による修正結果と DrRepair による修正結果を比較し，修正の対象となったプログラムを **(1)** DeepFix と DrRepair の両方で修正に成功したもの，**(2)**

[*2] https://codeforces.com/

表 3 各データセットに含まれるエラーの件数

データセット	e_1	e_2	e_3	e_4	その他	合計
IITK	1,061 (6.41 %)	1,659 (10.02 %)	5,468 (33.03 %)	6,236 (37.67 %)	2,132 (12.88 %)	16,556
ED	676 (7.84 %)	485 (5.62 %)	887 (10.29 %)	2,182 (25.30 %)	4,393 (50.95 %)	8,623
PS	391 (8.35 %)	176 (3.76 %)	784 (16.74 %)	947 (20.22 %)	2,386 (50.94 %)	4,684
LA	308 (7.85 %)	257 (6.55 %)	937 (23.88 %)	1,013 (25.82 %)	1,408 (35.89 %)	3,923

DeepFix のみで修正に成功したもの，**(3)** DrRepair でのみ修正に成功したもの，**(4)** DeepFix でも DrRepair でも修正することができなかったものの 4 つに分類した．さらに，(2) と (3) にはソースコードメトリクスについて有意差があるのかを，マン・ホイットニーの U 検定を利用して評価した．

4.1 (RQ1) 各自動プログラム修正技術のいずれかのみによって修正できるプログラムは存在するのか．また，存在するならばそれらに含まれるエラーはどのようなものか．

4.1.1 動機

2.1 節でも述べたように現在様々なアプローチから自動プログラム修正技術が開発されているが，複数の技術を比較するために利用されるデータセットは限られている．そのため，データセットを新たに用意して既存の自動プログラム修正技術を適用し，自動プログラム修正技術の修正結果を詳細に比較した事例を増やしたいと考えた．そこで，DeepFix による修正結果と DrRepair による修正結果を比較し，どちらか一方でしか修正できないプログラムが存在するのかを調査した．さらに，そのようなプログラムが存在する場合には，それらがどのようなバグを抱えていたのかを分析した．これにより，それぞれの技術の採用した手法の違いが，修正できるバグの種類に影響しているのかを考察する．

4.1.2 結果

実験の結果を**表 4** に示す．データセットごとに違いはあるものの，どのデータセットでも DrRepair による修正率が DeepFix による修正率を上回っており，DeepFix では修正できなかったものの DrRepair では修正できたプログラムも多く存在することが分かった．また，その数はプログラム全体の数に対して少ないものの，DeepFix でのみバグを修正できるプログラムも存在することが分かった．ただし，これは DrRepair においてモデルの事前学習を行わず，モデルのアーキテクチャのみによる修正を比較した結果であるため，事前学習を利用した場合とは結果が異なる可能性については留意すべきである．

そして，DeepFix または DrRepair のどちらか一方でのみバグを修正できたプログラムがどのようなエラーメッセージを生じていたのかを**表 5** に示す．ここで，エラーメッセージの分類については表 2 と同じものを利用した．

ソースコード 1 DrRepair でのみバグを修正できたプログラムの例

```
1     #include<stdio.h>
2     int main( int argc, const char *argv[] ){
3         int a, b, c;
4
5         a = 6;
6         b = 2;
7         c = a - b;
8 -       \ c = a * b;
9 -       \ c = a % b;
8 +       c = a * b;
9 +       c = a % b;
10        printf ( "%d|n", c );
11        return 0;
12    }
```

表 5 から，DeepFix でのみバグを修正できたプログラムと DrRepair でのみバグを修正できたプログラムでは，それらのバグによって生じるエラーメッセージ e_1〜e_4 の数について大きな特徴は見られないことがわかる．

4.1.3 考察

エラーメッセージ e_1〜e_4 に分類されないその他のエラーメッセージの内容を確認すると，「error: stray ‘X’ in program」という形のエラーメッセージは DeepFix でのみ解消できたエラーメッセージにはほとんど存在せず，DrRepair でのみ解消できたエラーメッセージに多く存在していた．このようなエラーメッセージを生じさせていたソースコードの 1 つを**ソースコード 1** に示す．修正前の**ソースコード 1** では，8 行目および 9 行目の先頭に不要なバックスラッシュがそれぞれ挿入されており，これらがコンパイラに認識されず文法エラーとなってしまう．DeepFix では，ソースコードをトークン列へと変換する際，これらのような C 言語では本来使用されない記号は捨象されるため，そもそもバグと認識されず修正パッチも出力されていなかった．DrRepair でも同様にソースコードをトークン列へと変換しているが，DrRepair ではバックスラッシュも捨象されず残されていた．さらに DrRepair では，学習データセットを生成する際，DeepFix のものよりも種類の多い記号を利用してバグを埋め込んでいた．そのため，**ソースコード 1** のようにバグを修正することができていた．

> 九州大学で収集されたデータセットにおいて，バグのあるプログラムを DeepFix と DrRepair によって修正したところ，修正率という点では DrRepair が大

表 4 各ツールによってコンパイルを通すことのできたプログラムの数とその比較

データセット	どちらも修正できた	DeepFix のみ修正できた	DrRepair のみ修正できた	どちらも修正できなかった	合計
IITK	2,138 (30.64%)	190 (2.72%)	2,676 (38.35%)	1,974 (28.29%)	6,978
ED	162 (5.13%)	26 (0.82%)	1,521 (48.18%)	1,448 (45.87%)	3,157
PS	79 (5.09%)	17 (1.10%)	715 (46.10%)	740 (47.71%)	1,551
LA	33 (2.03%)	76 (4.67%)	391 (24.00%)	1,129 (69.30%)	1,629

表 5 DeepFix または DrRepair でのみ解消できたエラーメッセージの種類とその件数

DeepFix でのみ解消できたエラーメッセージ

データセット	e_1	e_2	e_3	e_4	その他	合計
IITK	18 (4.53 %)	23 (5.79 %)	183 (46.10 %)	138 (34.76 %)	35 (8.82 %)	397
ED	2 (3.23 %)	6 (9.68 %)	6 (9.68 %)	39 (62.90 %)	9 (14.52 %)	62
PS	0 (0.00 %)	4 (9.76 %)	13 (31.71 %)	11 (26.83 %)	13 (31.71 %)	41
LA	13 (11.02 %)	24 (20.34 %)	16 (13.56 %)	45 (38.14 %)	20 (16.95 %)	118

DrRepair でのみ解消できたエラーメッセージ

データセット	e_1	e_2	e_3	e_4	その他	合計
IITK	226 (39.00 %)	610 (10.52 %)	1,988 (34.30 %)	2,445 (42.18 %)	528 (9.11 %)	5,797
ED	160 (4.48 %)	188 (5.27 %)	396 (11.09 %)	1,305 (36.55 %)	1,521 (42.61 %)	3,570
PS	65 (3.18 %)	47 (2.30 %)	405 (19.79 %)	605 (29.56 %)	925 (45.19 %)	2,047
LA	30 (3.22 %)	30 (3.22 %)	248 (26.61 %)	315 (33.80 %)	309 (33.15 %)	932

きく上回ったものの，DeepFix でのみ修正できたものと DrRepair でのみ修正できたものがともに存在した．また，DrRepair でのみ修正できたものの内容を確認したところ，ソースコードをトークン列に変換する際の手法や，学習データセットを生成する際の手法に関する，両者の違いが修正結果に現れていた．

4.2 (RQ2) 各自動プログラム修正技術によって修正できるプログラム群にはソースコードメトリクスの観点における違いはあるのか．

4.2.1 動機

修正に成功したプログラムについて自動プログラム修正技術ごとに違いがあるならば，各技術によって修正できたプログラムの持つ傾向を明らかにしたいと考えた．これにより，各技術ごとにどのようなプログラムの修正が得意なのかを知ることができれば，修正したいプログラム群の傾向によって自動プログラム修正技術を使い分けることが可能になるためである．本研究では，各技術が修正に成功したプログラム群について物理行数と循環的複雑度[*3] という 2 つのソースコードメトリクスを測定し，技術間で有意差があるのかを調査した．ここで，物理行数はプログラムの規模を表すソースコードメトリクスとして，循環的複雑度はプログラムの複雑さを表すソースコードメトリクスとして，それぞれ採用した．

表 6 DeepFix でのみ修正できたプログラム群と DrRepair でのみ修正できたプログラム群のソースコードメトリクスに有意差があるのかをデータセットごとに U 検定で評価した結果

物理行数

データセット	p 値	有意水準	帰無仮説
IITK	$8.43 * 10^{-45}$	$\alpha/4 = 0.0125$	棄却される
LA	$2.11 * 10^{-20}$	$\alpha/3 = 0.0167$	棄却される
ED	$2.46 * 10^{-2}$	$\alpha/2 = 0.025$	棄却される
PS	$2.10 * 10^{-1}$	$\alpha/1 = 0.05$	棄却されない

循環的複雑度

データセット	p 値	有意水準	帰無仮説
LA	$3.36 * 10^{-28}$	$\alpha/4 = 0.0125$	棄却される
PS	$8.09 * 10^{-3}$	$\alpha/3 = 0.0167$	棄却される
ED	$8.31 * 10^{-2}$	$\alpha/2 = 0.025$	棄却されない
IITK	$2.66 * 10^{-1}$	$\alpha/1 = 0.05$	保留される

4.2.2 結果

DeepFix でのみバグを修正することができたプログラム群と，DrRepair でのみバグを修正することができたプログラム群を，物理行数および循環的複雑度の分布から比較した結果を**図 3**，**図 4** にそれぞれ示す．また，この比較で見られた差が有意なものであるのかをマン・ホイットニーの U 検定を利用して評価した結果を**表 6** に示す．ただし，有意水準を $\alpha = 0.05$，帰無仮説を「それぞれのデータセットにおいて DeepFix でのみ修正できたプログラム群と DrRepair でのみ修正できたプログラム群では物理行数または循環的複雑度に有意な差はない」と設定した．さらに，検出力を落とすことなく検定の多重性による問題を

[*3] プログラムの動作を制御フローグラフで表し，閉じたループの個数に 1 を加えることで算出できる．

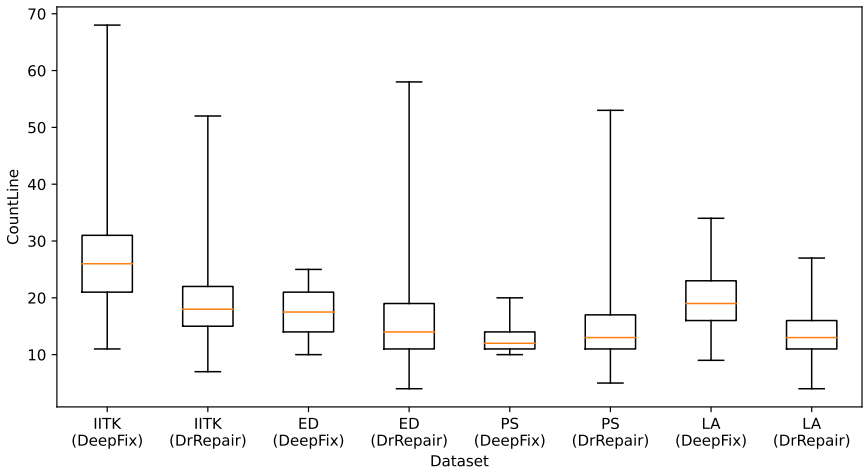

図 3 DeepFix でのみ修正できたプログラムと DrRepair でのみ修正できたプログラムにおける物理行数の比較

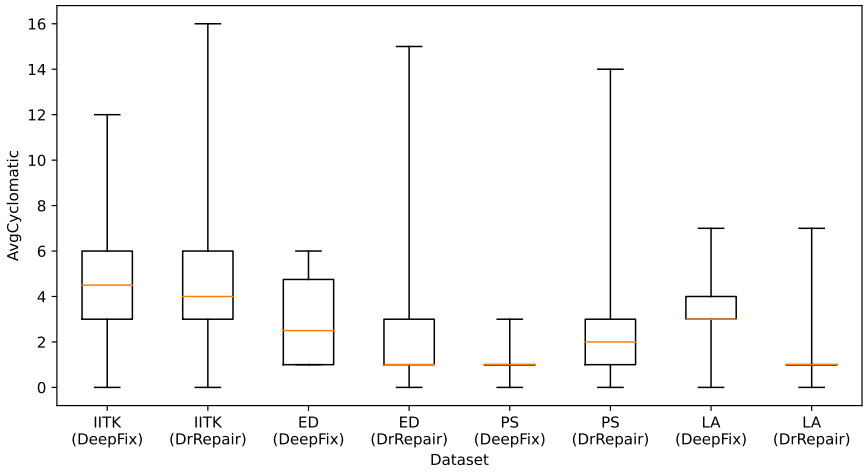

図 4 DeepFix でのみ修正できたプログラムと DrRepair でのみ修正できたプログラムにおける循環的複雑度の比較

解消するため，有意水準にはホルム法[*4] による補正をかけた．これらの図や表から，データセットごとに，DeepFix でのみバグを修正できたプログラム群と DrRepair でのみバグを修正できたプログラム群の物理行数や循環的複雑度を比較しても，大きな差があるとは言い難いことがわかる．

4.2.3 考察

九州大学で収集されたデータセットである ED，PS，LA データセットの 3 つにおいて，DeepFix でのみバグを修正できたプログラム群と DrRepair でのみバグを修正できたプログラム群のソースコードメトリクスを比較しても，結果はデータセットごとに異なっていた．このことから，この 2 つのプログラム群においてソースコードメトリクスに関する明確な差はなく，プログラムのソースコードメトリクスによって利用すべき自動プログラム修正技術を選定することは難しいことがわかる．

また，IITK データセットにおいてもソースコードメトリクスについて大きな差は見られなかったため，この結果が九州大学によるデータセットを利用したことによるものではないと考えられる．

> DeepFix でのみバグを修正できたプログラム群と DrRepair でのみバグを修正できたプログラム群において，ソースコードメトリクスに関して十分に大きな差はなかった．よって，バグを修正したいプログラムのソースコードメトリクスによって自動プログラム修正技術を使い分けることは難しいと考えられる．

5. 妥当性に対する脅威

内的妥当性． 本研究で利用した ED，PS および LA という 3 つのデータセットは，九州大学の学生がソースコー

[*4] 複数の帰無仮説を検定する際に，正しい帰無仮説が誤って棄却される可能性を抑えるための方法．N 個の帰無仮説を p 値の小さい順に並べ，k 番目の帰無仮説に対しては有意水準を $\alpha/(N+1-k)$ として，p 値の小さいものから順に検定を行う．棄却されない帰無仮説が見つかれば，それ以降の帰無仮説をすべて保留とする．

ドをコンパイルした際の記録を集めたものであった．そのため，学生が同じコードを複数回コンパイルしていると，同じ内容のソースコードが複数存在してしまう．それらが DeepFix や DrRepair にとって極端に修正しやすいものであったり極端に修正しにくいものであったりすると，実験結果に影響を与えている可能性がある．

外的妥当性． 本研究で利用した ED，PS および LA という 3 つのデータセットは，九州大学の学生が記述したソースコードを収集したものであった．これまで複数の自動プログラム修正技術の評価に用いられてきたデータセットは Gupta らによるもの [8] のみであったため，新しく用意したデータセットで複数の技術を評価した点は貢献であると言える．しかし，九州大学の学生の習熟度が影響していることが考えられ，どのような学生の集団に対しても必ず同様の結果が得られるとは言い切れない．

構成概念的妥当性． 本研究では DeepFix によって修正することができたプログラムと DrRepair によって修正することができたプログラムを物理行数や循環的複雑度というソースコードメトリクスにおいて比較したが，他にも差のあるソースコードメトリクスが存在している可能性がある．そのため，この 2 つの技術の差を網羅できているとは言い切れない．

6. まとめと今後の課題

本研究では，九州大学によるプログラミング初学者向けの講義の中で収集されたデータセットを利用して，2 つの自動プログラム修正技術 DeepFix と DrRepair の性能を比較した．まず，どちらか一方でのみバグを修正できたプログラム群の集合を比較したところ，修正率では DrRepair が DeepFix を大きく上回っていたものの，DeepFix でのみ修正できたものと DrRepair でのみ修正できたものがともに存在した．また，これら 2 つの集合を物理行数と循環的複雑度という 2 つのソースコードメトリクスにおいて比較したが，大きな差は見られなかった．そのため，バグを修正したいプログラムのソースコードメトリクスによって自動プログラム修正技術を使い分けることは難しいと考えられた．

今後の課題としては，より様々な自動プログラム修正技術を比較対象として取り入れていくことが挙げられる．**2.2 節**でも触れた RLAssist や Break-It-Fix-It を始め，現在様々な自動プログラム修正技術が盛んに提案されている．これらの性能を共通のデータセットで比較した事例を増やし，それぞれの技術に関する知見を広げていきたいと考える．

謝辞 本研究の一部は JSPS 科研費 JP18H04097・JP21H04877，および，JSPS・国際共同研究事業（JPJSJRP20191502）の助成を受けた．

参考文献

[1] Ahmed, U. Z., Kumar, P., Karkare, A., Kar, P. and Gulwani, S.: Compilation error repair: for the student programs, from the student programs, *Proceedings of the 40th International Conference on Software Engineering*, pp. 78–87 (2018).

[2] Argall, B. D., Chernova, S., Veloso, M. M. and Browning, B.: A survey of robot learning from demonstration, *Robotics Auton. Syst.*, Vol. Vol. 57, No. 5, pp. 469–483 (2009).

[3] Das, R., Ahmed, U. Z., Karkare, A. and Gulwani, S.: Prutor: A System for Tutoring CS1 and Collecting Student Programs for Analysis, *CoRR*, Vol. abs/1608.03828 (2016).

[4] Fu, X., Shimada, A., Ogata, H., Taniguchi, Y. and Suehiro, D.: Real-time learning analytics for C programming language courses, *Proceedings of the 7th International Learning Analytics & Knowledge Conference*, pp. 280–288 (2017).

[5] Fu, X., Yin, C., Shimada, A. and Ogata, H.: Error log analysis in c programming language courses, *Proceedings of the 23rd International Conference on Computers in Education*, pp. 641–650 (2015).

[6] Gazzola, L., Micucci, D. and Mariani, L.: Automatic Software Repair: A Survey, *IEEE Transactions on Software Engineering*, Vol. Vol. 45, No. 1, pp. 34–67 (2019).

[7] Gupta, R., Kanade, A. and Shevade, S. K.: Deep Reinforcement Learning for Syntactic Error Repair in Student Programs, *The 33rd AAAI Conference on Artificial Intelligence*, pp. 930–937 (2019).

[8] Gupta, R., Pal, S., Kanade, A. and Shevade, S. K.: DeepFix: Fixing Common C Language Errors by Deep Learning, *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, pp. 1345–1351 (2017).

[9] Hajipour, H., Bhattacharyya, A. and Fritz, M.: SampleFix: Learning to Correct Programs by Sampling Diverse Fixes, *CoRR*, Vol. abs/1906.10502 (2019).

[10] Long, F. and Rinard, M.: Automatic patch generation by learning correct code, *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pp. 298–312 (2016).

[11] Sutskever, I., Vinyals, O. and Le, Q. V.: Sequence to Sequence Learning with Neural Networks, *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems*, pp. 3104–3112 (2014).

[12] Weimer, W., Nguyen, T., Le Goues, C. and Forrest, S.: Automatically finding patches using genetic programming, *Proceedings of the 31st International Conference on Software Engineering*, pp. 364–374 (2009).

[13] Yasunaga, M. and Liang, P.: Graph-based, Self-Supervised Program Repair from Diagnostic Feedback, *Proceedings of the 37th International Conference on Machine Learning*, pp. 10799–10808 (2020).

[14] Yasunaga, M. and Liang, P.: Break-It-Fix-It: Unsupervised Learning for Program Repair, *Proceedings of the 38th International Conference on Machine Learning*, pp. 11941–11952 (2021).

[15] Yi, J., Ahmed, U. Z., Karkare, A., Tan, S. H. and Roychoudhury, A.: A feasibility study of using automated program repair for introductory programming assignments, *Proceedings of the 11th Joint Meeting on Foundations of Software Engineering*, pp. 740–751 (2017).