ミューテーション法を応用した
学習者プログラムの誤り箇所特定方法の提案

2019SE042 中島亜美　2019SE061 月原花菜　2019SE065 山本詠一朗
指導教員：蜂巣吉成

1　はじめに

大学のプログラミング演習において，学習者のプログラムがコンパイルは可能だが期待する実行結果が得られないことがある．学習者は自身で誤り箇所を見つけようとするが，エラーメッセージが出力されないので誤り箇所が分からないことがある．教員や TA に質問しようとしても学習者の数に比べて教員と TA の数が少ないので，アドバイスがもらえず，やる気が削がれてしまうことがある．

武藤らはコーディングに行き詰まりコンパイルできる状態までソースコードを書くことが困難な学習者に対し，模範解答を使用してチャットボット上で二者択一の質問を行い，どのようなコードを書くべきかを支援する方法を提案している [1]．しかし，学習者が自分自身で間違えているか判断を行うので，学習者が間違えていると認識できていない箇所については支援が十分でないという問題がある．

本研究は，コンパイル可能だが期待する実行結果が得られないプログラムを，学習者が自分で誤り箇所に気付いて修正できるような支援を行うことを目的に，実行結果から間違い箇所を絞り込む方法を提案する．間違い箇所を絞り込む方法は，学習者のよくある誤りを含んだプログラム (ミュータントと呼ぶ) を作成し，実行結果を得る．その後，学習者のプログラムの実行結果がどのミュータントの実行結果と一致するか比較し，実行結果が一致する場合ミュータントと同じ誤りをしている可能性があるとして絞り込みを行う．効率よく間違い箇所の絞り込みを行うためにテストケースの入力の値の順番を決める決定木を作成する．学習者に質問やヒントを与えるためのチャットボットを使用するための質問木を作成する．本研究ではチャットボットの使用は対象外とし，質問木の作成までを対象とする．本研究の技術的課題には学習者の誤りと一致する可能性のあるミュータントが 1 つに特定できない場合のヒントの出し方が挙げられる．一度に各ミュータントのヒントを送るかランダムに送るのかを考え，それを元に質問木を生成する必要がある．

2　背景技術・関連研究

ミューテーション解析とは，テストケースの欠陥検出能力を測定する方法のひとつである．テスト対象プログラムに故意に誤りを含ませ，テストケースによってその誤りが検出できたかどうかによって能力の測定を行う [2]．本研究では，誤りのことをミューテーション，誤りを含んだプログラムのことをミュータント，ミューテーションをどのように作成するかの規則をミューテーションオペレータとする．

武藤ら [1] は，教員や TA にすぐ質問できず学習者のやる気が削がれてしまう問題に対し，コーディングに行き詰まりコンパイルまで至っていない学習者を対象とした，チャットボットを使用して模範解答から二者択一の質問を行い，どのような文を書くべきかを支援する方法を提案している．学習者が回答しやすく効率のいい質問を考えることで，問題解決の糸口を掴みやすくし，質問効率を高めることで学習者が欲する解答に早く辿り着くことができ，学習者の学習意欲が向上するのではないかと考えている．武藤ら [1] と本研究の違いは，対象としているソースコード，間違い箇所の特定方法の 2 つである．武藤ら [1] の対象としているソースコードはコンパイルに至っていないコードだが，本研究の対象としているソースコードはコンパイルは可能だが期待される実行結果が得られないコードである．間違い箇所を特定する方法は，武藤ら [1] は模範解答を使用し選択肢を用いた質問を行うことで間違い箇所を特定していたが，この方法では学習者が間違えていないと思い込んでいる間違いを特定できないと考え，本研究では実行結果を用いて間違い箇所を特定する．

3　ミューテーション法を応用した
学習者プログラムの誤り箇所特定方法

3.1　概要

本研究では，ミューテーション法を応用した学習者プログラムの誤り箇所特定方法を提案する．ミューテーション法を応用して実行結果から誤り箇所を絞り込む理由は，学習者自身が気がついていない間違いを見つけ出すためである．図 1 は，提案する方法の流れを図にしたもので，本提案方法の工程は次のようになる．

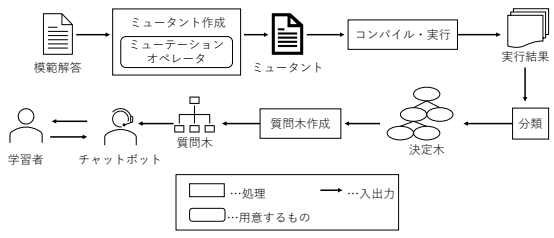

図 1　提案した方法の流れ

1. 模範解答からミュータントの作成

2. 作成したミュータントのコンパイル・実行と実行結果の保存
3. 実行結果をテストケースごとに分類し，実行結果グループの作成・学習者に質問するテストケースの順番を決めるための決定木の作成
4. 学習者に対し質問やヒントを提示するための質問木の作成
5. 質問木を元にルールベース型のチャットボットを使用して学習者にヒントを与える

本研究では 1 から 3 を自動に行うツールを作成しており，4 は作成途中である．1 で使用するミューテーションオペレータはツール上で用意してある．教員は各行に「その行がどのような動きをしているか」というコメントを記入した模範解答とテストケースを用意する．実行結果グループを作成するために出力は”&@”で区切る．Listing1 に例を示す．

Listing 1 組み合わせの数を求めるプログラム 模範解答

```
1  int comb(int a,int b){
2      int result;
3      int upper = 1; // 分子の計算用の変数宣言
4      int lower = 1; // 分母の計算用の変数宣言
5
6      if(a <= 0 || b <= 0){// 全体の個数を表す変数と取り出す個数を表す変数がどちらも0 以下の時
7          return -1;
8      }
9      if(a == b){// 全体の個数を表す変数と取り出す個数を表す変数が同値の時
10         result = 1;// 全体の個数を表す変数と取り出す個数を表す変数が同値の時の結果の代入
11         return result;
12     }
13     if(a < b){// 全体の個数を表す変数が取り出す個数を表す変数より小さい時
14         return -1;
15     }
16
17     for(int i = 0;i < b;i++){// 分子の計算
18         upper = upper * (a - i);
19     }
20     for(int i = 0;i < b;i++){// 分母の計算
21         lower = lower * (b - i);
22     }
23
24     result = upper / lower;
25     return result;
26 }
27 int main(void){
28     int case1 = comb(5,3);
29     int case2 = comb(8,2);
30     int case3 = comb(4,4);
31     int case4 = comb(2,5);
32
33     printf("%d\n",case1);
34     printf("&@\n");
35     printf("%d\n",case2);
36     printf("&@\n");
37     printf("%d\n",case3);
38     printf("&@\n");
39     printf("%d\n",case4);
40
41     return 0;
42 }
```

3.2 前提条件

本研究の対象ソースコードは，コンパイルは可能だが期待される実行結果が得られないコードとし，対象言語は C 言語とする．対象とするミュータントは，1 つのミューテーションしか存在しないミュータントとする．ミューテーションに書き換える箇所は，実際に学習者が書く箇所のみとし，ミューテーションオペレータは学習者が行いそうな間違いとする．本研究では，main 関数の誤りは対象外とする．また模範解答を用意する際に，各行にその行が何をしている部分かをコメントで残すこと，実行結果を出力するプログラムは Listing1 の main 関数のように書くものとする．

3.3 ミューテーションオペレータ

ミューテーションオペレータは参考文献 [2] をもとに設定する．参考文献 [2] では，関係演算子が違う，変数に代入する値が違う，初期値の宣言忘れなどが提案されている．我々自身の経験から関係演算子間違いと初期値の値間違いに着目した．この間違いに着目した理由は，自身が授業でプログラムを書いていた際よくあった間違いだからである．本研究のミューテーションオペレータを以下に示す．

- [==] が [<=], [>=], [<], [>], [! =]
- [<=] が [==], [>=], [<], [>], [! =]
- [>=] が [==], [<=], [<], [>], [! =]
- [<] が [==], [>], [<=], [>=], [! =]
- [>] が [==], [<], [<=], [>=], [! =]
- [! =] が [==], [<], [>], [<=], [>=]
- =0 が=1,=1 が=0

箇条書きの 7 つ目は，Listing1 の 3 行目の変数に初期値など値を代入する際の誤りを想定している．

3.4 ミュータント作成

教員が用意した模範解答プログラムから 3.3 節に基づいてミュータントを作成する．Listing1 を用いて作成されるミュータントの例に，3 行目の=1 が=0 や 9 行目の == が <=, >=, <, >, ! = が挙げられる．Listing1 では 35 種類のミュータントが作成される．

3.5 プログラムの実行・無限ループと等価ミュータントへの対応

ミュータント作成ツールで作成されたミュータントを自動でコンパイル・実行し，実行結果をテキストファイルに出力する．ミュータントが無限ループであった場合には，タイムアウトを使って判断する．ミュータントの実行結果と模範解答の実行結果が一致した場合，等価ミュータントと判断する．

3.6 決定木作成

決定木は，学習者に質問するテストケースの順番を決めるために作成する．決定木のノードはミュータントの集合，枝はテストケースと実行結果の値とする．

貪欲法によるアルゴリズム 1 で決定木を作成する．最初は makeNode(root) となる．アルゴリズム内の divide(M, t) は，ミュータントの集合 M をテストケース t の結果で同値分割する関数で，実行結果とミュータントの集合の組の集合を返す．この方法でテストケースの順番を決める理由は，testcase1 から順番に質問していった場合，決定木の高さが高くなり辿り着きたい誤りまでに無駄な質問をする可能性があるためである．

アルゴリズム 1 決定木作成アルゴリズム

```
1: function MAKENODE(n)
2:     M = n のミュータントの集合
3:     if M の要素が 1 つ then
4:         return
5:     end if
6:     root から n に至る枝に出現しないテストケースの集合を T とする
7:     X = 空集合
8:     for t in T do
9:         X にdivide(M, t) を追加
10:    end for
11:    D = X の中で最も多く組ができた集合
12:    if D の組の集合が 1 つでない then
13:        for d in D do
14:            d のミュータントの集合を持つノード c を作る
15:            n から c に d の出力をラベルにした枝を作る
16:            makeNode(c)
17:        end for
18:    end if
19: end function
```

Listing1 の学習者に質問するテストケースの順番を決める決定木の例を図 2 に示す．

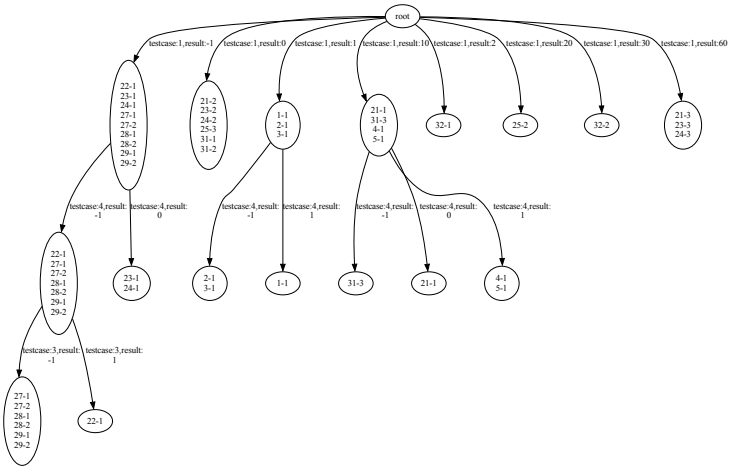

図 2 決定木の例

実行結果グループの数はそれぞれ testcase1 は 8，testcase2 は 7，testcase3 は 3，testcase4 は 3 で実行結果グループが一番多いのが testcase1 であるので，testcase1 から決定木の枝になる．次に一番実行結果グループを分割できるのが testcase4 であるので，testcase4 が枝になり，次に一番実行結果グループを分割できるのが testcase3 であるので，testcase3 が枝になる．testcase2 は 7 つのグループに分かれるが，testcase1 のグループと包含関係があり，ミュータントがこれ以上分割できないので枝にしない．

3.7 質問木作成

質問は「以下の値を入力として実行してください」とテストケースの入力の値を送ることとし，選択肢は実行結果とする．質問木のノードには質問，エッジには選択肢をラベルとしてつける．決定木の葉ノードに該当する部分までは決定木と同じように作成していき，葉ノードに達したときにヒントの設定を行う．ヒントは原則として，学習者に考えさせる内容とし直接的な内容は避けるよう設定する．用意するヒントは 2 つで，書き換えられた後の演算子について，書き換えられた箇所のコード内での役割についてとする．ヒントを二段階に分けた理由は，まず学習者にどの演算子に着目すれば良いかのヒントを与えることで，該当演算子が含まれている文がどのように動いているのかを考えさせるようにし，それでもわからない場合さらに確認する箇所を限定させることで，放置しないようにするためである．

Listing1 の 9 行目が == が ! = に書き換えられていたとする．最初に「! = でない箇所が ! = になっています」というヒントを出す．その後学習者に解決したか尋ねる質問し，解決していない場合コメントから「全体の個数を表す変数と，取り出す個数を表す変数が同値の時の対応をする箇所に間違いがあります」とヒントを出す．このヒントを出しても学習者が理解できない，または誤りが訂正できなかった場合は，教員に相談するよう促す文を提示する．ま

た技術的課題として挙げたヒントの出し方について，1 段階目のヒントは一括表示し，2 段階目のヒントは書き換え箇所が同じミュータントが多い順にヒントを出力することとする．2 段階目のヒントの順序の決め方は，葉ノードにあるミュータントを書き換え箇所ごとにグループ分けを行い，分類されたグループのうち最も多くのミュータントが含まれているグループのヒントから出力する．

図 3 に質問木の例を示す．

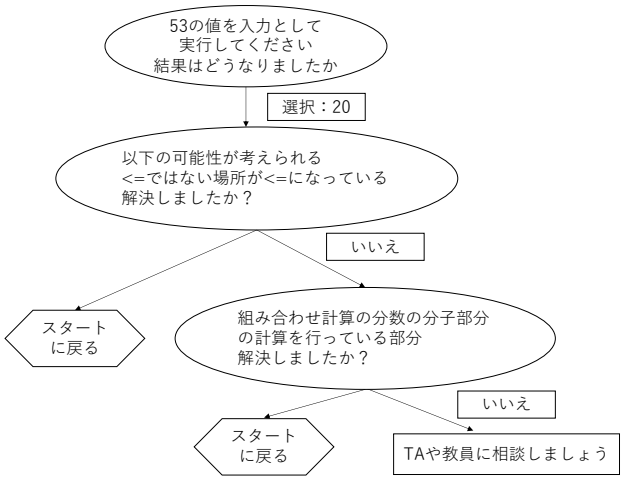

図 3 質問木の例 (一部)

4 評価・考察

4.1 間違い箇所絞り込みの評価

模範解答と違う書き方をされたソースコードの誤り箇所を特定できるか評価を行った．Listing1 と分母の計算方法が異なるソースコードを用意し，誤りを埋め込んだプログラムを 5 種類用意した．図 2 の決定木を用いると，5 種類中 3 種類は誤り箇所・ミューテーションの種類ともに一致し，1 種類は誤り箇所は一致したがミューテーションの種類が一致せず，1 種類は誤り箇所・ミューテーションの種類ともに一致しなかった．別解であっても，用意された模範解答と同じ動きをしている部分の誤りであれば，誤り箇所やミューテーションの種類の特定をすることは可能だが，模範解答にない動きをしている箇所の特定は難しいことがわかった．

4.2 決定木の比較

図 2 では, 各枝の葉に到達するまでに平均で約 1.917 個の実行結果が必要である．testcase1-testcase2-testcase3…という順番で作成される決定木の場合は，平均で約 2.692 個必要になる．testcase1 から順番に決定木を作るよりも，実行結果グループが一番分かれるテストケースの入力の値から作る方が高さが低くなることがわかった．以上のことから作成した決定木は，順番に testcase1 から質問するより効率的であり，妥当であると考えられる．

4.3 提示するヒントの改善

現在複数のミュータントが同一の実行結果グループ内に存在した場合，書き換え箇所が同じミュータントの多い順にヒントを出力する．しかし，この方法で出力するヒントが学習者が必要としているヒントとは限らない．対処法として，ツールを実行し作成したミュータントの中で学習者が行いやすい誤りを順位付けし，その順番でヒントを提示するように改善する方法がある．順位付けを行うために，教員等の経験則以外に実際にどの誤りが多いかデータを集める必要がある．

ミューテーションが 2 つ以上に増えた場合，現在のヒントの提示方法では学習者が必要とするヒントを提示するまでに時間がかかってしまう問題が生じる．ヒントの出し方を工夫し，はやく学習者が必要としているヒントを提示できるようにする．方法として，現在設定している 1 段階目のヒントを廃止し，2 段階目のヒントの精度を上げる．1 段階目のヒントを廃止する理由は，ミューテーションが 2 個になると演算子で特定することは難しいと考えられ，ミューテーション 1 個の場合と 2 個の場合で実行結果が同じ場合学習者が混乱する可能性があるためである．2 段階目のヒントの精度を上げる方法として，ヒントに対し優先順位をつける方法がある．教員が模範解答の中で誤りやすい箇所の上位数箇所を設定し，それらの書き換え箇所が出現するミュータントのヒントの優先度を高くする．これにより，できるだけ早く学習者が必要とするヒントが提示できるようになると考える．

5 おわりに

本研究では, 学習者が気付かない誤りを見つけ出し, 自身で解決できるようなヒントを与えることを目的に，ミューテーション法を応用して実行結果から間違い箇所を特定する方法を提案した．テスト実行の結果でミュータントをグループ化して，決定木を作る方法を提案し，その有効性を確認した．

今後の課題として，想定されるミューテーションの再度検討，2 つ以上のミューテーションが存在する際の対応，複数のミュータントが同一の実行結果グループに存在する場合の提示するヒントの出す順番を決めるために，学習者が行いやすい誤りの調査などが挙げられる．実際のプログラム演習で提案方法を用いたチャットボットを利用して評価することも必要である．

参考文献

[1] 武藤健太，西山雄也：“チャットボットを利用したプログラミング学習者に対するコーディング支援方法の提案”，南山大学理工学部 2021 年度卒業論文 (2022)

[2] Agrawal, H., DeMilo, R.A. et al.: Design of Mutant Operators for the C Programming Language, Technical Report SERC-TR41-P (1989)