プログラムの構造に着目したソースコードのクラスタリングによる論理エラーの推定手法

A20-1420　原田裕太

1.　はじめに

昨今の大学等の多くの教育機関では，プログラミング演習授業が広く実施され，支援ニーズが高まっている．プログラミング演習において的確な指導を行うためには，学習者の行き詰まりなどの状況を把握することが重要である．しかし，大学におけるプログラミング演習授業では，一般的に多数の学習者に対して少数の教授者や TA（Teaching Assistant）が対応する必要があるため，学習者一人一人に対して学習支援をすることは困難である．プログラミング演習において，学習者が直面する問題は，文法エラーと，コンパイルエラーとして表出しない論理エラーに分類することができる．文法エラーは，コンパイルエラーとして表出されるため，学習者自身の試行錯誤だけでの解決をある程度期待することができる．その一方で，論理エラーは，自身の力だけでその原因を探して修正をする必要があるため解決することが他のエラーと比べて困難である．同様に，教授者においても，論理エラーを解決することは容易ではなく，解決のための模索にある程度の時間や手間を要してしまう．

これに対して，プログラミング演習における学習状況把握支援を提案する研究[1]やプログラミング演習者の行き詰まり自動検出に関する研究[2]などが報告されている．また，論理エラーを対象とした行き詰まりとその箇所を推定する手法[3]が報告されているが，現状では対応可能な行き詰まりが限定的である．

本研究では，論理エラーを起こしている学習者を対象に，現在着手しているソースコードが起こしている論理エラーを推定する手法を開発する．これにより，教授者によるプログラミング演習中の行き詰まりの把握に対する新たな支援の可能性を示す．

2.　問題点と本研究の目的

2.1　支援対象

プログラミング演習において的確な指導を行うためには，個々の学習者の状況を把握することが重要であるが，躓きは多種多様である．ここで，プログラミング演習における学習段階として，(1)全くの初心者で，文法などプログラミングの基礎を学ぶ段階，(2)プログラミングの基礎は習得しており，個別の命令文だけは記述できる段階，(3)自ら簡単なプログラムを作成できる段階，の 3 つが考えられる．

本研究では，プログラムを形作ることに苦慮する(2)の学習段階の学習者を対象として,特に把握が困難である論理エラーでの躓き把握の支援を目指す．

2.2　関連研究

これまでにも，学習者の学習状況把握や，学習者の躓き推定に関する研究は数多く行われている．

まず，演習における学習者の行動を可視化することで学習状況把握を支援する研究[1]が報告されている.これは学習者によるプログラム作成の過程における作業履歴や，課題の進度を教授者に提供する取り組みである．しかし，コンパイルエラーの分析による状況提示が大きいため，ロジック構成面での行き詰まりの把握には対応していない．

また，プログラミング演習者の行き詰まり自動検出に関する研究が報告されている．浦上ほか[2]の研究では学習者が最終的に完成させたソースコードを正解とし，ある地点での実行ソースコードを比較し同一である行数を数えることで，学習者の躓きを検出した．この研究では，教授者が授業後に学習者の躓き動向を観察することを目的としており，演習中の躓き検出は考慮されていない．

さらに，ロジック構成面でのエラーを対象とした行き詰まりとその箇所を推定する手法が報告されている．川崎ほか[3]の研究ではある地点での実行ソースコードと事前に用意した正解ソースコードを比較し，一定以上類似度が上昇していない場合にロジック構成面での躓きと推定した．教授者が授業中に躓きの検出をすることを目的としているが，課題内容によってはその解法（正解ソースコード）は一つとは限らず，対応可能な行き詰まりが限定的であると考えられる．

2.3　本研究の目的

関連研究を踏まえ，本研究では，プログラミング演習における問題点のうち，以下の 3 つに焦点を当てる．

問題点 1：少数の教授者が多数の学習者それぞれの行き詰まりを把握することには限界がある．

問題点 2：教授者は演習中，コンパイルエラーとして表出しない行き詰まりを把握することが難しい．

問題点 3：解法が複数存在する課題は，解法ごとに行き詰まり方が異なるため把握することがより難しい．

上記の問題点の解決を目的とし，論理エラーを起こしている学習者を対象に，現在着手しているソースコードが起こしている論理エラーを推定する手法を開発する．本研究では特に，プログラムの構造に着目して，ソースコードの類似度をもとにしたクラスタリングを行うことにより，目的の達成を目指す．

3.　論理エラー推定手法

3.1　研究対象

東京学芸大学で実施されている授業「プログラミング演習 I」では，過去の学習者のソースコードの履歴が蓄積されている．今回の実験では，2019 年から 2022 年までの履歴を使用して，以下 2 つの課題を対象に論理エラーの推定を行う．

課題 35…2 つの整数（m, n）を入力として， m から n までの総和を求める再帰関数 sum をもとに結果を出力せよ（ただし，最初の入力 m が n より小さいと仮定する）

課題 43…1 つの整数を入力として，その値の各桁を以下のように出力せよ．また再帰関数を用いること．
（入出力例）　入力：123　出力：321123

3.2　手法の概要

手法の概要を**図 1** に示す．本研究では，演習中に論理エラーを起こしている学習者を対象に，現在着手しているソースコードが起こしている論理エラーを推定する手法を開発する（**問題点 2** に対応）．

また，解法が複数存在する課題に対しても論理エラーの推定を達成させるために，機械学習手法の一つであるクラスタリングを活用する．具体的には，過去の学習者たちの論理エラーを含むソースコードをクラスタリングさせて，論理エラーごとにデータセット（実際の代表ソースコードとそのコードの論理エラー）を作成する．その後，躓いている学習者のコードと用意したデータセットのコードを比較して，最も類似度の高かったデータセットに結び付けられている論理エラーを学習者が起こしていると推定する（**問題点 3** に対応）．

これらの手法を用いて，学習者の論理エラーの推定結果を教授者に提示する（**問題点 1** に対応）．

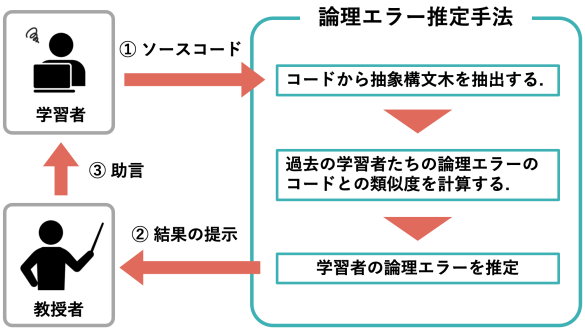

図 1　手法の概要

3.3 データセットの作成

本手法の開発には，過去の学習者たちの論理エラーを含むソースコードをクラスタリングして，論理エラーごとにラベル付けする必要がある．そのためはまず，ソースコードを LLVM（Low Level Virtual Machine）でコンパイルして抽象構文木を抽出する．次に，抽象構文木のノードをもとに，リテラル部と重みからなるトークン列の生成を行う．その後，Torres ほか[4]の提唱する Kast1 spectrum kernel を用いてソースコード間の類似度を算出する．最後に，算出した類似度から SciPy の linkage 関数を用いたクラスタリングによりラベル付けを行う．デンドログラムの一例を**図 2** に示す．

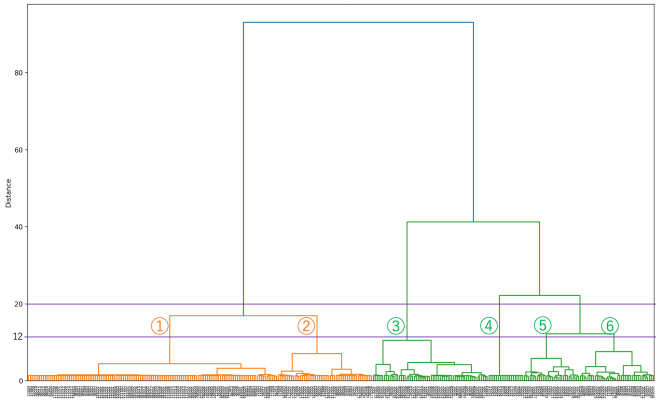

図 2　課題 35 のデンドログラム

図 2 は課題 35 のデンドログラムである．デンドログラムは，縦軸を閾値としており，閾値によってクラスタ数は変動する．例えば，**図 2** において，閾値を 20 とするとクラスタ数は 4 となる．本研究ではデータセット作成にあたってどの閾値が最適かを判断するため様々な閾値での分析を行なった．

表 1 のように，閾値 20 では，大まかな解法ごとにクラスタが分けられており，さらに閾値 12 では，使用する文法や細かな記述の違いが見られ，おおよそクラスタごとにまとまっていることが確認された．これらのことから，クラスタリングによる論理エラーの分類はある程度妥当であると考えられる．

クラスタ①②は想定される解法に沿った論理エラーであり，クラスタ③④は想定外の解法による論理エラーである．このように，教授者の想定する解法から大きく逸れてしまっている場合の論理エラーも抽出することができた．

表 1 では，作成した論理エラーと，その論理エラーを含むソースコード一つを選んで代表ソースコードとして，データセットとした．

表 1　課題 35 の論理エラー

クラスタ	閾値 20	閾値 12
クラスタ①	“m から n-1 の総和” + n のように，漸化式で解いている．	再帰終了のための if 文の条件部が間違っている．
クラスタ②		再帰終了のための if 文そのものが間違っている．
クラスタ③	sum(n) - sum(m-1) のように，差をとって解いている．	1~n の総和を求める関数を別に作成して，sum 関数で用いている．
クラスタ④		1~n の総和を求める関数を作成して，main 関数で用いている．
クラスタ⑤	漸化式が main 関数内にあり，再帰が上手くいっていない．	
クラスタ⑥	再帰関数内の処理が全体的に間違っている．	

3.4 論理エラーの推定

論理エラーを推定したいソースコードを入力とし，**3.2** で作成したデータセットの各クラスタの代表ソースコードとの類似度を求めて，一番類似度の高かったものを論理エラーとして出力する．類似度の計算は，**3.3** で行った，データセットの作成と同じ手順を用いる．

4. 実験と考察

4.1 実験概要

論理エラー推定の有用性の検証実験を行った.検証手法は様々なものが存在するが，過学習を回避するため，またデータのサンプルが多いことから，今回はホールドアウト法を用いて評価を行うこととした．

データのサンプルは，課題 35 は学生数 94 人，ソースコード 357 個，また，課題 43 は学生数 112 人，ソースコード 654 個である．このうち，データセットの作成のための訓練用データと検証用データを 9 : 1 の割合で分けた．

その後，学習者の実際の論理エラーと今回の手法で推定された論理エラーを比較し，適合率，再現率，F 値を算出した．また今回は多クラス分類であるため，マクロ平均を取った上で，それらを結果とした．

4.2 結果と考察

実験では**表 2** の結果が得られた．課題 43 については良好な結果が得られたが，課題 35 はいずれもやや低い結果となった．前提として，課題 35 と課題 43 にはデータのサンプル数に大きな差があり，それが今回の精度に大きな影響を及ぼしたと考えられる．

また，課題 35 に関して，実際の状況を精査したところ，クラスタ内のばらつきが最も大きい思われるクラスタ⑥のデータセットに分類された検証用データが一つも無いことが分かった．一方で，課題 43 は課題 35 と比較して，ばらつきの大きなクラスタは無かった．このことから，ばらつきの大きなクラスタが存在してしまうと，論理エラーの推定精度に影響を及ぼしてしまうと考えられる．

ばらつきの大きなクラスタに対する処置は改善の必要があると考える．

表 2　論理エラーの推定結果

実験課題	適合率	再現率	F 値
課題 35	0.67	0.72	0.69
課題 43	0.92	0.95	0.93

5. おわりに

本研究では，論理エラーを起こしている学習者を対象に，現在着手しているソースコードが起こしている論理エラーを推定する手法について提案した．また，提案手法を既存の検証手法に則り評価を行った結果，ある程度の精度で論理エラーの推定を行えることが示唆された．

今後は，他の課題に対しても同様に実験を行い，各課題における適切な閾値やクラスタ数を見定め，論理エラー推定のための更なる精度向上を目指す．

また，開発した提案手法を基に実際に運用を行いたい．

参考文献

[1] 市村哲，梶並知記，平野洋行，“プログラミング演習授業における学習状況把握支援の試み”，情報処理学会論文誌，Vol.54, No.12, pp.2518-2527, 2013.

[2] 浦上理，長島和平，並木美太郎，兼宗進，長慎也，“プログミング学習者のつまずきの自動検出”,情報処理学会研究報告，研究報告コンピュータと教育，Vol.2020-CE-154, No.4, pp.1-8, 2020.

[3] 川崎満広，大波奨，大沼亮，中山祐貴，神長裕明，宮寺庸造，中村勝一，“ロジック構成躓き把握支援のためのプログラミング演習における模索痕跡分析手法”，電子情報通信学会技術研究報告，Vol.123, No.184, pp.23-28, 2023.

[4] Raul Torres et al., "Comparison of Clang Abstract Syntax Trees Using String Kernels", 2018 Int. Conf. on High Performance Computing & Simulation, pp. 106-113, 2018.