一般社団法人 電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report

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

原田 裕太† 佐藤 綜一郎† 中村 勝一†† 宮寺 庸造†

† 東京学芸大学 〒184-8510 東京都小金井市貫井北町 4-1-1

†† 福島大学 〒960-1296 福島県福島市金谷川 1 番地

E-mail: †{ a201420y@st. , m218113@st, miyadera@}u-gakugei.ac.jp, †††nakamura@sss.fukushima-u.ac.jp

あらまし 大学で行われるプログラミング演習授業において，学習者が直面する問題は，コンパイルエラーとして表出する文法エラーと，コンパイルエラーとして表出し難い論理エラーに分類することができる．特に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，解決することが困難である．さらに，学習者の取り組む課題の解法は一つとは限らず、学習者ごとに解法が異なるため，教授者が学習者の論理エラーを解決させる際には，学習者それぞれの解法を考慮した上での支援が必要とされる．本研究では，解法を考慮した上で，学習者が現在起こしている論理エラーを推定する手法を開発することを目的とする．本研究では特に，学習者の解法が表現されていると考えられる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，目的の達成を試みた．提案手法を評価実験した結果，7 割ほどの精度で論理エラーの推定を行えることが示唆された．

キーワード 論理エラー，躓き推定，プログラミング学習支援，ソースコード編集履歴

Method for Identifying Logic Errors by Clustering Source Codes with a Focus on Program Structure

Yuta HARADA† Soichiro SATO†† Shoichi NAKAMURA††† and Youzou MIYADERA††

†,††Tokyo Gakugei University 4-1-1 Nukuikitamachi, Koganei-shi, Tokyo, 184-8501 Japan

†††Fukushima University 1 Kanayagawa, Fukushima-shi, Fukushima, 960-1296 Japan

E-mail: †a201420y@st.u-gakugei.ac.jp, ††{satoyo@ , miyadera@ }u-gakugei.ac.jp,†††nakamura@sss.fukushima-u.ac.jp

Abstract In programming practice classes held at universities, the errors faced by students can be divided into grammatical errors, which are expressed as compilation errors, and logic errors, which are difficult to express as compilation errors. In particular, it is difficult to resolve logic errors because students must find the cause and correct it themselves. Furthermore, there is not only one way to solve a problem that a student is working on, and the method of algorithm coding differs for each student. Therefore, when teachers help students solve logic errors, it is necessary to support them by considering each student's method of algorithm coding. The purpose of this research is to develop a method for identifying the logic errors that learners are currently making, considering each student's method of algorithm coding. In this study, we focused in particular on program structure, which is thought to represent the student's method of algorithm coding, and attempted to achieve this goal by performing clustering based on source code similarity. The results of an evaluation experiment of the proposed method suggested that logic errors can be estimated with an accuracy of about 70%.

Keywords Logic Error, Identifying Stumbling, Programming Learning Support, Source Code Editing History

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，学習支援の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難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困難である．さらに，学習者の取り組む課題の解法は一つとは限らず、学習者ごとに解法が異なるため，教授者が学習者の論理エラーを解決させる際には，学習者それぞれの解法を考慮した上での支援が必要とされる．

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[1-2]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[3]などが報告されている．また，論理エラーを対象として行き詰まりと行き詰まり箇所を推定する手法[4]が報告されているが，現状ではプログラムの解法を考慮した推定に対応していない．さらに，学習者のソースコード編集過程を用いた学習状況を推定する研究[5-6]が報告されている．これは，ソースコードの編集履歴を入力としたときに該当する学習状況を出力する機械学習モデルを作成することによって学習状況の自動推定を実現している．しかし，トークンの書き換え情報をもとにしているため，プログラムの解法は考慮されていない．先行研究の課題を整理すると，学習者ごとに異なる解法を考慮した論理エラーに対する支援が十分ではないと言える．

そこで本研究では，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の解法を考慮した上で推定する手法の開発を目的とする．本研究では特に，解法が表現されていると考えられるプログラムの構造に着目し，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試みた．その後，提案手法をホールドアウト法によって評価することとした．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. 研究方針

2.1. 論理エラー推定手法の概要

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解法ごとに推定する手法を開発する．本研究では，プログラムの解法がプログラムの構造に現れていると仮定し，プログラムの構造に着目した分析に挑戦する．そのために，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を行い解法別に分類し，その中で論理エラーを分類することを試みる．過去の学習者たちの論理エラーを含むソースコードを分析し，解法ごと，さらに論理エラーごとの分類をあらかじめ明らかにしておくことで，新規の学習者のプログラムの構造に基づいた論理エラーの推定が実現可能となる．

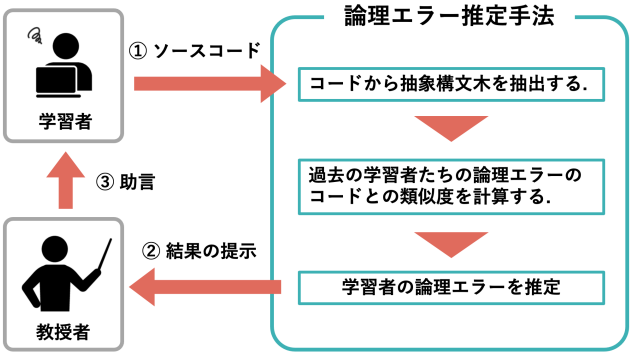

図 1 論理エラー推定手法の概要

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から抽象構文木を抽出する．次に抽象構文木をもとに Torres ほか[7]が提唱する Kast1 spectrum kernel を用いてソースコード間の類似度を算出する．その後，算出した類似度をもとに，教師なし学習のクラスタリングを行い，解法別に分類し，その中で論理エラーの分類をすることを試みる．分類した結果をもとに，解法ごと，さらに論理エラーごとにデータセット（論理エラーとその論理エラーを代表するソースコードのセット）を作成する．

最後に，過去の学習者をもとに作成した複数のデータセットと，新規の学習者が現在着手している論理エラーを含んだソースコードの類似度を計算して，最も類似度が高いとされたデータセットに結び付けられた論理エラーを学習者が起こしていると推定し，その結果を教授者に提示する．本研究では提案手法の有用性の検証実験としてホールドアウト法を採用する．

2.2. 研究手順

本研究は以下の手順で進めていくこととする．

手順 i) ソースコード間の類似度算出……第 3 章
手順 ii) データセットの作成……第 4 章
手順 iii) 論理エラー推定手法の実施……第 5 章
手順 iv) 評価と考察……第 6 章

手順 i では，過去の学習者たちの論理エラーを含むソースコードから抽象構文木を抽出し，抽象構文木をもとに Torres ほか[7]が提唱する Kast1 spectrum kernel を用いてソースコード間の類似度を算出する．この手法により，プログラムの構造に着目した解析が可能となる．

手順 ii では，算出した類似度をもとに，教師なし学習のクラスタリングによる論理エラーごとのソースコードの分類を行い，論理エラーごとにデータセット（ソースコードから選んだ代表ソースコード 1 つとその論理エラー）を作成する．

手順 iii では，過去の学習者の論理エラーを含むソー

スコードを訓練用データと検証用データの 2 つに 9:1 の割合で分け，訓練用データから手順 ii に則って複数のデータセットを作成し，検証用データを学習者が現在着手しているソースコードに見立てて，提案手法を開発し，その実施を行う．

手順 iv では，推定手法の評価と考察を行う．

3. ソースコード間の類似度算出

3.1. 概要

本章では，前章で述べた論理エラー推定手法の開発に必要な手順であるソースコードの類似度算出について述べる．論理エラーのデータセットを作成するためには，過去の学習者が起こした論理エラーを含むソースコードを，ソースコード間の類似度を算出して，教師なし学習のクラスタリングにかける必要がある．ソースコードを比較する際に，学習者によって変数名や関数名に細かな違いがあったり，意味を持たないコードがプログラム内に混在していたりすることがあるため，ある程度共通の表記に変更する必要がある．今回はコンパイラが生成する抽象構文木に変換することでコード生成に必要のない部分は削除され意味的に同様のものは共通のノードとして表現される．今回は LLVM を使用することで抽象構文木を生成する(3.2 節)．次に，リテラル部と重みからなるトークン列の生成を行う(3.3 節)．その後，Torres ら[7]が提唱する Kast1 Spectrum Kernel を用いてソースコード間の類似度を算出する(3.4 節)．

3.2. 抽象構文木の生成

ここでは，ソースコードから抽象構文木を生成する詳細について定義する．

今回は LLVM（Low Level Virtual Machine）を使用して抽象構文木を生成した．抽象構文木にはノード以外にも様々な情報が含まれているが，本研究においては Kast1 Spectrum kernel で使用するノードと木構造のみを保持し，その他は除去した（図 2）．

3.3. トークン列への変換

次に，プログラムの構造情報を維持したままクラスタリングをかけたい．そのため，抽象構文木を数値情報に変換する必要がある．今回は，Torres ほか[7]の提唱するアルゴリズムに則って，抽象構文木をトークン列に変換する．このとき，変換と同時にトークン列を圧縮することによって扱いやすい形に変換している．

抽象構文木に現れるノードを単に羅列するだけでは木構造が失われてしまうため，新たなトークン“LEVEL_UP”を導入する．これは抽象構文木を読み取る際に下から上に移動する分だけ重みを増やして追加することで，木構造を失わずにトークン列として扱えることを目的にしている．

```
TranslationUnitDecl
|-TypedefDecl
| `-BuiltinType
|-TypedefDecl
| `-BuiltinType
|-TypedefDecl
| `-RecordType
|   `-Record
|-TypedefDecl
| `-PointerType
|   `-BuiltinType
|-TypedefDecl
| `-ConstantArrayType
|   `-RecordType
|     `-Record
|-FunctionDecl
| `-CompoundStmt
|   |-CallExpr
|   | |-ImplicitCastExpr
|   | | `-DeclRefExpr
|   | `-ImplicitCastExpr
|   |   `-ImplicitCastExpr
|   |     `-StringLiteral
|   `-CallExpr
|     |-ImplicitCastExpr
|     | `-DeclRefExpr
|     `-ImplicitCastExpr
|       `-ImplicitCastExpr
|         `-StringLiteral
`-FunctionDecl
  |-ParmVarDecl
  |-BuiltinAttr
  `-FormatAttr
```

図 2 必要な情報を保持した抽象構文木

次に，生成されたトークン列は冗長なので圧縮を行う．圧縮のアルゴリズムは以下の 4 つである．

① 連続する同じリテラルを持つトークンは重みを合計する（図 3）
② キャスト式等は削除し後続の重みに加算する（図 4）
③ 宣言文と LEVEL_UP の間は削除し前者に加算する（図 5）
④ 同じリテラルペアは重みを合計する（図 6）

$$[\mathrm{BinaryOp}]_1[\mathrm{BinaryOp}]_1[\mathrm{BinaryOp}]_1 \rightarrow [\mathrm{BinaryOp}]_3$$

図 3 連続して同じリテラルを持つトークンは重みを合計する

$$[\mathrm{CStyleCastExpr}]_1[\mathrm{CallExpr}]_1[\mathrm{DeclRefExpr}]_1 \rightarrow [\mathrm{DeclRefExpr}]_3$$

図 4 キャスト式等は削除し後続の重みに加算する

$$[\mathrm{DeclStmt}]_1[\mathrm{VarDecl}]_1[\mathrm{DeclRefExpr}]_1[\mathrm{LEVEL\_UP}]_1 \rightarrow [\mathrm{DeclStmt}]_2[\mathrm{LEVEL\_UP}]_3$$

図 5 宣言文と LEVEL_UP の間は削除し前者に加算する

$$[\mathrm{IntegerLiteral}]_1[\mathrm{LEVEL\_UP}]_3[\mathrm{IntegerLiteral}]_1[\mathrm{LEVEL\_UP}]_1 \rightarrow [\mathrm{IntegerLiteral}]_2[\mathrm{LEVEL\_UP}]_4$$

図 6 同じリテラルペア重みを合計する

以上で，リテラル部と重みからなるトークン列が完成した．

3.4. 類似度算出

ここからは Torres ら[7]が提唱する Kast1 Spectrum Kernel を用いてソースコード間の類似度の算出を行う．基本的な流れは，2 つのトークン列を比較し，トークン列の最長共通部分列とその部分の重みの合計を 1 次元目の値に，1 次元目の算出に使われなかったトークン列の部分から次に長い共通部分列を見つけ，その部分の重みを 2 次元目の値に，同様に 3 次元目の値まで求める．それぞれのトークン列に 3 次元ベクトルが生成されるので，これら 2 つのベクトルの内積を 2 つのトークン列全体の重みの合計の積で割ることで，類似度を算出することができる．

4. データセットの作成

4.1. 概要

本章では，第 3 章で算出した類似度をもとにしたデータセットの作成について述べる．第 3 章までででソースコードの類似度を算出することができたので，教師なしの階層型クラスタリングを行う．次に，クラスタリング結果をもとに，プログラムの解法別に，論理エラーごとにソースコードの分類を行い，論理エラーごとにデータセット（ソースコードから選んだ代表ソースコード 1 つとその論理エラーのセット）を作成する．以下，データセット作成の過程の詳細を述べる．

4.2. クラスタリングによる論理エラーの分類

第 3 章までで，ソースコードの類似度を算出することができたので，SciPy の linkage 関数を用いて教師なしの階層型クラスタリングを行う．本研究では，階層クラスタリングによってデータを可視化するためにデンドログラムを採用する．デンドログラムは，過去の学習者の論理エラーを含むソースコードを第 3 章で求めた類似度に基づいてグループにまとめ，それらのまとまりがどの程度類似しているのかを木構造で表現している．

クラスタリングの結果を示すデンドログラムの一例として，課題 35 番の内容を**表 1** に，デンドログラムを**図 7** に示す．デンドログラムの縦軸は，クラスタ間の距離を表す閾値を示している．

デンドログラムからも分かる通り，閾値によってクラスタ数は変動する． 例えば，**図 7** のデンドログラムにおいて，閾値を 60 とするとクラスタ数は 2 となり，閾値を 30 とするとクラスタ数は 3 となる．このように閾値を設定すると，クラスタが形成される条件が変わるため，閾値を低く設定すると，より細かいクラスタが形成される．

論理エラーを分類するにあたってどの閾値が最適かを判断する必要がある．そこで本研究では，様々な閾値で検証を行なった．

表 1　課題 35 番の内容

課題	2 つの整数（m, n）を入力して，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するプログラムを作成せよ．ただし m<m と仮定してよい．

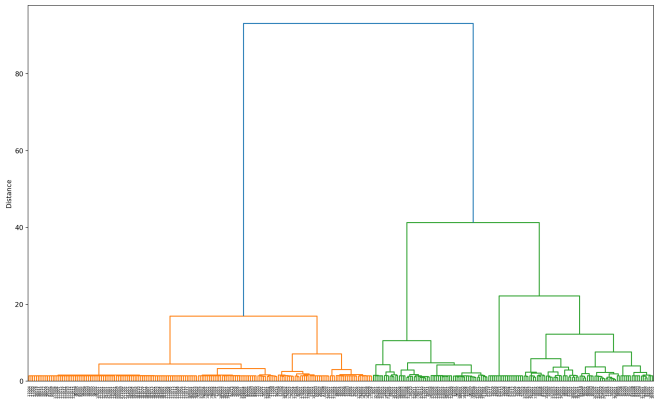

図 7　出力された課題 35 番のデンドログラム

4.3. クラスタリング結果の検証

本節では，前節までのクラスタリング結果が妥当なものであるかを検証する．検証の一例として，**図 7** でも使用した課題 35 を用いて説明する．この課題は解法が複数あることが予想され，様々な論理エラーが出現すると思われたため選出した．

検証方法として，大きな閾値から分析をはじめ，解法ごとに分類がされているかを確認し，続けて，小さな閾値では論理エラーごとに分類ができるかどうかを検証する．

まず，閾値を 20 としたとき，クラスタは 4 つに分かれている．一つ目のクラスタでは，m から n までの総和を「”m から n-1 の総和”+1」のように漸化式を用いるような解法でまとまっていた．次に，二つ目のクラスタでは，m から n までの総和を「sum(n) - sum(m-1)」のようにそれぞれの和の差をとるような解法でまとまっていた．三つ目のクラスタでは，二つ目のクラスタに似ていたが，作成した関数の数が異なる解法がまとまっていた．四つ目のクラスタでは，再帰関数内の処理が全体的に間違っている解法がまとまっていた．このことから，大きな閾値において，解法がクラスタごとにまとまっていることが確認された．

また，閾値を小さくして 12 としたとき，閾値 20 の一つ目のクラスタはさらに 2 つに分かれていた．一つ目は再帰終了のための if 文の条件部が間違っているソースコードがまとまっており，二つ目は再帰終了のための if 文の構造そのものが間違っているソースコードがまとまっていることが分かり，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．考察の詳細は第 6 章にて述べる．

これらのことから，このクラスタリングによる解法

ごとの論理エラーの分類はある程度妥当であると考えられる.

4.4. データセットの作成

前節で妥当性が示されたクラスタリング結果を用いて, 論理エラーを推定するためのデータセットの作成に関して述べる.

必要とされるデータセットは, 解法を踏まえた論理エラーとその論理エラーを代表するソースコードのセットである. 節 4.3 で行った論理エラーの分類結果をもとに, 論理エラーごとに代表ソースコードを一つずつ選出した.

5. 論理エラー推定手法の実施

5.1. 実験の概要

本章では, 前章までで作成したデータセットを用いて, 論理エラー推定手法の開発と評価を行う. 前章で, 過去の学習者の論理エラーを含んだソースコードをもとに作成した複数のデータセットと, 新規の学習者が現在着手している論理エラーを含んだソースコードの類似度を算出して, 最も類似度が高いとされ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して, その結果を教授者に提示することとした. 本研究では, 学習者の論理エラーを含むソースコードを, 解法を踏まえた論理エラーごとに分類してデータセットを作成しており, データセットは基本的に 3 種類以上存在することから開発する論理エラー推定手法は多クラス分類となる. 目標とする論理エラー推定手法が多クラス分類であることと, データのサンプルである過去の学習者の論理エラーを含んだソースコードが多いこと, また過学習を回避するために, 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. 以下, 論理エラー推定手法の開発とその評価を順に述べる.

5.2. 論理エラー推定手法の開発

本研究では, 本学で実施されているプログラミング演習授業で出題されている全 111 個の課題のうち, 解法が複数存在して解法別に何種類かの論理エラーが見込まれる 3 個の演習課題を選び, それぞれ, 論理エラー推定手法の開発を試みた. 選出した演習課題を**表 2** に示す.

さらに, 前章の**図 7** で示したデンドログラムからも分かるように, 閾値の設定によってクラスタ数が変動し, 推定精度にも大きな影響を及ぼすと考えられる. よって本研究では, 演習課題ごとに適切な閾値の値を調べる作業を行った.

なお, 本研究の論理エラー推定の開発に使用するデータのサンプルは, 先述の通り本学授業における 2019~2022 年度の受講者 162 人のソースコードから論理エラーを含んだソースコードを収集して行った. その結果を**表 3** に示す.

表 2　選出した演習課題の内容

課題番号	内容
35	2 つの整数（m, n）を入力して, 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するプログラムを作成せよ. ただし m<m と仮定してよい.
43	1 つの整数値を入力して, その値の各桁を以下のように出力するプログラムを作成せよ. このとき, 再帰関数 reverse2 を用いること. 入力: 12345 出力: 5432112345
73	1 つの整数値を入力して, その値を 10 進数の値としたとき, 2 進数に変換し結果を出力するプログラムを作成せよ. このとき, 繰り返し構文と配列を用いること.

表 3 論理エラー推定の開発のために使用した人数

課題番号	人数（人）	ソースコード（個）
35	94	357
43	112	654
73	108	934

6. 評価と考察

6.1. 概要

前節までの内容をもとに論理エラー推定実験の評価を行った. まず, 前章で選出した 3 個の課題のデータのサンプルを用いて, 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て, 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.

以下の節では, 各課題で開発した論理エラー推定手法を, クラスタリング結果の検証, 代表ソースコードの選出, 論理エラーの推定結果の順で述べる.

6.2. 課題 35 番における論理エラー推定

課題 35 番のデンドログラムを**図 8** に示す. 課題 35 における適切な閾値を調べるために検証を行った.

まず, 閾値を 20 としたとき, クラスタは 4 つに分かれている. 一つ目のクラスタでは, m から n までの総和を「”n から m-1 の総和”+1」のように漸化式を用いるような解法でまとまっていた. また, このクラスタの解法は, 教授者の想定される解法に沿った方針で解いていることが分かった. 次に, 二つ目のクラスタでは, m から n までの総和を「sum(n) – sum(m-1)」のよう

にそれぞれの和の差をとるような解法でまとまっていた．三つ目のクラスタでは，二つ目のクラスタに似ていたが，作成した関数の数が異なっていることが分かった．四つ目のクラスタでは，再帰関数内の処理が全体的に間違っているソースコードがまとまっていることが確認された．二つ目のクラスタ以降は，教授者の想定外の解法による論理エラーである．その中でも，三つ目のクラスタは，**表 2** の問題文にある ”m から n までの総和を求める再帰関数 sum2 をもとに”の部分に反したものとなっているため，そもそも今回の課題を解くのに適していない方針であることが分かる．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．

このことから，閾値を 20 とした場合は，解法ごとにある程度の分類がされていることが確認された．しかし，これでは細かい論理エラーの推定までには結びつかないため，さらに閾値を小さくすることで，より細かい分析を行うこととした．

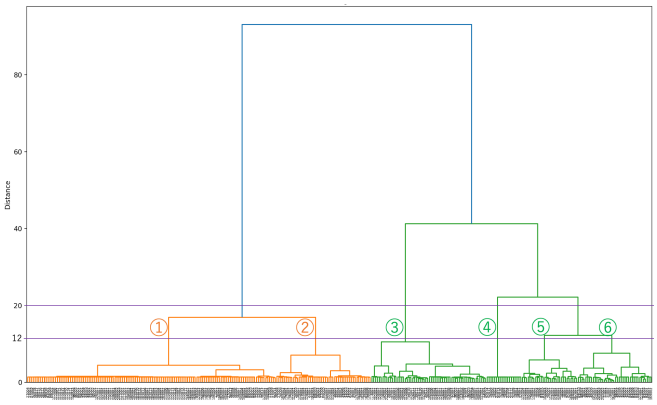

図 8　出力された課題 35 番のデンドログラム

次に閾値を 12 に設定した．この時クラスタは 6 つに分かれている．閾値 20 での一つ目のクラスタがさらに二つに分かれたことが確認でき，一つ目は再帰終了のための if 文の条件部が間違っているソースコードがまとまっており，二つ目は再帰終了のための if 文の構造そのものが間違っているソースコードがまとまっていることが確認できた．また，閾値 20 での四つ目のクラスタがさらに二つに分かれたことが確認できたが，一つ目は，再帰関数で行うはずの漸化式的な処理が main 関数内にあり，再帰がうまくいっていないようなソースコードがまとまっており， 二つ目は，再帰関数内の処理が全体的に間違っているソースコードがまとまっていることが確認された．

閾値を 12 よりも小さくすることも考慮したが，教授者に必要な論理エラーの分類は閾値 12 の段階である程度できていると判断したため，課題 35 における適切な閾値は 12 とした．

6.3. 課題 43 における論理エラー推定

課題 43 番のデンドログラムを**図 9** に示す．課題 43 における適切な閾値を調べるために検証を行った．

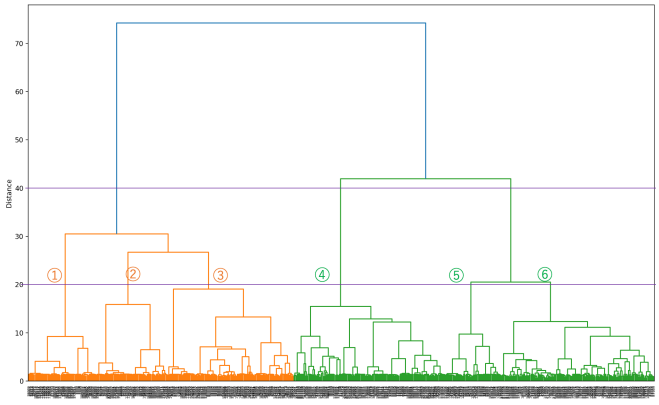

図 9　出力された課題 43 番のデンドログラム

まず，閾値を 40 としたとき，クラスタは 3 つに分かれている．一つ目のクラスタでは，再帰関数の返り値が void 型として用いられている解法でまとまっていた．また，このクラスタの解法は，教授者の想定される解法に沿った方針で解いていることが分かった．次に，二つ目のクラスタでは，再帰関数の返り値が int 型として用いられている解法でまとまっていた．三つ目のクラスタでは，再帰関数が main 関数で適切に呼び出せていないソースコードがまとまっていることが確認された．

このことから，閾値を 40 とした場合は，解法ごとにある程度の分類がされていることが確認された．しかし，これでは細かい論理エラーの推定までには結びつかないため，さらに閾値を小さくすることで，より細かい分析を行うこととした．

次に閾値を 20 に設定した．この時クラスタは 6 つに分かれている．閾値 40 での一つ目のクラスタがさらに三つに分かれたことが確認でき，それぞれ，プログラムの順序が不適切なもの，再帰終了のための if 文の条件式もしくは if 文そのものの間違いの違いによって分類されていることが分かった．また，閾値 40 での三つ目のクラスタがさらに二つに分かれたことが確認でき，一つ目は，main 関数の中身が reverse()のみのため，scanf 文が何度も呼び出されてしまうというエラーを抱え，二つ目は，main 関数の中身が，int a; reverse(a); のように scanf 文が用いられていないので適切でないことが分かった．

閾値を 20 よりも小さくすることも考慮したが，教授者に必要な論理エラーの分類は閾値 12 の段階である程度できていると判断したため，課題 43 における適切な閾値は 20 とした．

6.4. 課題 73 における論理エラー推定

課題 73 番のデンドログラムを図 10 に示す．課題 73 における適切な閾値を調べるために検証を行った．

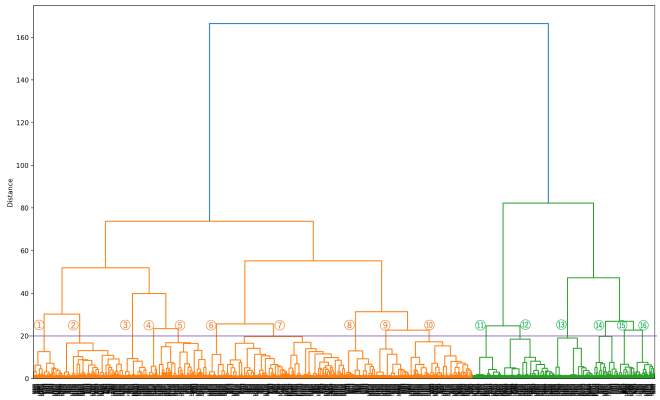

図 10　出力された課題 73 番のデンドログラム

課題 35 番と課題 43 番と同様に閾値を変更しながら分析を行ったが，課題 35 番と課題 43 番のように，解法や論理エラーに基づいたクラスタに相関が見られなかった．

閾値を 20 としたとき，クラスタは 16 個に分かれている．図 10 では閾値 20 における各クラスタに番号を振っている．クラスタごとに使用している文法やプログラムの構造が似ていることが見受けられたが，解法や論理エラーの観点でまとまっていることが確認できなかった．

そのため，ソースコードを解法ごとにも論理エラーごとにも分類することが出来ず，データセットを作成することが出来なかった．

課題 73 番は他 2 つの課題と比べてソースコード数が非常に多く，1 人に対して複数のソースコードが収集されてしまっているため， 閾値 20 において，1 人が 1 つのクラスタを形成してしまっている場合も見受けられた．

6.5. 評価と考察

評価実験では表 4 の結果が得られた．課題 73 番はデータセットを作成することが出来なかったため，評価実験を行うことは出来なかった．

表 4 論理エラーの推定結果

実験課題	適合率	再現率	F 値
35	0.67	0.72	0.69
43	0.92	0.95	0.93
73	-	-	-

評価結果から，プログラムの解法ごとに論理エラーを約 70%の割合で推定できる手法が作成可能であることが示唆された．

具体的にどの程度推定可能性があるのか考察を行う．課題 73 を除いて，最も評価結果が悪かった課題 35 において，F 値は 0.69 であった．F 値は論理エラー推定がどの程度正しいかを示す値である．そのため評価結果が悪かった課題に対しても約 70%の割合で推定できていることが分かる．従って，新たな学習者の論理エラーを含んだソースコードを入力としたときに，該当する論理エラーを 7 割程の推定が行えると考えられる．

提案手法では，プログラミング演習授業のような場合には有用性があるが，新しいプログラミング課題には対応することが出来ない．

本研究は、データセットをあらかじめ作成しておくことが条件となるため，これが本研究の限界である．また，過去のソースコードのデータから全ての解法と論理エラーが得られていることが条件となっている．そのため，推定の精度を上げるために今後は今回の手法に対して，プログラムの編集過程を考慮した分析を加えた手法の開発が望まれる．

7. おわりに

本研究では，複数の解法のある課題において，プログラムの解法ごとに論理エラーを推定して，その結果を教授者に提示することを目指した．本研究では，プログラムの解法がプログラムの構造に現れていると仮定し，プログラムの構造に着目した分析に挑戦した．そのために，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を行い解法別に分類し，その中で論理エラーを分類することを試みた．

提案手法を既存の検証手法に則り評価実験を行った結果 新たな学習者の論理エラーを含んだソースコードを入力としたときに，該当する論理エラーを概ね 7 割程の推定が行えた．そのため，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が示唆された．

今後としては，推定の精度を上げるために今回の手法に対して，プログラムの編集過程を考慮した分析を加えた手法の開発が望まれる．また，開発した提案手法を基に実際に運用を行いたい．

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] Ali Alammary, Angela Carbone, Judy Sheard, “Implementation of a smart lab for teachers of novice programmers”, ACE 2012, pp.121-130, 2012.

[3] 浦上理，長島和平，並木美太郎，兼宗進，長慎也，“プログラ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，Vol.2020-CE-154，No.4，pp.1-8，2020.

[4] 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.

[5] 石和田圭, 森本康彦, 中村勝一, 宮寺庸造, “プログラミング演習における学習状況推定のためのソースコード編集過程分析手法の開発”, 電子情報通信学会技術研究報告, vol. 116, no. 438, ET 2016-92, pp. 75-80, 2017.

[6] 川口翔大, 佐藤克己, 大沼亮, 中山祐貴, 中村勝一, 宮寺庸造,“プログラミング演習授業における AI 手法を用いた学習状況自動推定システム”, 電子情報通信学会技術研究報告, vol. 119, no. 468, ET2019-101, pp. 141-146, 2020.

[7] Raul Torres, Thomas Ludwig, Julian M. Kunkel, Manuel F. Dolz, "Comparison of Clang Abstract Syntax Trees Using String Kernels", 2018 Int. Conf. on High Performance Computing & Simulation, pp. 106-113, 2018.