オンラインジャッジシステムにおける エラーの組み合わせと解決時間の実態調査

清水 翔太[1]　槇原 絵里奈[1,a)]　吉田 則裕[1]

概要：オンラインジャッジシステム (OJS) とは，ユーザが提出したプログラムを自動でコンパイル，テスト・検証，実行し，評価結果をユーザへフィードバックするシステムを指す．既存の OJS にはプログラミングに関する大量かつ多様な問題が収録されており，ユーザはこれらの問題を解き進めることでプログラミングやアルゴリズムに関する自学自習が可能となる．OJS で生じるエラーの多くは論理エラーであることが知られている一方，ユーザごとに生じるエラーの実態調査や支援はほとんど行われていない．そこで本研究では，頻繁に生じる論理エラーの組み合わせや，単一あるいは複数のエラーが生じた際の解決にかかる時間について，CodeNet に含まれる AtCoder の提出履歴，1,519 問に対する約 1,200 万件のデータを対象に調査を行った．本調査の結果，実行時間制限を超過した際のエラー (TLE: Time Limit Exceeded) の解決割合が最も低く，また，他のエラーから TLE に遷移した場合，TLE に遷移しない場合に比べてエラー解決時間が増加する組み合わせが存在することが分かった．

キーワード：オンラインジャッジシステム，論理エラー，行動分析

An Empirical Study of Error Combinations and Solution Times in an Online Judge System

Shota Shimizu[1]　Erina Makihara[1,a)]　Norihiro Yoshida[1]

1. はじめに

ソフトウェアの需要拡大によるプログラミング教育に対する興味関心の高まり，さらに教育者の人員不足などの背景から，オンラインジャッジシステム（OJS）を用いたプログラミング教育に注目が集まっている．OJS はユーザが提出したソースコードを，多くのテストケースや検証器によって評価し，結果のフィードバックを自動で行うシステムを指す [1]．著名な OJS として，海外では Codeforces[*1]や TopCoder[*2]，日本国内では AOJ[*3]や AtCoder[*4]があげられる．これら OJS は，競技プログラミングにおけるコンテストを開催するためのプラットフォームであるだけでなく，データ構造とアルゴリズムに関する基礎的な問題や過去に出題された問題を入手するためのプラットフォームでもある．つまり，OJS は時間に縛られず多くの問題に取り組みたい学習者にとって，無料で使用可能なプラットフォームと言える．そこで本研究では，OJS を自学自習の支援環境としてプログラミング教育への導入を目指す．

ユーザは OJS 上で問題を閲覧後，個人の開発環境で題意を満たすプログラムを作成し，システム上でプログラムを提出する．提出されたプログラムは OJS 内においてコンパイル，テスト，処理時間の計算などが行われ，出題者が設定した全ての条件を満たした場合，Accepted(AC) として受領される．一方，ユーザが AC とならないプログラム（以下 Reject と呼ぶ）を OJS に提出した場合，少ないフィードバックからエラーの原因を特定し，修正する必要がある．Reject 時の例として，例えば文法エラーの場合 Compile

[1] 立命館大学
Ritsumeikan University, Ibaraki, Osaka 567–8570, Japan
[a)] makihara@fc.ritsumei.ac.jp
[*1] https://codeforces.com
[*2] https://www.topcoder.com
[*3] https://onlinejudge.u-aizu.ac.jp/home
[*4] https://atcoder.jp/?lang=ja

Error(CE)，テストケースを一部満たしていないと Wrong Answer(WA)，実行時エラーである Runtime Error(RE)，実行時間を超過した場合 Time Limit Exceeded(TLE) などのフィードバックが返ってくる．

文法エラーの場合，ユーザは個人の開発環境においてエラーメッセージを閲覧可能だが，WA や TLE のような論理エラーの場合，テストケースが公開されていない点や実行環境が異なる点から，ユーザはエラー原因の特定が困難であると考えられる．エラーの解決に時間がかかるとユーザのモチベーション低下やドロップアウトに繋がる恐れがある．実際，OJS でフィードバックされるエラーの種類はほとんどが論理エラーであり，3 章にて述べる CodeNet の AOJ のデータを対象に調査したところ，AOJ を 1 年以上継続したユーザは 15 %程度であった．そこで，本研究では Reject の過半数を占める論理エラーを中心に特徴を調査し，Reject 時におけるフィードバックの拡充やエラー解決のための支援を目指す．具体的には，IBM が提供する CodeNet[*5]に含まれる AtCoder の提出履歴を分析する．CodeNet のデータから，各問題をユーザ ID，提出日時でソートし，対応する結果を元に，下記の RQ に従いエラーの分析を行った．

RQ1 Reject にはどのようなエラーが含まれるか
RQ2 各種エラーはどれくらい解決可能か
RQ3 各種エラーから解決するためにはどれぐらいの時間がかかるか

RQ を調査することで，複数種類のエラーが生じた際のエラーの解決割合や解決時間の違いを明らかにする．本調査の結果は，エラー解決にかかる時間の見積もりや予測，OJS ユーザのドロップアウトの防止に活用できると考える．

本論文の構成は次の通りである．まず 2 章で本研究の対処となる OJS を，既存研究を交えて説明する．次に 3 章において，分析対象とした CodeNet の詳細やデータの整形内容，各 RQ を説明する．さらに RQ に基づいた分析結果および結果に対する考察を 4 章で述べ，最後に 5 章でまとめを述べる．

2. オンラインジャッジシステム (OJS)

OJS にはプログラミングコンテストの過去問やプログラミングの基礎的な問題など，多くの問題が収録されており，ユーザはプログラミングコンテストに参加しない場合でも，自由に問題に着手することができる．Wasik ら [2] によると，OJS の目的はユーザが提出したソースコードの安全性，信頼性を継続的に評価することである．そのため，OJS に提出されたソースコードは，コンパイル，複数のテストケース，リソース制限など，様々な観点で厳格に評価される．

提出したプログラムは，評価に何らかの問題があった場合，問題の内容に応じた途中結果が返され，不正解となる評価項目を全て通過した場合 Accept と見なされる [3]．通常のプログラムには文法エラーや論理エラーなど複数種類のエラーが含まれる場合もあるが，OJS は特定の評価項目を満たさなかった時点で結果がフィードバックされるため，1 回の提出で帰ってくる評価結果は常に 1 つとなる．また，たとえ Accept とならなかった場合でも，ユーザは何回でも提出し直すことができる．

OJS はプログラムのコンパイル，評価，結果のフィードバックを自動化可能であるため，多くの教育機関で導入されている．Dong ら [4] は青島大学のプログラミングコースにて導入されている OJS に対し，エラー予測モデルとプログラミング理解度の予測モデルを構築し，フィードバックの拡充を目指した．Xu ら [5] は OJS におけるデータに対し，深層強化学習を利用して学習者の行動やプログラム理解度の定量化を試みた．Liu ら [6] は OJS の社会的影響力の高まりに触れつつ，カテゴリの不十分さや解答の不十分さを指摘し，OJS における問題の自動分類および解説の紐付けを行った．吉村ら [7] は，特定の課題に対し類似した問題およびソースコードの例を提示する WOJ Recommender を提案し，ユーザが難易度の高い問題へ着手するための支援を行った．以上より，OJS がプログラミング教育に対して有効であることや，教育効果を高めるための支援に注目が集まっていることが分かる．

一方，学習者はどのようなエラーに躓くのか，各種エラーの解決はどれほど困難であるかなど，エラーの実態調査は，我々の知る限り行われていない．特定の教育機関において使用されている OJS では，学習者は類似した教育を受けていると考えられるため，生じるエラーなどに偏りが生じる可能性がある．そこで本研究では，世界中のユーザを持つ OJS である，AtCoder のデータを対象に，ユーザが提出したプログラムのエラー分析を行う．なお，対象となる AtCoder のデータは CodeNet にて提供されているものを扱う．

3. データと分析方法

3.1 CodeNet

本論文では IBM が提供する Project CodeNet[*6](以下 CodeNet と呼ぶ) に含まれる AtCoder の提出履歴を分析する．CodeNet では AtCoder の提出履歴のほかに AIZU Online Judge(AOJ) の提出履歴も含まれる．だが，データ数が AtCoder のみで十分にあることと，AtCoder では難易度で問題を分けてあることから AtCoder のみのデータに絞っている．CodeNet での AOJ の問題数は 2534 問，提出数は 1,943,791 件に対して，AtCoder での問題数は 1519

[*5] https://github.com/IBM/Project_CodeNet

[*6] https://github.com/IBM/Project_CodeNet

問，提出数は 11,961,010 件である．すなわち，AtCoder の方が提出数が約 6 倍多くなっており，AtCoder のみでも OJS におけるエラーの特徴を知るためには十分なデータ数であると考えた．また，元々 AtCoder では問題が複数の難易度に分類してある．AtCoder のみに絞ることで，今後の研究で難易度別に分けた分析を進めることができると考えたため，今回は AtCoder のみのデータを扱う．また，CodeNet のデータを AtCoder のみに絞った場合の言語割合の上位 5 つは C++が 58%, Python が 26%, Java が 4%, C が 3%, Ruby が 2%となった．言語ごとに生じやすいエラーや解決時間に差が生じる可能性があるが，言語ごとのプログラムの特徴調査については今後の課題としたい．

CodeNet のデータは問題別に csv ファイルで分かれており，csv ファイルの各行には 1 つの提出ごとのデータが含まれる．1 レコードに含まれる主な情報は提出 ID，提出した対象の問題 ID，ユーザ ID，提出日時，コンパイル対象の言語，ジャッジのステータス (状態)，CPU 時間，使用メモリ量，コードサイズである．ジャッジのステータスとは Accept や RE(実行時エラー) など，ユーザが提出したプログラムを，OJS がテストケースや検証器によって判定した結果を示す．また，問題ごとの csv ファイル内では言語別にデータがソートされている．今回はユーザごとに調査を行いたいため，ユーザ ID，その次に提出日時でソートを行ったデータを用いて，以降の RQ の分析を行う．

本論文では, 提出したプログラムが出力内容や実行時間などのすべての要件を満たした状態のことを Accepted(AC) と呼び，何らかの要件が満たされなかった状態を Reject と呼ぶ．そして Reject には満たされなかった要件によってさまざまな理由が存在する．Reject の理由は OJS によって様々であるが，AtCoder では主に，文法エラーや実行時エラーなどプログラムの不具合によるものと，実行時間超過やメモリ超過など問題の制約を超過した際に生じるものがある[*7]．以降，本稿では Reject の原因をまとめてエラーとして扱う．表 1 に AtCoder における Reject の原因となるエラーの一部をまとめる．一般的に 1 つのプログラムは複数のエラーを含む可能性があるが，2 章で述べたように，AtCoder では特定の評価項目を満たさなかった時点で Reject の結果が返却される．したがって，1 つの提出に対する結果は，Accept の場合は AC，Reject の場合はエラーの種類が 1 つと，常に 1 つのみ返却される．

次に, 学習者はどのようなエラーに躓くのか, 各種エラーの解決はどれほど困難であるかを分析するため，各 RQ を設定し，それぞれについて詳述する．

3.2 RQ1: Reject にはどのようなエラーが含まれるか

OJS では Reject の理由として様々なエラーが存在するが，具体的に提出結果の割合がどのようになっているのかはわからない．そこで，OJS 利用者にとって起こる頻度が多いエラーを特定する．優先して支援をした方がよいエラーはどのようなものなのか，そして，支援の効果が薄いものを除外するために，本調査を行う．

表 1: AtCoder における Reject の一部

エラーの種類 (略称)	原因
Compile Error(CE)	提出したプログラムがコンパイルに失敗する
Wrong Answer(WA)	一部のテストケースを満たさない
Runtime Error(RE)	プログラム実行中にエラーが生じる
Time Limit Exceeded(TLE)	問題で指定された実行時間以内にプログラムが終了しなかった

具体的な分析方法としては，CodeNet の AtCoder のみの部分に絞ったデータから，問題ごとに判定結果が Reject となった提出履歴のみを抜き出す．そして Reject の提出履歴を，エラーごとに分け，取得する．そして，問題別に分けられたエラー数をすべて合計し，AtCoder のすべての問題での AC 以外の状態数，すなわちエラーごとの提出数を取得する．

3.3 RQ2: 各種エラーはどれくらい解決可能か

RQ1 ではどのような種類のエラーが多く存在するのか分析するが，どのエラーが学習者にとって解決が難しいのかは分からない．そこで，現状の AtCoder では各種エラーがどれほど解決されているのか，エラーが複数組み合わさった場合に解決割合がどのように変わるのかを分析する．

分析内容としては 2 種類行う．まず，エラーの組み合わせごとの解決割合の調査を行ったのち，その結果を基に各種エラーが含まれている場合の解決割合を分析する．以下，それぞれの分析について詳述する．

3.3.1 エラーの組み合わせごとの解決割合

具体的な分析方法としては，特定の組み合わせのエラーが起こったユーザ数と，その組み合わせのエラーから AC にすることができたユーザ数を組み合わせごとに集計する．なお，複数の問題で同じユーザが提出していた場合でも，それぞれの問題でそのユーザを含めた集計を行う．つまり，複数の問題に対して提出しているユーザの除外などは行わない．

分析例として，ユーザ A が特定の問題に対し 2 回プログラムを提出しており，それぞれの提出の状態 (評価結果) が CE，RE だった場合と，さらにユーザ B が特定の課題に対し 3 回プログラムを提出しており，それぞれの提出の状態が CE，RE，AC だった場合を考える．まずユーザ A もユーザ B も，特定の課題に対し CE と RE の Reject を得たため，CE と RE の組み合わせのエラーとしてそれぞれ集計する．次に，ユーザ B は最終的に AC となりエラー

[*7] https://atcoder.jp/contests/abc313/glossary

を解決したため，CE と RE の組み合わせのエラーから AC にすることができたとして集計する．

そうして取得できたすべての組み合わせ，すべての問題を対象に，AC 以外の状態の組み合わせごとの解決割合を取得する．解決割合は以下の式で導き出す．

$$\text{エラー解決割合} = \frac{\begin{array}{c}\text{特定の組み合わせのエラーから} \\ \text{AC にすることができたユーザ数}\end{array}}{\text{その組み合わせが起こった全ユーザ数}}$$

3.3.2 エラーごとの解決割合

エラーごとの解決割合では，先ほどのエラーの組み合わせごとの解決割合のデータから，特定のエラーが含まれるデータを合計し，解決割合を調査する．例として WA に対して調査する場合，WA と CE，WA と RE と CE など WA が含まれる解決割合のデータをすべて合計し，WA が含まれる場合の解決割合を調査する．そして解決割合は以下の式に沿って調査する．

$$\text{エラー解決割合} = \frac{\begin{array}{c}\text{特定の状態が含まれたエラーから} \\ \text{AC にすることができたユーザ数}\end{array}}{\begin{array}{c}\text{特定の状態が含まれた} \\ \text{エラーが起こった全ユーザ数}\end{array}}$$

3.4 RQ3: 各種エラーから解決するためにはどれぐらいの時間がかかるか

RQ2 によってエラーの組み合わせごとやエラーごとの解決割合を調査できる．一方，解決割合と解決にかかる時間や作業量は常に比例するとは限らず，解決するために時間がかかるような特徴のあるエラーも存在すると考えられる．そのため，RQ3 では以下に示すエラーの組み合わせごとと，特定のエラーごとの解決に必要な時間を調査する．

3.4.1 エラーの組み合わせごとに AC になるまでにかかった時間

エラーの組み合わせごとに，エラーから AC になるまでにかかった時間を調査する．方法としては RQ2 で取得した，いずれかのエラーが含まれており，AC を出すことができたユーザを対象とする．なお，一部の提出履歴には，特定の提出から次の提出までの時間が非常に長いものが含まれる．翌日や数日後に提出された場合，エラーの問題解決にかかる時間を正確に計測できないと考え，本調査では解決までに 12 時間以上時間がかかっているユーザの提出履歴はデータから除外することにした．

解決時間の求め方は，**初めて AC となるプログラムを提出した時間から，初めてエラーとなったプログラムを提出した時間を引いた時間 (秒)** とし，得られたエラー解決時間を，エラーの組み合わせごとに記録する．

具体例としては，RQ2 で用いたユーザを例に取る．ユーザ A が特定の問題に対し 3 回プログラムを提出しており，それぞれの提出時刻は 12:00，12:03，12:10，結果は CE, RE, AC を得たとする．この場合，CE と RE の組み合わせのエラーとなる．そして，ユーザ A が AC となったプログラムを提出した時間 (12:10) から，初めてエラーが生じた CE となったプログラムを提出した時間 (12:00) の計算結果である 10 分，すなわち 600 秒を，CE と RE の組み合わせの解決時間として記録する．

3.4.2 特定のエラーから他のエラーへ遷移するまでの時間

本調査は，特定の単体エラーから他の単体エラーへ遷移したものを対象する．本調査でも，遷移までに 12 時間以上かかっているユーザは，その時間全て問題解決のために試行錯誤したとは考えにくいため，除外する．遷移時間は**他の状態に遷移した際の提出時間から，初めて遷移前の状態になった提出時間を引いた時間 (秒)** とし，状態の遷移ごとに調査する．

具体例として，前節で用いたユーザ A を例に取る．ユーザ A は 12:00 に CE，12:03 に RE，12:10 に AC の結果を得た．この場合，(1)CE から RE への遷移と (2)RE から AC への遷移をそれぞれ集計する．(1) の CE から RE への遷移は，CE が解決され異なるエラーである RE に変化したとみなす．RE が発生した時刻が 12:03 で CE が発生した時刻が 12:00 であるため，CE から RE への遷移には 3 分，すなわち 180 秒かかったと記録する．同様に (2) の RE から AC への遷移も，AC が発生した時刻が 12:10，RE が発生した時刻が 12:03 であるため，遷移に 7 分，すなわち 420 秒かかったとして記録する．

また，ユーザによっては同様のエラーが複数回生じる場合もある．例えばユーザ C が特定の問題に 5 回プログラムを提出し，TLE，CE，CE，CE，AC の結果を得たとする．この場合，1 回目の提出から 2 回目の提出にかけて TLE が CE に遷移しており，2 回目の提出から 5 回目の提出にかけて CE が解決された．そのため，ユーザ C の提出履歴からは，TLE から CE に遷移した時間と，初めて CE が生じた時刻 (2 回目提出時の時刻) から AC を得た時刻までの 2 つのデータを取得する．

4. 結果と考察

4.1 RQ1: Reject にはどのようなエラーが含まれるか

図 1 は AC 以外の状態の提出の結果である．図 1 から見て取れるように，CodeNet における AtCoder の Reject に関する状態は，ほとんどが 4 つのエラーで構成される．このグラフにある WA, RE, TLE, CE とは表 1 にあるエラーの種類のものと同一である．

結果，WA が 66%，RE が 14%，TLE が 12%，CE が 8%，その他の状態が 1%未満という結果となった．そのことから，優先して支援をすべきエラーは表 1 に示した WA, RE, TLE, CE であり，それ以外のエラーに関しては支援の効果

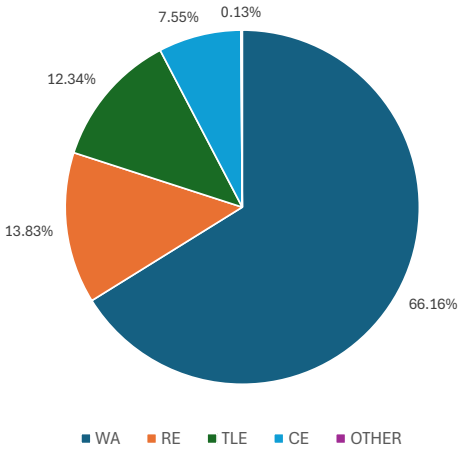

図 1: エラーの提出割合

が薄いと考える．そのため，これ以降の調査では WA, RE, TLE, CE を対象とし，それ以外のエラーの提出は除外し，調査を進める．

4.2 RQ2: 各種エラーはどれくらい解決可能か

4.2.1 エラーの組み合わせごとの解決割合

図 2 にエラーの組み合わせごとの解決割合を示す．結果として，エラーが CE のみの場合の解決割合が 91%と一番高く，エラーに RE と TLE が含まれている場合の解決割合が 68%と一番低い結果となった．また，エラーの組み合わせが多くなるほど解決割合が低くなるわけではなく，エラーの種類の組み合わせによって解決割合が変わることがわかった．

4.2.2 エラーごとの解決割合

エラーごとの解決割合では，図 2 のデータから，特定のエラーが含まれるデータを合計し，解決割合を求めている．その結果，図 3 のようになった．結果として，TLE, RE, WA, CE の順に解決割合が高くなる結果となった．特に TLE では解決割合の順位が一つ上の RE と 5%以上も離れており，TLE はほかのエラーと比べて解決が難しいことがわかる．

CE や WA の解決割合が 80%を超えていた理由として，CE に関してはエラー文がユーザのプログラミング環境にて確認でき，解決すべき問題がわかるためだと思われる．WA では問題文や出力例などを参考にエラーの解決を行いやすいため，解決割合が他のエラーと比べて高くなったのではと考える．また，TLE や RE の解決割合が 80%を超えなかった理由としては，TLE や RE はエラーの原因がユーザのプログラミング環境で確認，再現が難しいため，解決が WA や CE に比べて難しかったのではと考える．

特に TLE に関してはソースコードのアルゴリズム自体が間違っており，実行時間制限に間に合っていないため，使用するアルゴリズムをまた考え直すことが必要な場合が多い．以上の理由により，TLE が一番解決割合が低い結果となったと考える．

4.3 RQ3: 各種エラーから解決するためにはどれぐらいの時間がかかるか

4.3.1 エラーの組み合わせごとに AC になるまでにかかった時間

図 4 はエラーの組み合わせごとに AC になるまでにかかった時間の結果である．主にエラーの組み合わせの数が増えるほど解決時間が長くなる傾向にあることがわかった．エラーの組み合わせの数が増えるということは，少なくともその組み合わせの個数分エラーを出しているため，解決時間が長くなると考える．

単体 TLE とその他の単体エラーの解決時間に対して，Welch の T 検定を有意差 0.05 で行ったところ，すべて有意差が確認された．CE, RE, WA のエラーは，複数のエラーが組み合わせられた場合に比べて，単体であるときが解決にかかる時間は短い．一方，TLE 単体のエラーは，RE, CE の組み合わせと，WA, CE の組み合わせのエラーよりも解決に時間がかかる．RQ2 の調査より，TLE は解決が難しいことが分かったが，加えて TLE は解決にかかる時間も長いことが本調査より判明した．

4.3.2 特定のエラーから他のエラーへ遷移するまでの時間

図 5 に AC 以外の状態から他の状態へ遷移するまでの時間の結果を示す．CEtoAC は CE から AC に遷移するためにかかった時間，TLEtoRE は TLE から RE に遷移するためにかかった時間の合計を示す．一部異なる部分はあるが，概ね CE, RE, WA, TLE が発生した順に遷移時間が長くなっている．

RQ2 の結果から RE と WA では WA が RE よりも解決割合が高いという結果となった．一方，図 5 より，REtoAC と WAtoAC，REtoTLE と WAtoTLE，REtoCE と WAtoCE を比べて RE の方が WA よりも遷移時間が短い結果となった．つまり，RE から他の状態への遷移時間は WA から他の状態への遷移時間よりも短い結果となった．RE から AC，WA から AC など，RE と WA から他の状態への遷移時間で Welch の T 検定を有意水準 0.05 で行ったところ，AC と TLE への遷移時間は有意差あり，CE への遷移時間は有意差なしという結果となった．つまり，WA の方が解決割合が高いが，少なくとも AC と TLE への遷移に関しては RE の方が遷移時間が早いことがわかった．

また，図 5 より，特定のエラーから AC へと遷移する時間は，エラーからエラーに遷移する時間と比べて，遷移時間が短いということが分かった．これもエラーから AC となる遷移時間と，AC 以外のエラーとなる遷移時間で Welch の T 検定を有意水準 0.05 で行ったところ，すべて有意差が確認された．よって本調査の結果より，一定時間を超えてエラーを解決できない場合，自力での解決が困難となる

図 2: エラーの組み合わせごとの解決割合

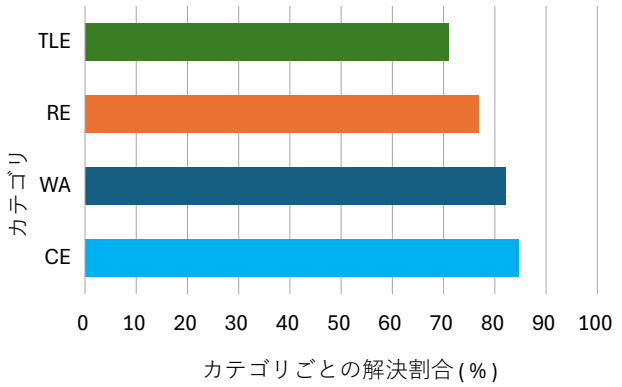

図 3: エラーごとの解決割合

可能性が上がるため，自動でヒントを与えるなどの支援に繋げることができる．

他にも，図 5 より，TLE から他の状態に遷移するまでの時間が，他のエラーから遷移する場合と比べて長い．よって，TLE を解決するために時間がかかることがわかった．そこで，TLE から他の状態に遷移するまでにかかった時間と，TLE 以外のエラーから他の状態に遷移するまでにかかった時間を，Welch の T 検定を有意水準 0.05 で行ったところ，すべてに有意差が確認された．以上より，TLE が他のエラーの中でも解決するために最も時間のかかるエラーであると考えられる．

さらに，図 5 から，TLE 以外の状態から TLE へと遷移する場合に関しても，他の状態へと遷移する場合と比べて一番長いことがわかる．よって，TLE へと遷移するプログラムは，遷移前のエラーを解決するために時間がかかることがわかった．こちらも TLE 以外のエラーから TLE へと遷移するまでにかかった時間と，TLE 以外のエラーから他の状態へと遷移するまでにかかった時間で Welch の T 検定を行った．有意水準 0.05 で Welch の T 検定を行ったところ，WA から TLE への遷移と WA から RE への遷移のみ有意差が確認されず，それ以外は有意差が確認された．すなわち，WA については TLE へと遷移するものが遷移時間が長いかはわからないが，それ以外のエラーから TLE に遷移するものは解決に時間がかかることが分かった．

4.4 妥当性への脅威

内部妥当性として，RQ2 や RQ3 において，解決割合や解決までにかかった時間について調査を行ったが，AtCoder は他者のソースコードや公式の解答を閲覧可能である．そのため，一部のユーザの提出プログラムは解答例を見て書いたものもある可能性があり，ユーザーの実際の解決割合はさらに低い可能性がある．そのため，より詳しく解決割合を調査する際に，解答例が出ている問題か出ていない問題か分けて解決割合を分析する必要がある．

外部妥当性として，AtCoder のみのデータを対象としたため，他の OJS でも同じ結果となるかは分からない．だが，簡単な問題はそれほど問題が複雑にならないため，OJS ごとの特徴などは現れにくく，他の OJS での問題と本調査の結果はそれほど変わらないと考える．一方で複数のアルゴリズムを使用するような難しい問題の場合，OJS ごとの特徴によってエラーごとの解決割合などの結果の一部が

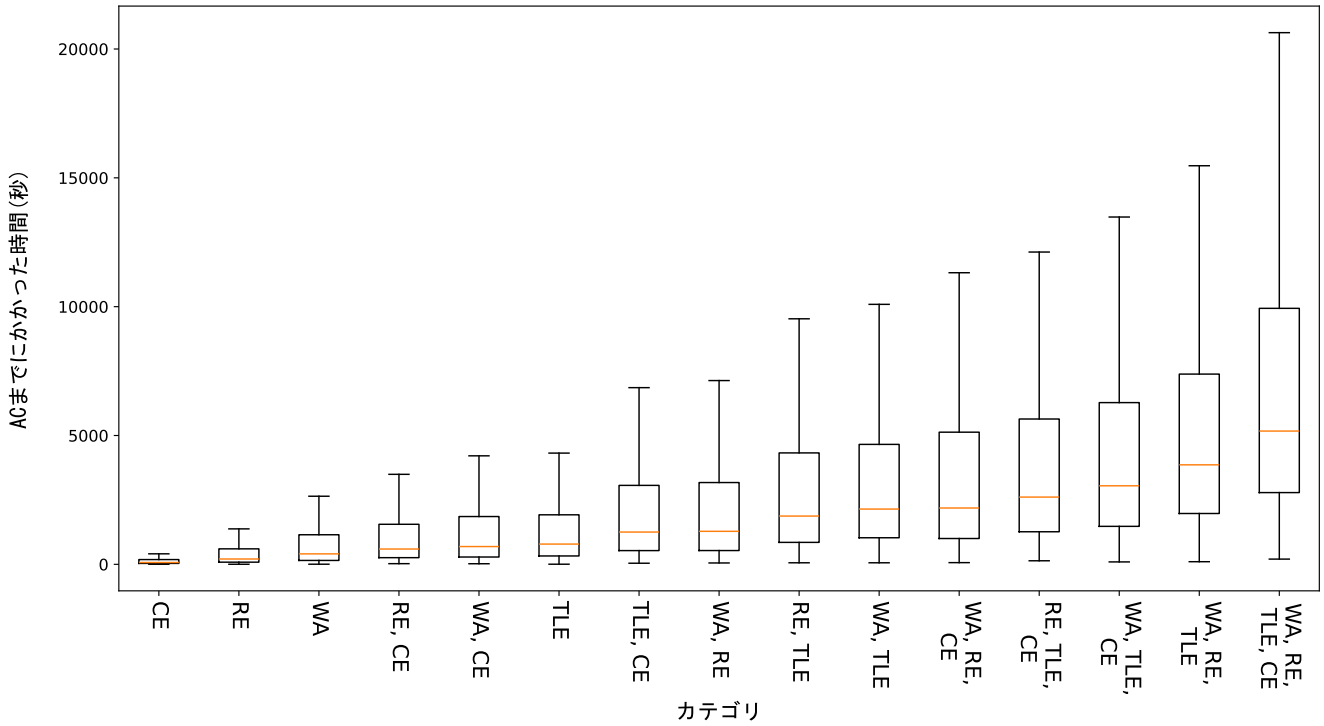

図 4: エラーの組み合わせごとに AC になるまでにかかった時間

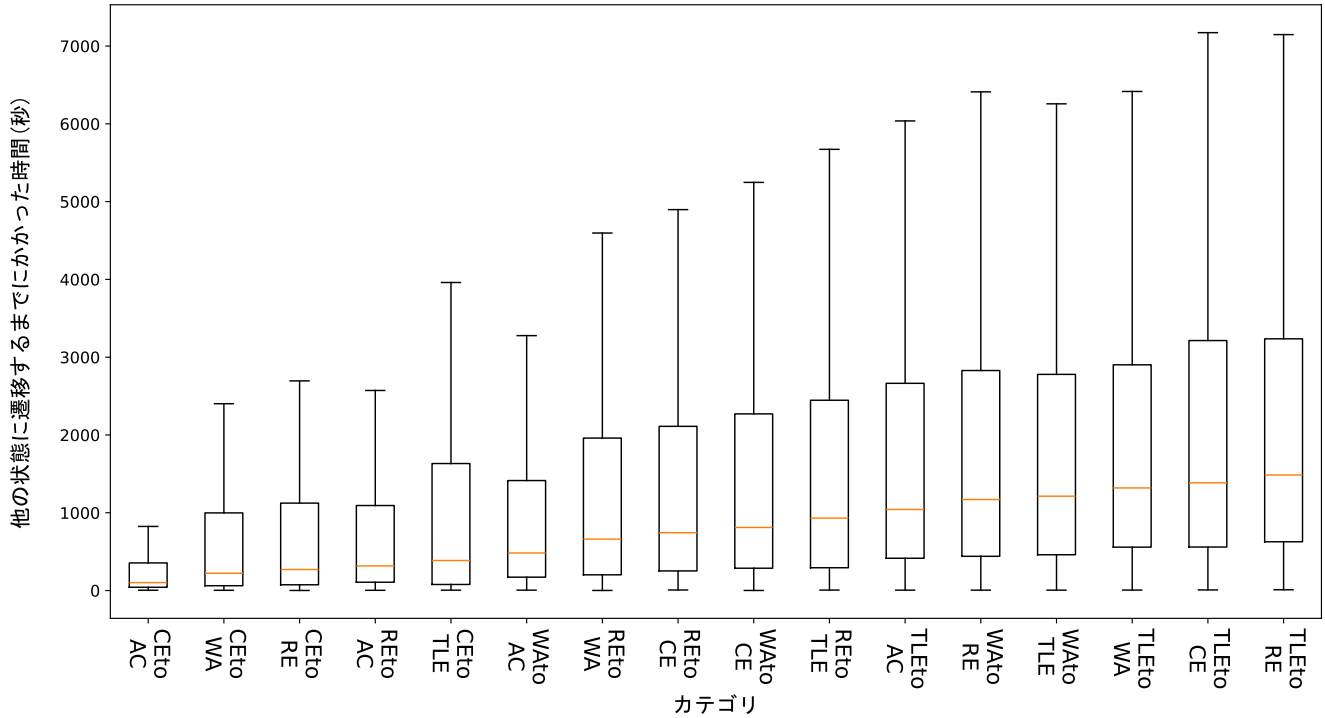

図 5: 特定のエラーから他のエラーへ遷移するまでの時間

変わる可能性がある．問題の難易度ごとのエラー解決割合の調査は，今後の課題とする．

構成概念妥当性として，RQ3 においてエラーの組み合わせを使用し，AC までにかかる時間や遷移するまでにかかった時間の調査を行った．しかし，AtCoder 含め多くの OJS では返却される Reject の原因が 1 つである以上，2 回目以降に出現するエラーがいつ含まれていたかが不明瞭である．例えば，CE，RE，AC と結果が遷移した場合，1 回目に CE の結果を受けているが RE となるエラーも含まれている可能性がある．したがって，今後は Reject の原因となるエラーがいつ混入したか，ソースコードの分析も交える必要がある．

4.5 関連研究

教育機関におけるプログラミング演習を対象にした論理エラーの調査研究は多く存在する．MacNeil ら [8] は論理エラーのわかりにくさを指摘し，大規模言語モデルを用いた論理エラーの検出およびフィードバックを目指した．Miao[9] も既存のプログラミング環境は論理エラーに対するフィードバックが不十分であることを述べており，Bosse ら [10] の調査によると，構文エラーに比べて論理エラーを発生させた学生は，授業をドロップアウトする傾向にあった．以上より，論理エラーは構文エラーに比べて解決が困難である一方，既存のプログラミング環境やコンパイラでは十分なフィードバックが行われておらず，初学者のドロップアウトの原因の 1 つであることが分かる．

Ettles[11] らや Albrecht[12] らは学生が実際に引き起こした論理エラーの分類を行った．Ettles ら [11] は論理エラーを，アルゴリズムの誤り，問題の理解不足，知識不足の 3 つに分類し，知識不足が起因した論理エラーが最も頻繁に生じ解決に時間がかかることが分かった．Albrecht ら [12] は 1.2 万件以上のエラーログを手動で 6 つのカテゴリに分類した結果，単純なタイプミスや，演習の制約を無視した論理エラーが頻出したことが分かった．OJS でも入出力のフォーマットは厳格に指定されているため，Ettles らや Albrecht らと同様の原因による論理エラーが含まれる可能性は高い．実際に OJS では WA が最も多く，WA は問題の読解ミスで生じる場合もある．一方，本研究で得られた，解決に最も時間がかかる論理エラーの種類は，Ettles らや Albrecht らの調査結果とは異なった．したがって，本研究で得られた TLE が関わるエラーが最も解決困難である結果は，OJS 独自の問題である可能性が高く，今後より調査を深めることで，OJS を利用したプログラミング教育やリクルーティング活動に寄与できると考える．

5. おわりに

本論文では OJS における Reject 時のフィードバックの拡充や，Reject 解決の支援のため，CodeNet の AtCoder のデータを分析した．3 つの RQ を設定し，Reject の状態ごとの割合やその解決割合，AC になるまでにかかった時間や遷移時間を分析し，主に起こっているエラーやその割合，エラーの組み合わせごとの解決割合や解決時間，他の状態へと遷移するまでにかかった時間などからエラーの実態の調査を行った．その調査の結果，WA, RE, TLE, CE が主要なエラーであり，その中でも TLE の解決割合が一番低く，解決にかかる時間が一番長い結果となった．また，AC へと遷移する場合，エラーへと遷移する場合と比べて遷移時間が短いという結果を得た．

今後の展望として，問題の難易度で分けることでエラーの割合や解決割合などがどのように変わるのか分析することや，提出結果だけではなくソースコードの編集量からエラーの組み合わせごとにどのような違いが表れるのか調査していきたいと考える．

謝辞 本研究は JSPS 科研費 24K20905 の助成を受けた．

参考文献

[1] Andy Kurnia, Andrew Lim, and Brenda Cheang. Online judge. *Computers & Education*, Vol. 36, No. 4, pp. 299–315, 2001.

[2] Szymon Wasik, Maciej Antczak, Jan Badura, Artur Laskowski, and Tomasz Sternal. A Survey on Online Judge Systems and Their Applications. *ACM Computing Surveys (CSUR)*, Vol. 51, No. 1, pp. 1–34, 2018.

[3] 渡部有隆. オンラインジャッジの開発と運用 -Aizu Online Judge-. 情報処理, Vol. 56, No. 10, pp. 998–1005, 2015.

[4] Yu Dong, Jingyang Hou, and Xuesong Lu. An Intelligent Online Judge System for Programming Training. In *In Proc. of the International Conference on Database Systems for Advanced Applications(DASFAA)*, pp. 785–789, 2000.

[5] Yuhui Xu, Qin Ni, Shuang Liu, Yifei Mi, Yangze Yu, and Yujia Hao. Learning Style Integrated Deep Reinforcement Learning Framework for Programming Problem Recommendation in Online Judge System. *International Journal of Computational Intelligence Systems*, Vol. 15, No. 114, pp. 1–22, 2022.

[6] Jianyu Liu, Shaohong Zhang, Zongbao Yang, Zhiqian Zhang, Jing Wang, and Xiaofei Xing. Online Judge System Topic Classification. In *In Proc. of the 14th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery(ICNC-FSKD)*, pp. 993–1000, 2018.

[7] 吉村涼矢, 坂本一憲, 鷲崎弘宜, 深澤良彰. プログラミング教育のための類題出題システムの提案. 研究報告コンピュータと教育（CE）, pp. 1–6, 2020.

[8] Stephen MacNeil and Paul Denny and Andrew Tran and Juho Leinonen and Seth Bernstein and Arto Hellas and Sami Sarsa and Joanne Kim. Decoding Logic Errors: A Comparative Study on Bug Detection by Students and Large Language Models. In *In Proc. of the 26th Australasian Computing Education Conference(ACE)*, pp. 11–18, 2024.

[9] Dezhuang Miao, Yu Dong, and Xuesong Lu. PIPE: Predicting Logical Programming Errors in Programming Exercises. In *In Proc. of the International Conference on Educational Data Mining(EDM)*, pp. 473–479, 2020.

[10] Yorah Bosse and Marco Aurélio Gerosa. Why is programming so difficult to learn?: Patterns of Difficulties Related to Programming Learning Mid-Stage. *ACM SIGSOFT Software Engineering Notes*, Vol. 41, No. 6, pp. 1–6, 2017.

[11] Andrew Ettles, Andrew Luxton-Reilly, and Paul Denny. Common logic errors made by novice programmers. In *In Proc. of the 20th Australasian Computing Education Conference(ACE)*, pp. 83–89, 2018.

[12] Ella Albrecht and Jens Grabowski. Sometimes It's Just Sloppiness - Studying Students' Programming Errors and Misconceptions. In *In Proc. of the 51st ACM Technical Symposium on Computer Science Education(SIGSCE)*, pp. 340–345, 2020.