システム創成工学専攻
情報システムコース
Department of Systems Innovation
Advanced Information System Course

平成30年度 専攻科特別研究論文

ソースコードのスナップショットに基づいた無作為修正の検出

Detection of Random Correction from Source Code Snapshots

指導教員名　上野 秀剛 准教授
論文提出者名　大野　優

独立行政法人 国立高等専門学校機構
奈良工業高等専門学校 専攻科
National Institute of Technology, Nara College
Faculty of Advanced Engineering

ソースコードのスナップショットに基づいた無作為修正の検出

Detection of Random Correction from Source Code Snapshots

大野　優
Yu OHNO

独立行政法人 国立高等専門学校機構
奈良工業高等専門学校 専攻科 システム創成工学専攻 情報システムコース
大和郡山市矢田町 22 番地（〒 639-1080）
National Institute of Technology, Nara College, Faculty of Advanced Engineering
22 Yata-cho, Yamatokoriyama, Nara 639-1080, Japan

Abstract: Classifying student's situation helps improve educational effect in programming course with snapshots. Snapshots can grasp student who falls "pitfall" during a course. The purpose of this study is to classify students who make random correction in the programming course with Online Judge System. Random Correction is an action that source code correction without understanding the exercise contents. Then we propose metrics to classify students who make random corrections from snapshots of source code submitted by students and verify their usefulness. The result of the experiment shows that students who cannot reach perfect score had high value of both metrics; 1) a degree of imbalance corrections between source code lines, 2) the number of submitted revisions.

Keywords: Programming Education; Online Judge System; Automatic Detection;

関連業績リスト

1. 大野 優，上野 秀剛，内田 眞司：ソースコードのスナップショットに基づいた無作為修正者の検出（教育工学:ET），電子情報通信学会技術研究報告，vol. 118，no. 214，ET2018-37，pp. 53–58，2018.

2. Y. Ohno, H. Uwano, S. Uchida: Detection of Random Correction from Source Code Snapshots, *Proceedings of the 8th International Conference on Software and Computer Applications (ICSCA 2019)*, 2019（予定）.

目次

図目次

表目次

1. 序論

教育現場におけるプログラミング教育では座学と演習を組み合わせた授業形式が一般的である．授業では少数の教員が多数の受講者を対象に講義を行うため，演習中に個々の受講者についてその理解状況や作業の様子を把握することが難しい．また，理解していない学生を把握できた場合でも具体的に何を理解しておらず，どのような指導が適切なのか判断することは時間がかかる．

教育効果を高めるために講義の受講者を対象とした状況識別が研究されている [1–8]．その中でも，プログラミング講義を対象にスナップショットを用いた研究が複数報告されている [9–19]．スナップショットはある時点でのプログラムの情報であり，ソースコード，日時，エラー情報などが記録される．先行研究は，プログラミング受講者のスナップショットから受講者がどのような箇所で行き詰まっているか把握する．本研究では，各受講者のスナップショットから算出されるメトリクス値をもとに受講者の状況識別を行う．

プログラミングの学習環境の 1 つとして，Online Judge System(OJS) がある．OJS は課題に対応したソースコードを学生が作成し提出すると，コンパイル・実行し，その実行結果を元にソースコードを採点する．プログラミング講義に OJS を用いることで，受講者は演習課題で作成したソースコードを OJS に提出し即座に採点結果を得ることができる．本研究では，OJS を用いたプログラミング講義を対象とする．受講者は OJS にソースコードを随時提出するため，提出ごとのすべてのソースコードがスナップショットとして記録される．そのため，スナップショットから各受講者がどのような編集をしたのか履歴をとることができる．OJS から得られるソースコードの編集履歴から各受講者の行動や理解状況を把握できれば，演習中に対象を絞った指導をすることが可能となる．

本研究では，無作為修正を対象として，編集履歴からの識別を試みる．無作為修正とは，誤答となった原因を考察せずにソースコードを修正する様子を指す．従来のプログラミング講義では受講者はソースコードを 1 度提出するだけで，評価もすぐにはわからない．そのため，受講者が提出したソースコードが演習で課された要求を満たしていないと

き，受講者は誤答であることに気づきにくい．一方で，OJS による講義では，提出したソースコードの評価がすぐにわかる．そのため，誤答に対して原因を考察し修正する受講者や，原因を考察せず無作為に修正する受講者など複数の振る舞いが考えられる．無作為な修正によってプログラムが正しく動いた場合，与えられた課題の仕様やアルゴリズム，新たに学習した文法などを理解せずに演習を終える可能性がある．演習中に無作為修正を識別することで，教員が指導して課題が設定された単元の理解を促すことができ，授業時間内の指導を効率化し，より多くの受講者が理解に辿り着くことができると考えられる．

本論文の構成は以下の通りである．2 章では，様々な講義およびプログラミング講義における受講者の状況識別に関してこれまで報告された研究について述べる．また，本研究で対象としている OJS に関連する研究についても述べる．3 章では，本研究で対象としている OJS によるプログラミング講義，および識別対象である無作為修正の定義・例・調査観点について述べる．4 章では，無作為修正を識別するために提案したメトリクスとその算出方法について述べる．5 章では，4 章で提案したメトリクスの有用性の評価に関する 3 つの実験とその結果について述べる．6 章では，結論と今後のまとめについて述べる．

2. 関連研究

2.1 受講者の状況識別

講義の教育効果を高めるために受講者を対象とした状況識別が研究されている．Yoshihashi らは高等教育機関における授業の教育効果を改善することを目的に，畳み込みニューラルネットワーク (CNN: Convolutional Neural Networks) を用いて聴講者の受講状態を識別するシステムを提案している [1,2]．システムはカメラで教室前方から後方に向かって撮影された授業映像から聴講者を検出する．検出された各聴講者に対して，検出された各聴講者の状態を CNN を用いて推定し，「集中」,「非集中」,「聴講者不在」の 3 種類の状態ラベルを割り当てる．Baradwaj らは，受講者の講義出席状況，試験の点数，課題の提出状況などから，その受講者の学期末時点の成績を決定木学習を用いて予測している [3,4].

また，学習管理システム (LMS: Learning Management System) などの授業支援システムを導入した講義で得られたデータに対してデータマイニングの手法を用いた「学習分析」(LA: Learning Analytics) や Educational Data Mining(EDM) に関する研究が報告されている [5–8]．加藤は LA や EDM の利点を以下のように述べている [7,8].

1. 学習者の傾向と行動パターンの解読
   学習者の傾向と行動パターンの解読により，脱落の恐れがあるグループを発見して，より良い教授法を構築し，在籍率向上が期待できる．
2. 理解度不足の学習内容と行き詰まり原因の推定
   理解度不足の学習内容をシステムが把握することにより，学習者に最良の学習方針を提案することが期待できる．
3. 到達学力の推定
   到達学力を推定することで，学習者の進捗状況に対してリアルタイムに反応して教材への取り組みを深められる教育ソフトウェアや適応学習環境の設計に役立つ情報

が得られる可能性がある．

以上の研究報告から，講義の受講者の状態や行動を記録することで，講義の雰囲気や受講者の集中度合いや取り組み状況を自動的に推定できる．これにより，指導すべき受講者を自動で識別できるため，教員は効率よく指導できることが期待される．

2.2 プログラミング講義における状況識別

プログラミング講義を対象に受講者の状況識別を行った研究が複数報告されている．Ihantola らは，プログラマがプログラムの実装時に行なったコーディング，コンパイル，デバッグ，テストなどによって作成される成果物，およびそれに対応するメトリクスを通してプログラミング行動が計測できると述べている [9]．Dutt らの文献によると，Zimmerman は継続的にプログラミング行動を収集・観察できる環境は教育現場において重要であると述べている [10]．本研究で識別に用いるスナップショットも，プログラムのコーディングで得られる成果物であり，プログラミング行動を計測できると考えられる．

Fujiwara や伏田らはプログラミング演習時に取得した初学者のソースコード編集履歴を分析し，その傾向を明らかにした [11,12]．分析では (1) プログラミング熟練者の目視による定性的分析と，(2) ソースコードのトークンレベルでの Levenstein 距離の計測に基づく定量的分析を実施した．分析 (1) の結果，初学者が誤りに陥っている際に本来修正すべき箇所ではなく，課題内容とは関係の無い箇所を編集するといった行動パターンが観測された．分析 (2) の結果，演習において理解困難な状態に陥っている場合，距離の変動を確認することで初学者が誤りに陥っているか検出できる可能性があることがわかった．

また，同著者は受講者のソースコード編集履歴を分析し，どのような箇所で行き詰まっていたのか特定する手法を提案した [13]．提案手法は，ある時点における演習課題のソースコードを，演習課題完了時のソースコードにするために必要な作業を推定残作業量として定量化する．推定残作業量の変化を元に，受講者が作成しているソースコードが正解に近づいていない，行き詰まり区間を検出し，その区間のスナップショットからどのような箇所で行き詰まっていたのかを特定する．

槇原らは探索的プログラミングと呼ばれる，プログラムの実現方法が定かではないときに学習者が修正・コンパイル・実行を繰り返すプログラミング行動に着目した手法を提案している [14]．手法は学習者がプログラムのどこで探索しているかを検出し，課題に対する取り組みや行き詰まりをリアルタイムに特定する．

藤原らはプログラミング学習者の学習状況から測定されるソースコードの行数，コンパイル回数，プログラムの実行回数などの時系列データから学習者の特徴を定量化する手法を提案した [15]．複数の時系列データをグラフで可視化し，関連性のあるところに対してデータ値の増減，データ値の変化量，データ値の上限， データ値の下限，データの時間微分値の 5 つの定性的な表現に当てはめ，特定の学習状況に現れるデータ値の推移の特徴を数式として定量化する．分析の結果，一定時間でのソースコードの行数の増減やプログラムの実行回数の増加などから学習者の行き詰まりを定量化できることがわかった．

Jadud らは初学者向けプログラミング講義を対象に，エラーの発生と修正の傾向を識別する手法を提案した [16]．手法は発生したエラー数と修正から求められる値である Error Quotient (*EQ*) を用いる．演習の点数と講義の成績のそれぞれと *EQ* に相関関係があるか検証した結果，両方に負の相関があることが示された．また，Watson らは *EQ* の算出にエラーに対してどれだけ時間をかけて考察したかを考慮した *Watwin* を提案した [17]．また，*EQ* と *Watwin* のそれぞれにおいて機械学習を用いて受講者を識別している [18]．

Blikstein は自由解答 (Open-ended) 形式によるプログラミングの課題に対するスナップショットからコンパイル回数やソースコードのサイズの増減，コンパイルの成功・失敗の回数などを算出し，受講者の状況や振る舞いを分析した [19]．

以上の研究は，いずれもプログラミング受講者のスナップショットを分析することで，演習中の受講者の状況を把握している．本研究は，Online Judge System(OJS) を用いたプログラミング講義を対象とし，識別対象を無作為修正に限定している点でこれらの研究と異なっている．本研究はプログラミング学習者の中でも無作為修正に対象を絞り，スナップショットから得られるメトリクスを用いてその識別を試みる．

2.3 Online Judge System に関連する研究

本研究では Online Judge System(OJS) を用いたプログラミング講義を対象としている．OJS は Automatic Assessment(AA) Tool，AA System とも呼ばれており，これまでにプログラミング教育現場での活用 [20]，OJS の実装（特定の機能を追加した OJS の実装 [21,22]，学習管理システム (LMS: Learning Management System) とのプラグイン連携に関する調査 [23] など），データマイニング [24–26] など多岐に渡って研究されている [20–33]．本論文では，プログラミング教育現場での OJS の活用による影響について述べる．

2.3.1 プログラミング教育現場での OJS の活用のメリット

OJS は自動でプログラムを採点するため，手動での採点（手動でプログラムを実行し，目視で実行結果を確認する方法）に比べ，OJS はより短い時間で，かつ人手がより少ない状態で採点できる [30,31].

則行らは，OJS でのプログラミング学習者の解答履歴データを用いて，学習効果が十分に得られるように各学習者の能力に適した問題を提案するアプローチを提案した [24]. 岩本らはプログラミング講義受講者の演習課題に対するソースコードの不正コピーを検出する OJS を実装した [21,22]．著者らは，コードクローン検出手法を用いて，ソースコードにおける不正コピーを検出している．岩本らのシステムによって，ソースコードの不正コピーを常習的に行う受講者を特定できるようになり，演習課題に対して手をつけられない，全くやる気の無い受講者に対する支援が可能となった.

以上の研究報告から，プログラミング教育現場に OJS を導入することで，教員の負担を軽減する効果や特定の性質を持つ受講者を識別できると期待されている.

2.3.2 プログラミング教育現場での OJS の活用のデメリット

OJS による受講者への採点結果のフィードバックによって，受講者にプログラミング演習中に誤答に対する原因を考える機会を与えられる一方で，1 章で述べた無作為修正（誤答の原因を考察せず，ソースコードを修正する）を繰り返す受講者が発生することが考えられる点が指摘されている [29,32]．Ben-Ari は誤答の原因を考察せず，多くの回数をかけてソースコードを修正する行為を *bricolage* と定義しており，開発の上で非効率であると述べている [32].

これらの研究報告に対し，受講者の無作為修正を防ぐための研究がいくつか報告されている．Karavirta らは，ソースコードの修正・提出できる回数を制限して受講者に修正内容を考察させることが無作為修正の抑制に有用かどうかを検証した [25,26]．Guerreiro らは，フィードバックのコメント量を制限することで，受講者に修正内容を考察させようと試みた [33]．また，松永は授業中に紙と鉛筆を取り入れ，プログラムの作成・修正の前に，論理的に正しいかどうか考えさせる手法を試みた [20]．これらの研究では，プログラミング講義の受講者から得られるデータを細かく採ることができず，受講者が無作為修正を行っているかどうかを判断することが困難となる．また，修正内容を考えて修正してい

る中で提出の制限回数に到達してしまったとき，受講者の考えている時間や努力が無駄となるため，受講者のモチベーション低下を生んでしまうことが考えられる．そのため，提出回数やフィードバックの制限を無くすことが必要であると考えられる．

以上の研究報告から，受講者にプログラミング演習中に誤答に対する原因を考察せず，試行錯誤にソースコードを修正する受講者が発生することがデメリットとして考えられている．本研究では，無作為修正を行う受講者を OJS に提出されたソースコードのスナップショットから算出されるメトリクス値で識別できるか検証する．

3. 準備

3.1 OJS を用いたプログラミング講義

本研究が対象とするプログラミング講義は，ある単元に対して座学の後に演習課題を解く形式である．受講者は演習課題に対してオンラインエディタで作成したソースコードを OJS に提出する．OJS は提出されたソースコードをコンパイル・実行し，教員があらかじめ作成したテストケースと一致するかどうかを判定する．テストケースとは作成したプログラムが要件を満たしているかテストするための入力と，入力に対して望まれる出力の組みを表す．テストケースの出力と提出されたプログラムの出力の一致した数に応じて以下の式で点数 *score* が計算される．

$$score = \frac{\text{テストケースの正解数}}{\text{テストケース数}} \times 100 \tag{3.1}$$

受講者は各テストケースの正誤判定結果，*score*，コンパイル時のエラー，実行時エラーを得る．受講者は *score* が 100 点になるまで，ソースコードの修正と提出を繰り返す．作成されたソースコードは提出ごとにリビジョンとして，提出日時，点数とともに記録される．

3.2 無作為修正

本研究では演習の内容を理解しないままソースコードの修正を繰り返す行動を「無作為修正」と定義する．図 3.1 に無作為に行われる修正の例を示す．図はある課題におけるある受講者の提出履歴である．この受講者はリビジョン 1 の提出から 5 分間で 20 回提出している．また，すべてのリビジョンを通して print 文の表示内容だけを修正している．無作為修正を行う受講者は複数のリビジョンをかけて，if 文や for 文の条件式，print 文の中身といった，エラーの原因と思われる箇所に対し自分が持つ修正パターンを闇雲に適用

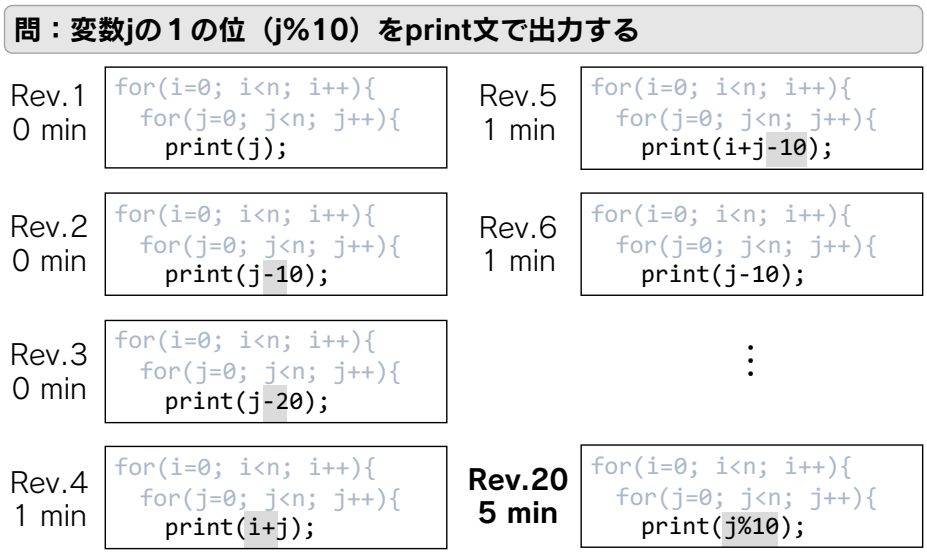


図 3.1: 無作為な修正の例

する傾向がある．なお，本研究ではソースコードの修正を「追加」,「削除」,「変更」の 3 つの動作から成ることとする．

受講者は演習中に作成したソースコードにエラーが出たり，仕様通りの動作にならないといった行き詰まりに陥った場合，自分で理由を考えたり，調査した上でソースコードを修正するのが理想である [20]．しかし，無作為な修正によってプログラムが正しく動いた場合，与えられた課題の仕様やアルゴリズム，新たに学習した文法などを理解せずに演習を終える可能性がある．演習中に無作為修正をリアルタイムに識別することで，教員が指導して課題が設定された単元の理解を促すことが可能になると考えられる．

本研究では，無作為修正を，ソースコードの修正行動に対する以下の 3 つの観点で計算したメトリクスを用いて識別する．

P1： 修正箇所が偏っている
P2： 一定時間に修正・コンパイルした回数が多い
P3： 1 回あたりの修正量が小さい

4. 提案手法

4.1 メトリクス

提案手法は以下の 4 つのメトリクスを用いて無作為修正の特徴を把握する．提案手法は，演習の時間中に無作為修正を識別し，指導を行うことを目的としているため，リビジョン 1 の提出から t 分間に提出されたソースコードの変更履歴からメトリクスを計算して識別に用いる．

$Freq(t)$： 修正行の偏り
$Revs(t)$： t 分間のリビジョン数
$DiffLine(s,t)$： s 行以下の修正が続いた回数
$AveDiff(t)$： 1 回あたりの平均修正行数

$Freq(t)$ は t 分間に修正された行の偏りを表す．受講者が無作為修正を行う時，エラーの原因と思われる箇所に対して，自分が持っている修正パターンを闇雲に適用する傾向があると考えられる．そのため，たとえば if 文の条件式のような 1 行に対して繰り返し修正を行うと考えられ，一定時間内の修正がわずかな行に集中している可能性がある（観点 P1）．提案手法ではソースコードの各行に対する修正回数から分散を求めることで，t 分間に行われた修正がどれだけ特定の行に集中しているかを分析する．無作為修正によって $Freq(t)$ が大きくなる傾向があると考えられる．

$Revs(t)$ は，t 分間に提出されたリビジョン数である．受講者が無作為修正を行う時，エラー箇所と思われる部分に自分の修正パターンを闇雲に適用する傾向があると考えられる．そのため一定時間内の修正・コンパイル回数が多く（観点 P2），無作為修正によって $Revs(t)$ が大きくなる傾向があると考えられる．

$DiffLine(s,t)$ は，s 行以下の修正が続いた回数から求められる．受講者が無作為修正を行う時，エラーが見つかると原因と思われる箇所をわずかに修正し，その正しさを考え

ることなくコンパイル・実行を繰り返す傾向があると考えられる．そのため，たとえば if 文の条件式だけのような 1 行の修正を繰り返す可能性がある（観点 P3）．提案手法では，t 分間に提出されたリビジョン群において s 行以下の修正が連続した回数の最大値と，$Revs(t)$ から $DiffLine(s,t)$ を算出する．

$$DiffLine(s,t) = \frac{s \text{ 行以下の修正が続いた回数の最大値}}{Revs(t) - 1} \tag{4.1}$$

無作為修正によって $DiffLine(s,t)$ が大きくなる傾向があると考えられる．なお，本研究では $s = 1$ とし，1 行以下の修正が続いた回数を用いて計算する．

$AveDiff(t)$ は，1 回あたりの平均修正行数である．受講者が無作為修正を行う時，エラーが見つかると複数個所が原因になっている可能性を考えず，ごく少量の修正を行い，コンパイル・実行を繰り返す傾向があると考えられる．そのため，1 回あたりの修正量は小さくなる可能性がある（観点 P3）．無作為修正によって $AveDiff(t)$ が小さくなる傾向があると考えられる．

4.2 算出手順

各メトリクスは以下 (1) ～ (5) の手順で算出する（図 4.1）．ここで，リビジョン k を R_k と表す．

(1) t 分間のスナップショット群の抽出
OJS に記録された各ソースコードの提出日時を元に，R_1 から t 分間に提出されたソースコードと，受講者 ID や評価結果 *score* を抽出する．

(2) $Revs(t)$ の算出
課題 ID と受講者 ID から同じ受講者が同じ課題に対して提出したソースコード数を求め，$Revs(t)$ を算出する．

(3) リビジョン間の修正箇所の検出
手順 (1) で抽出されたソースコードに対して，R_i と R_{i+1} 間の各行に対する修正箇所を diff コマンドと-W オプションを用いて検出し，差分テキストファイルとして生成する．差分テキストファイルは R_i と R_{i+1} 間の各行で，「追加」，「削除」，「変更」があれば，その行に各々「<」，「>」，「|」の記号が記され，修正がなければ何も記されない．この生成を各受講者に対して $i = 1 \sim (Revs(t) - 1)$ まで繰り返す．

(4) $DiffLine(1,t)$, $AveDiff(t)$ の算出

手順 (3) で生成した差分テキストファイルに対し，UNIX の wc コマンドを用いて各リビジョン間の修正行数を算出する．この結果を用いて，1 行以下の修正が続いた回数と $Revs(t)$ から $DiffLine(1,t)$ を算出する．同様に，各リビジョン間の修正行数と $Revs(t)$ から，各受講者に対する $AveDiff(t)$ を算出する．

(5) $Freq(t)$ の算出

手順 (3) で生成したテキストファイルに対し，ソースコードの各行に対して ID を割り当て，R_i と R_{i+1} 間の各行に対する修正箇所と R_{i+1} と R_{i+2} 間の各行に対する修正箇所をそれぞれ対応づける（図 4.2）．また，途中のリビジョンから追加された行の場合は新たに ID を割り当てる．これを $i = 1 \sim (Revs(t) - 1)$ まで繰り返し，全リビジョンにおいて修正した行 ID とその修正回数を求める．行 ID の集合 $ID = \{id_1, id_2, \cdots, id_k, \cdots, id_N\}$ に対して，各 ID に対応する行の修正回数を用いて分散を求め，$Freq(t)$ として算出する．

$$Freq(t) = \frac{1}{N} \sum_{k=1}^{N} \left(\frac{c_k}{s} - \mu \right)^2 \tag{4.2}$$

ここで，c_k は id_k の修正回数を表す．s, μ はそれぞれ次式を用いて算出する．

$$s = \sum_{k=1}^{N} c_k \tag{4.3}$$

$$\mu = \frac{1}{N} \sum_{k=1}^{N} \frac{c_k}{s} \tag{4.4}$$

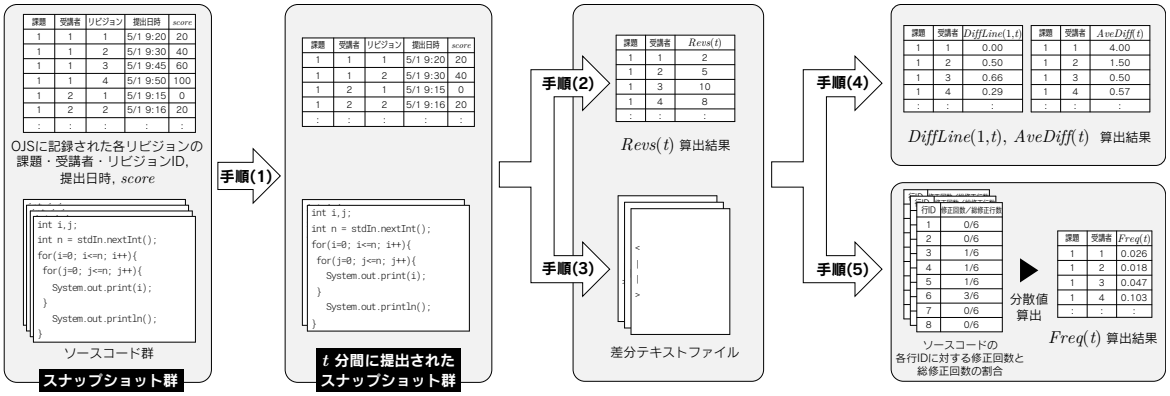


図 4.1: 各メトリクスの算出手順

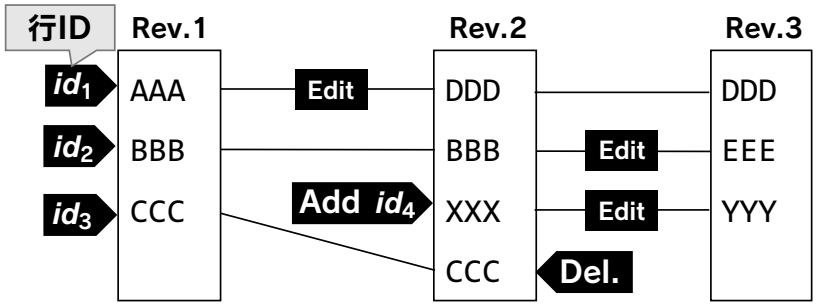


図 4.2: ID の割り当て

5. 実験

5.1 データセット

奈良工業高等専門学校の 2 年生が受講する授業であるプログラミング I で収集されたデータを分析対象とする．本講義は Java のプログラミング演習形式による，1 回あたり 90 分間の講義である．Learning Management System(LMS) の 1 つである Moodle と連携して OJS の機能を提供する CodeRunner を用いて演習の出題と回収，評価を行う．

CodeRunner は学生が提出したソースコードを課題 ID，受講者 ID，リビジョン番号，提出日時，*score* と共に記録する．なお，$Revs(t) = 1$ の場合，$Freq(t)$, $DiffLine(1,t)$, $AveDiff(t)$ の 3 つのメトリクスを算出できないため分析から除外する．本研究では，2017 年 6 月 21 日 ～ 9 月 1 日に収集された計 2,529 回（11 演習，述べ 364 人）の提出ソースコードのデータを対象に実験を行う．

5.2 実験 1

5.2.1 評価

提案したメトリクスの有用性を評価するために，各メトリクスと受講者の能力の関係を分析する．無作為修正を行う受講者は，演習内容を理解しないままソースコードの修正を繰り返すため，各リビジョンごとに OJS が算出する *score* が低いと考えられる．一方で，演習の内容やエラーの原因を理解して修正する受講者は各リビジョンの *score* が高いと考えられる．そこで，本研究では t 分間のスナップショット群における最新リビジョンの点数 $score(t)$ を求め，各メトリクスとの相関を求める．

本研究では演習中に無作為修正を識別することを目標としている．そのため，リビジョン 1 からの経過時間を表す t について複数の条件で分析を行うことが望ましい．そこで本研究では対象講義 1 コマ分の半分の時間である 45 分を t の最大とし，5 分刻みで各メト

リクスを算出し，比較する．

5.2.2 結果と考察

各メトリクスと $score(t)$ の相関係数および無相関検定の結果を表 5.1 に示す．以降では有意な相関が見られなかった $AveDiff(t)$ 以外の各メトリクスについて結果を説明する．

5.2.2.1 $Freq(t)$

$Freq(t)$ と $score(t)$ の相関係数は，$+0.159 \sim +0.199$ と弱い正の相関で，すべての t で有意な相関がみられた ($p < 0.01$)．この結果は，一定時間内の修正がわずかな行に集中している受講者の $score(t)$ が高いことを示しており，4.1 節で述べた観点 P1（同じ行を頻繁に修正する）とは反対の特徴を有している可能性を示唆する．また，t の値が異なっても相関係数はほとんど変化しなかった．

図 5.1 に $t = 10$ における $Freq(t)$ と $score(t)$ の散布図を示す．図から $Freq(t)$ と $score(t)$ の両方が高い受講者が多く見られた．また，このような受講者は $Revs(t)$ が小さい場合が多かった．これは，少数のリビジョンで特定の行を数回修正することで正答に到達していることを示している．

このような修正は OJS の採点方法が厳密であることが原因の 1 つとして起こっていると考えられる．OJS による採点はテストケースの出力部分との文字列のマッチングによって行われる．そのため，課題に対する回答のアルゴリズムが正しくても，空白の数やアルファベットの大文字・小文字，全角半角の違い，文字化けなどのフォーマットが異なる

表 5.1: $score(t)$ との相関

t	分析対象	$Freq(t)$		$Revs(t)$		$DiffLine(1,t)$		$AveDiff(t)$	
	人数	相関係数	p 値	相関係数	p 値	相関係数	p 値	相関係数	p 値
5	223	0.185	0.005**	-0.289	0.000**	-0.161	0.016*	-0.021	0.759
10	227	0.194	0.003**	-0.400	0.000**	-0.221	0.001**	-0.044	0.514
15	232	0.185	0.005**	-0.407	0.000**	-0.246	0.000**	-0.013	0.839
20	233	0.159	0.015*	-0.393	0.000**	-0.200	0.002**	0.003	0.967
25	233	0.171	0.009**	-0.331	0.000**	-0.211	0.001**	-0.007	0.915
30	233	0.186	0.004**	-0.380	0.000**	-0.183	0.005**	-0.035	0.596
35	234	0.194	0.003**	-0.405	0.000**	-0.169	0.010*	-0.126	0.055
40	234	0.199	0.002**	-0.355	0.000**	-0.168	0.010*	-0.111	0.090
45	234	0.194	0.003**	-0.366	0.000**	-0.135	0.039*	-0.118	0.071

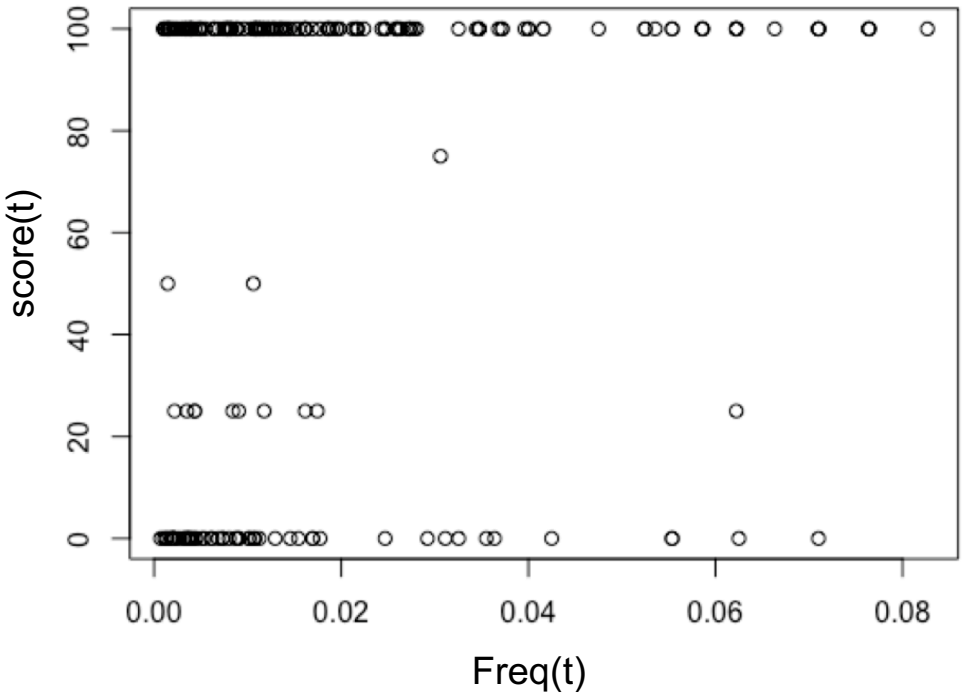


図 5.1: $Freq(t)$ と $score(t)$ の散布図 ($t = 10$)

と不正解と判定される．$Freq(t)$ と $score(t)$ の両方が高い受講者のリビジョン間のソースコード変化を見ると，正しい出力"`OutPut:0`"に対して誤って"`Output:0`", "`OutPut:␣0`"と出力している場合が複数見られた．また，これらの誤りは数回のリビジョンで修正され，正答に到達していた．$Freq(t)$ を観点 P1(同じ行を頻繁に修正する) のような振る舞いを検出するメトリクスとして使用するためには，同時にリビジョン数（$Revs(t)$）について考慮する必要があると考えられる．

5.2.2.2 $Revs(t)$

$Revs(t)$ と $score(t)$ の相関係数は，$-0.289 \sim -0.407$ と $Freq(t)$ と比べてやや強く，すべての t で有意な相関がみられた ($p < 0.01$)．この結果は，一定時間内の修正・コンパイル回数が多い受講者の $score(t)$ が低いことを示しており，4.1 節で述べた観点 P2（一定時間の修正・コンパイルの回数が多い）の特徴を有している可能性を示唆する．

また，t について 5 の時に最も相関が小さく，10 以上に大きな違いが見られなかった．

本研究で定義した $Revs(t)$ は t 分間に提出された累積リビジョン数のため，早い時間に 100 点に到達した受講者は t が大きくなったときにも $Revs(t)$ が変化せず，$Revs(t)$ と $score(t)$ の相関も一定のまま変化しない．そのため，t が増加したときに $score(t)$ が低いまま $Revs(t)$ のみが増大するような受講者がいても全体としての相関が変化しないことが考えられる．$Revs(t)$ を用いて無作為修正を検出するときには，すでに $score(t)$ が 100 点に到達している受講者を除外することが望ましいと考えられる．

表 5.2 に $Revs(t)$ と $score(t)$ のクロス集計表を示す．表は $score(t)$ が低く $Revs(t)$ が小さい受講者や，$score(t)$ が高く $Revs(t)$ が高い受講者が複数いることを表している．リビジョン間のソースコード変化を見ると，$score(t)$ が高く $Revs(t)$ が高い受講者は特定の行を繰り返し修正し，正答に到達していた（図 5.2）．これは，無作為な修正を繰り返しているうちに，たまたま正答に到達した可能性がある．

5.2.2.3 $DiffLine(1,t)$

$DiffLine(1,t)$ は，相関係数が $-0.135 \sim -0.246$ と弱かったが，すべての t で有意な相関がみられた $(p < 0.05)$．この結果は，1 行以下の修正が多い受講者の $score$ が低いことを示しており，4.1 節で述べた観点 P3（1 回あたりの修正量が小さい）の特徴を有している可能性を示唆する．$DiffLine(1,t)$ と $score(t)$ の相関係数が低い原因として，$Freq(t)$ や $Revs(t)$ と同様に，点数が高い受講者も細部の修正を施すことが考えられる．また，(4.1) 式より，$DiffLine(1,t)$ は修正回数 $Revs(t)$ に依存するメトリクスとして定義されている．そのため，1 行の修正を 1 回行ったときの影響が $Revs(t)$ が小さい受講者で大きくなる．$DiffLine(1,t)$ を観点 P3（1 回あたりの修正量が小さい）のような振る舞いを検出するメ

表 5.2: $Revs(t)$ の値ごとにおける $score(t)$ の受講者数 $(t = 10)$

$score(t)$	$Revs(t)$							
	2–3	4–5	6–7	8–9	10–11	12–13	14–15	16–
100	97	35	13	10	4	2	0	0
75	1	0	0	0	0	0	0	0
50	0	1	0	1	0	0	0	0
25	0	2	1	4	2	0	0	1
0	18	8	7	10	3	7	0	0

図 5.2: 無作為な修正の繰り返しで正答に到達した例

トリクスとして使用するためには，同時にリビジョン数（$Revs(t)$）について考慮する必要があると考えられる．

5.3 実験２

5.3.1 評価

実験１より，$Freq(t)$ や $DiffLine(1,t)$ は $Revs(t)$ の大小によって異なる受講者の様子を表している可能性がある．そのため，$Revs(t)$ の大小でグループ分けしたうえで，$Revs(t)$ の大きいグループに対して $Freq(t), DiffLine(1,t)$ で識別する必要があった．そこで，受講者を $Freq(t)$ や $Revs(t), DiffLine(1,t)$ の大小で 2 群に分け，各群の $score(t)$ に差があるか検証する．本論文では，$Freq(t)$ と $Revs(t)$ の検証結果のみ述べる．また，$Revs(t)$ が中央値以上の受講者群を R_H，中央値より小さい受講者群を R_L とする．同様に，$Freq(t)$ が中央値以上の受講者群を F_H，中央値より小さい受講者群を F_L とする．

5.3.2 結果

$t = 10$ における各群の $score(t)$ の平均値と検定（Welch の t 検定）の結果を図 5.3 に示す．結果は $score(t)$ の平均は F_L より F_H が，R_L より R_H が有意に高い．また，各群の平均値について $R_H \cap F_L$ より $R_L \cap F_L$ が有意に高く，$R_H \cap F_H$ より $R_L \cap F_H$ が有意に高

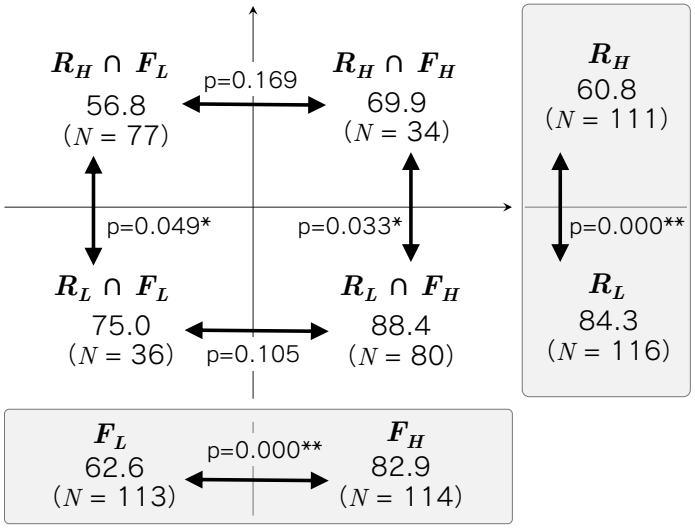


図 5.3: 各群の平均 $score(t)$

い．$R_H \cap F_L$ と $R_H \cap F_H$，および $R_L \cap F_L$ と $R_L \cap F_H$ の間には平均値の差が見られたものの，有意差は見られなかった．

F_H に属する受講者は修正行が偏っているため，演習中にエラーが出た時に，受講者がソースコードを修正すべき箇所は掴んでいたことが考えられる．しかし，F_H 群の中でも $Revs(t)$ の多い受講者 $(R_H \cap F_H)$ は特定の行に対して戦略が見られない修正やコンパイルエラーを含むような修正を連続して提出している様子が修正履歴から確認できた（図 5.4）．一方で，$Revs(t)$ の少ない受講者 $(R_L \cap F_H)$ は特定の行に対して 1 回もしくは 2 回の（主にフォーマットに関する）修正で正答に到達していた．これの受講者はいずれも $Freq(t)$ が高くなるが，$Revs(t)$ と組み合わせる事で，適切に弁別できると思われる．

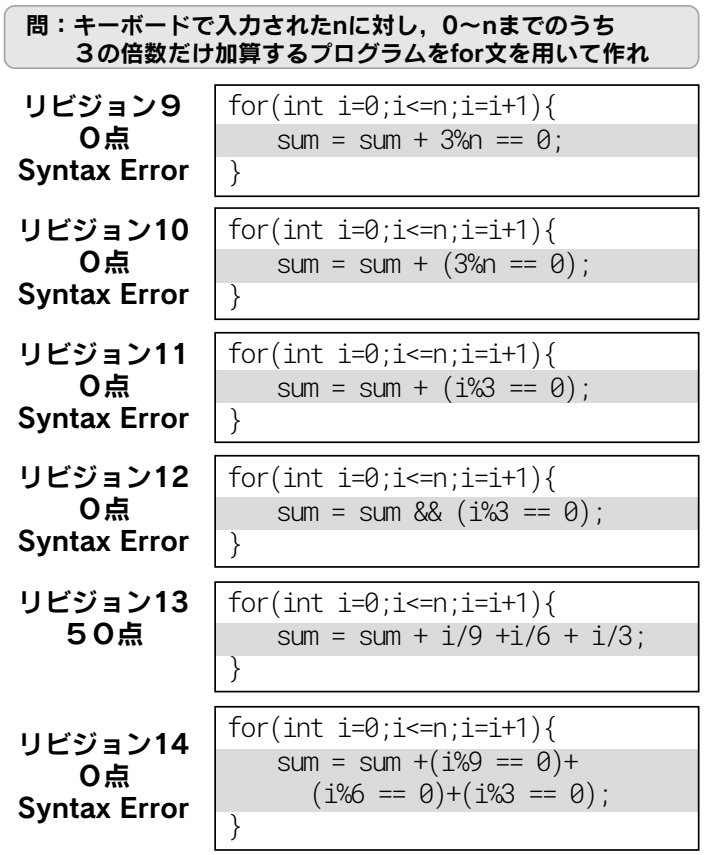


図 5.4: 戦略が見られない修正やコンパイルエラーを含むような修正

5.4 実験３

5.4.1 評価

無作為修正によって長時間正答に辿り着かない受講者を検出するために，OJS が算出する点数をもとに受講者をグループ分けし，各グループで $Revs(t)$ と $Freq(t)$ の 2 つのメトリクス値に差があるか検証する．

OJS によるプログラミング講義では，100 点を得た受講者は，ソースコードを正しく修正する方法を知っているため，無作為修正を滅多にしないことが考えられる．部分的正答（1 ～ 99 点）を得た受講者は，それまでに提出したリビジョンから，ソースコードの修正するべき箇所は掴んでいることが考えられる．部分的正答を得た受講者のうち一部の受講者は，部分的正答を得たとき誤答の原因を考えて，数回の修正で正答にたどり着くことが考えられる．一方で，他の一部の受講者は誤答の原因を考えずに無作為修正をして正答にたどり着かないことが考えられる．そのため，彼らの一部は無作為修正をすることが考えられる．また，0 点が続く受講者はソースコードの直すべき箇所さえ掴んでいないことが考えられる．

そこで，受講者をリビジョン 1 の点数 $score_1$ が 0 点と 1 ～ 99 点の 2 グループに分ける（$score_1 = 100$ の受講者はあらかじめ分析から除外している）．次に，受講者を $score(t)$ が 0 点，1 ～ 99 点，100 点の 3 グループに分ける．$score_1$ と $score(t)$ の点数の変化に着目し，図 5.5 に示す 6 通りの遷移をもとに受講者をグループ分けする（例えば，$score_1 = 0$, $score(t) = 50$ の受講者は $S_{(0,mid)}$ のグループに属する）．リビジョン 1 で 100 点でなかったときに，その受講者は t 分間でどのような修正を行いその結果が $score(t)$ としてどう反映されたかを見る．各受講者群において $Freq(t)$ と $Revs(t)$ に差があるかどうかを検証する．また，本実験では，$score_1$ が 1 ～ 99 点である受講者が少なかったため，$S_{(mid,0)}$，$S_{(mid,mid)}$，$S_{(mid,100)}$ のいずれかに属する受講者は分析対象から除外する．

実験 3 においても，実験 1 と同様にリビジョン 1 からの経過時間を表す t について複数の条件で分析を行うことが望ましい．そこで本実験では 40 分を t の最大とし，10 分刻みで各メトリクス・点数を算出したものを分析に用いる．

また，実験 3 では $t = 10, 20, 30, 40$ における $Freq(t)$，$Revs(t)$，$score(t)$ を用いるため，$t = 10$ で $Revs(t) \geq 2$ の受講者のみを分析対象とし，計 1,474 回（11 演習，述べ 227 人）の提出ソースコードのデータを対象に実験を行う．

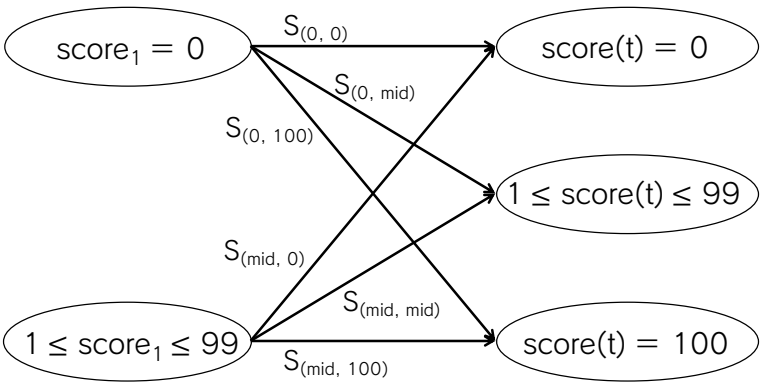


図 5.5: 受講者のグループ分け

5.4.2 結果と考察

$t = 10, 20, 30, 40$ における各群の $Freq(t)$，$Revs(t)$ の平均値と検定（Welch の t 検定）の結果を図 5.6(a)，図 5.6(b) に示す．

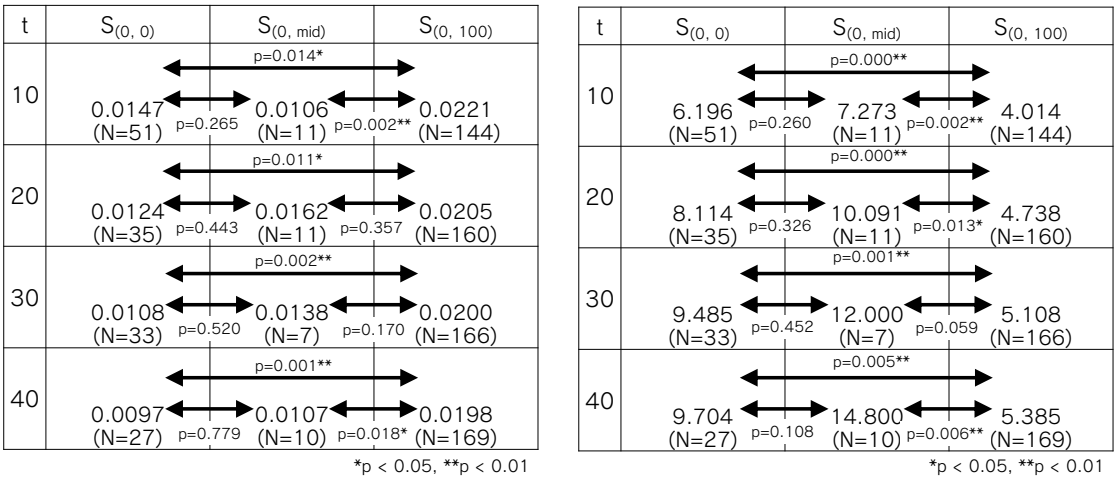
図 5.6(a) より，$Freq(t)$ は，$S_{(0,100)}$，$S_{(0,mid)}$，$S_{(0,0)}$ の順で高く，すべての t で $S_{(0,0)}$ より $S_{(0,100)}$ が有意に高い．また，$S_{(0,0)}$ と $S_{(0,100)}$ については t が大きくなるほど値が小さくなる．図 5.6(b) より，$Revs(t)$ は，$S_{(0,mid)}$，$S_{(0,0)}$，$S_{(0,100)}$ の順で高く，すべての t で $S_{(0,100)}$ より $S_{(0,0)}$ が有意に高い．また，すべてのグループについて，t が大きくなるほど値が大きくなる．

結果より，$S_{(0,100)}$ 群の $Revs(t)$ は他の 2 群に比べて低く，$Freq(t)$ は他の 2 群より高い．これは，$S_{(0,100)}$ 群の受講者は少数のリビジョンで特定の行を数回修正することで正答に到達していることを示しており，演習の内容やエラーの原因を理解して，修正箇所を正しく修正できていると考えられる．このような修正は条件式の簡単なミスや，OJS の採点方法が厳密であることが理由と考えられる．OJS による採点はテストケースの出力部分との文字列のマッチングによって行われる．そのため，課題に対する回答のアルゴリズムが正しくても，空白の数やアルファベットの大文字・小文字などのフォーマットが異なると不正解と判定される．$S_{(0,100)}$ 群に属する受講者の修正履歴を見ると，例えば正しい出力"`OutPut:0`"に対して誤って"`Output:0`", "`OutPut:␣0`"と出力している場合が複数見られた．また，これらの誤りは数回のリビジョンで修正され，正答に到達していた．

$S_{(0,0)}$ 群の $Revs(t)$ は $S_{(0,100)}$ 群に比べて有意に高く，$Freq(t)$ は他の 2 群より低い．これは，$S_{(0,0)}$ 群の受講者は複数回のリビジョンで広範囲に修正しており，正答に到達できていないことを示している．$S_{(0,0)}$ 群に属する受講者の修正履歴を見ると，複数の行に対して戦略が見られない修正やコンパイルエラーを含む修正を連続しており，修正戦略が把握できていない様子であった（図 5.7(a)）．

$S_{(0,mid)}$ 群の $Revs(t)$ は他の 2 群に比べて高く，$Freq(t)$ は $S_{(0,0)}$ より高い．この結果から，$S_{(0,mid)}$ 群の受講者は $S_{(0,0)}$ 群の受講者に比べ，多くのリビジョンをかけて特定の行を修正することで部分的に正答に到達していることを示しており，本研究で定義した無作為修正を行っていることが考えられる．0 点が続く状況から，あるリビジョンで 0 点でない評価を得たとき，受講者はそれまでのリビジョンで修正した箇所が修正すべき箇所だと捉え，その箇所を無作為に修正することが考えられる（図 5.7(b)）．

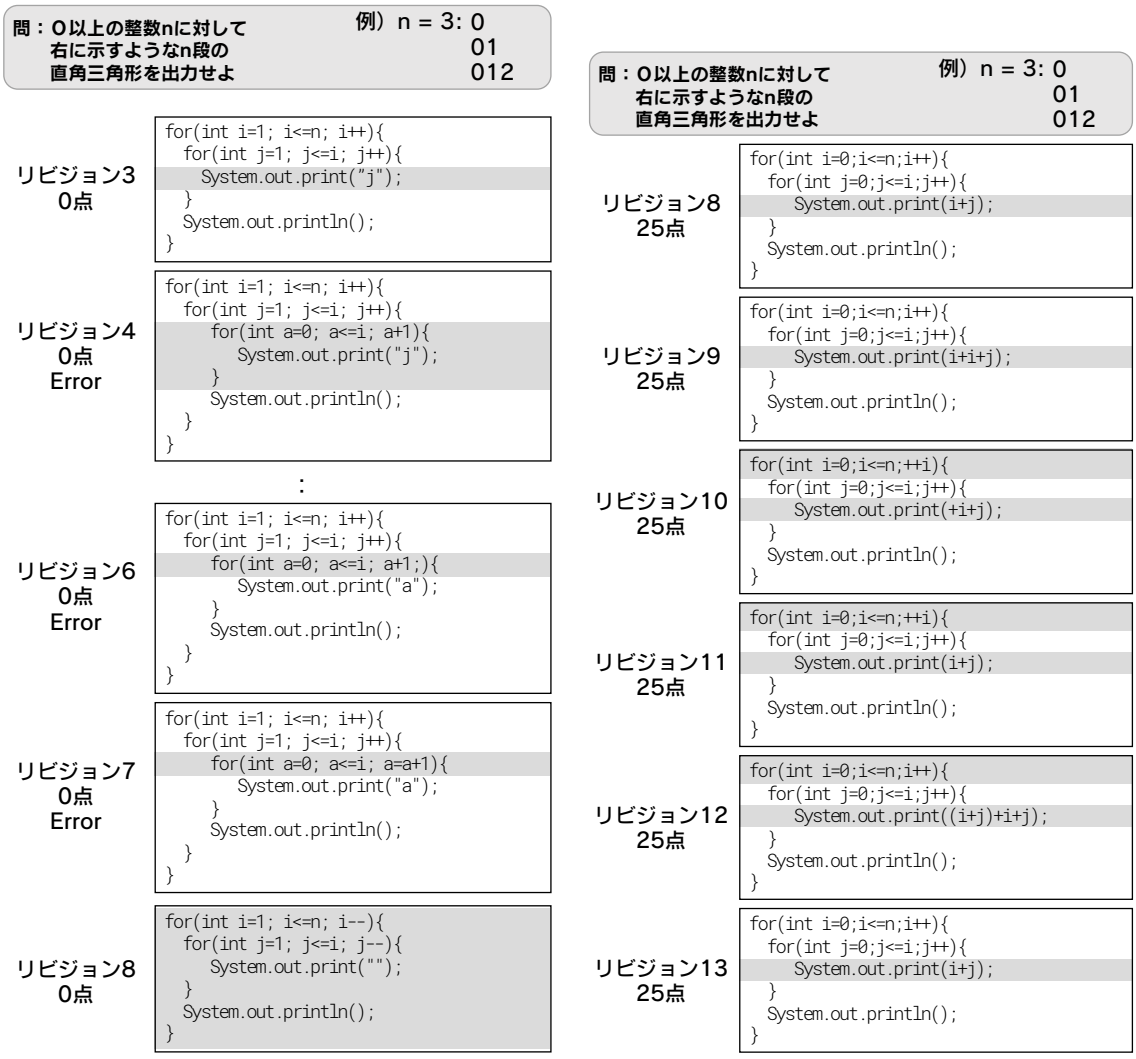
t が大きくなると，100 点を取っていない一部の受講者はソースコードを正しく修正し，$S_{(0,100)}$ 群に辿り着くため，t が大きくなると $S_{(0,0)}$ 群の受講者数が減少している．t が大きい時の $S_{(0,0)}$ 群の受講者は，長時間点数の低い受講者であるといえる．また，$S_{(0,0)}$ 群に属する受講者と $S_{(0,100)}$ 群に属する受講者の $Freq(t)$，$Revs(t)$ の差がより顕著になっている．t が大きい時に $S_{(0,0)}$ 群の $Freq(t)$ が小さく，$Revs(t)$ が大きいという結果は，長時間点数の低い受講者は多くのリビジョンをかけて，ソースコードをより広い範囲で修



(a) 平均 $Freq(t)$　　(b) 平均 $Revs(t)$

図 5.6: 各群の平均 $Freq(t)$ と平均 $Revs(t)$

正していることを示しており，彼らは誤答の原因を理解できていないことが考えられる．この結果は，$Freq(t)$ と $Revs(t)$ の 2 つを用いて長時間点数の低い受講者の編集履歴に基づいた各受講者の将来の点数予測に有用であることが示唆される．また，t が大きくなるにつれて，$S_{(0,mid)}$ 群に属する受講者の $Revs(t)$ の増え方が $S_{(0,0)}$ 群に比べてより急激になっている．しかし，$S_{(0,mid)}$ 群の人数が少ないため，この傾向については今後の課題としたい．



(a) $S_{(0,0)}$ 群の受講者　　(b) $S_{(0,mid)}$ 群の受講者

図 5.7: $S_{(0,0)}$ 群，$S_{(0,mid)}$ 群の受講者の編集履歴

6. 結論

本研究では，演習の内容を理解しないままソースコードの修正を繰り返す「無作為修正」を検出することを目的に，受講者が提出したソースコードのスナップショットから無作為修正を識別するメトリクスを提案し，その有用性を検証した．実験の結果，修正行の偏りを表す $Freq(t)$ と t 分間のリビジョン数を表す $Revs(t)$ の両方が高い受講者の点数が低く，また，その編集履歴から無作為修正である事が示唆された．

本研究の結果から，$Freq(t)$ と $Revs(t)$ を組み合わせることで，無作為な修正を繰り返し正答にたどり着かない受講者を把握し，指導することで教育効果を高められると考えられる．OJS にメトリクスを動的に算出・表示する機能を追加することで，支援が不要な受講者の作業を妨げることなく無作為修正に対して優先的に支援が可能になる．

今後の課題として，メトリクスの時系列分析が挙げられる．0 点が続く状況から，あるリビジョンで 0 点でない評価を得たとき，受講者はそのリビジョンで修正した箇所が修正すべき箇所だと捉え，その箇所を無作為に修正することが考えられた．これは無作為修正を行う受講者であっても，常に無作為な修正を行っているわけではなく，修正箇所の特定とその修正を交互に行っていると考えられる．今後，0 点が続く状況から，あるリビジョンで 0 点でない評価を得たときのメトリクス値を時系列分析する事で，受講者が無作為修正を行う瞬間を検出できると期待される．

本研究の発展として，受講者の特性を表すメトリクスを特定することで機械学習を用いた無作為修正の識別が可能になると思われる．受講者の特性を自動で識別することができれば，e-Learning コンテンツのみによる講義のような，教員のいない講義に対しても適用できる．また，本研究の分析では最初のリビジョンを提出してからの経過時間 t の時点におけるメトリクスとその時点の点数について分析を行っている．t の時点におけるメトリクスと，t から一定時間経過した（たとえば $t + 30$）時点における点数やメトリクスの関係を見ることで，長時間正答にたどり着かない受講者をより早い時点で検出することは興味深い発展の 1 つと考えられる．

謝辞

本研究を進めるにあたり，多くの方々に御指導，御協力を賜りました．ここに謝意を添えて御名前を記させて頂きます．

指導教員である上野 秀剛准教授には御多忙の中，研究ミーティング，論文の添削，研究発表のリハーサルなどについて多くの御指導と御助言を頂きました．また，日常生活についても御配慮，御助言を頂きました．心より感謝申し上げます．

内田 眞司准教授には，プログラミングⅠの講義データの利用許可を頂いただけでなく，研究に関して御指導と御助言を頂きました．心より感謝申し上げます．

山口 賢一准教授には，査読や研究力向上セミナーなどにおいて様々な視点から御助言を頂きました．心より感謝申し上げます．

松村 寿枝教授，市川 嘉裕助教には，研究力向上セミナーや最終発表会において様々な視点から御指摘と御助言を頂きました．心より感謝申し上げます．

上野研究室 OB/OG の皆様，後輩の皆様，専攻科生の同級生の皆様には，日々の生活において悩み相談や世間話に付き合ってくださり，おかげで有意義で楽しい 3 年間を過ごすことができました．心より感謝申し上げます．ありがとうございました．

参考文献

[1] R. Yoshihashi, D. Shimada and H. Iyatomi: Feasibility study on evaluation of audience's concentration in the classroom with deep convolutional neural networks, *2014 IEEE International Conference on Teaching, Assessment and Learning for Engineering (TALE)*, pp. 288–292 (online), 10.1109/TALE.2014.7062642 (2014).

[2] 島田大樹，彌冨　仁：畳み込みニューラルネットワークを使った授業映像中の聴講者の状態推定システムの構築と特徴量獲得に関する検討，知能と情報，Vol. 29, No. 1, pp. 517–526 (2017).

[3] B. Baradwaj and S. Pal: Mining Educational Data to Analyze Students' Performance, *International Journal of Advanced Computer Science and Applications*, Vol. 2, No. 6, pp. 63–69 (2011).

[4] S. Pal: Analysis and Mining of Educational Data for Predicting the Performance of Students, *International Journal of Electronics Communication and Computer Engineering*, Vol. 4, pp. 1560–1565 (2013).

[5] N. Tselios, A. Stoica, M. Maragoudakis, N. Avouris and V. Komis: Enhancing user support in open problem solving environments through Bayesian Network inference techniques, *Educational Technology & Society*, Vol. 9, No. 4, pp. 150–165 (2006).

[6] 植野真臣：e ラーニングにおけるデータマイニング (＜特集＞学習オブジェクト・学習データの活用と集約)，日本教育工学会論文誌，Vol. 31, No. 3, pp. 271–283（オンライン），10.15077/jjet.KJ00004964292 (2007).

[7] 加藤利康：授業支援システムにおける学習分析の展開，研究報告コンピュータと教育 (CE)， Vol. 2014, No. 23, pp. 1–7 (2014).

[8] 加藤利康，石川　孝ほか：プログラミング演習のための授業支援システムにおける学

習状況把握機能の実現，情報処理学会論文誌，Vol. 55, No. 8, pp. 1918–1930 (2014).

[9] P. Ihantola, A. Vihavainen, A. Ahadi, M. Butler, J. Börstler, S. H. Edwards, E. Isohanni, A. Korhonen, A. Petersen, K. Rivers et al.: Educational data mining and learning analytics in programming: Literature review and case studies, *Proceedings of the 2015 ITiCSE on Working Group Reports*, ACM, pp. 41–63 (2015).

[10] A. Dutt, M. A. Ismail and T. Herawan: A systematic review on educational data mining, *IEEE Access*, Vol. 5, pp. 15991–16005 (2017).

[11] 伏田享平，玉田春昭，井垣　宏，藤原賢二，吉田則裕：プログラミング演習における初学者を対象としたコーディング傾向の分析，電子情報通信学会技術研究報告. SS, ソフトウェアサイエンス， Vol. 111, No. 481, pp. 67–72 (2012).

[12] K. Fujiwara, K. Fushida, H. Tamada, H. Igaki and N. Yoshida: Why Novice Programmers Fall into a Pitfall?: Coding Pattern Analysis in Programming Exercise, *Fourth International Workshop on Empirical Software Engineering in Practice (IWESEP 2012)*, IEEE, pp. 46–51 (2012).

[13] 藤原賢二，上村恭平，井垣　宏，吉田則裕，伏田享平，玉田春昭，楠本真二，飯田　元：スナップショットを用いたプログラミング演習における行き詰まり箇所の特定，コンピュータソフトウェア， Vol. 35, No. 1, pp. 3–13（オンライン），10.11309/jssst.35.1_3 (2018).

[14] 槇原絵里奈，井垣　宏，吉田則裕，藤原賢二，飯田　元：プログラミング演習における探索的プログラミング行動の自動検出手法の提案，コンピュータソフトウェア，Vol. 35, No. 1, pp. 110–116（オンライン），10.11309/jssst.35.1_110 (2018).

[15] 藤原理也，田口　浩，島田幸廣，高田秀志，島川博光：ストリームデータによる学習者のプログラミング状況把握，第 18 回データ工学ワークショップ，pp. 1–6 (2007).

[16] M. Jadud: Methods and tools for exploring novice compilation behaviour, *ICER 2006 - Proceedings of the 2nd International Computing Education Research Workshop*, pp. 73–84 (2006).

[17] C. Watson, F. W.B. Li and J. L. Godwin: Predicting Performance in an Introductory Programming Course by Logging and Analyzing Student Programming Behavior, *2013 IEEE 13th International Conference on Advanced Learning Technologies, ICALT 2013*, pp. 319–323 (2013).

[18] A. Ahadi, R. Lister, H. Haapala and A. Vihavainen: Exploring Machine Learn-

ing Methods to Automatically Identify Students in Need of Assistance, *Proceedings of the Eleventh Annual International Conference on International Computing Education Research*, ICER '15, New York, NY, USA, ACM, pp. 121–130 (online), 10.1145/2787622.2787717 (2015).

[19] P. Blikstein: Using Learning Analytics to Assess Students' Behavior in Open-ended Programming Tasks, *Proceedings of the 1st International Conference on Learning Analytics and Knowledge*, LAK '11, New York, NY, USA, ACM, pp. 110–116 (online), 10.1145/2090116.2090132 (2011).

[20] 松永賢次：導入プログラミング教育におけるオンラインジャッジシステムの活用の試み，情報科学研究，Vol. 31, pp. 25–41 (2010).

[21] M. Iwamoto, S. Oshima and T. Nakashima: A Token-based Illicit Copy Detection Method Using Complexity for a Program Exercise, *2013 Eighth International Conference on Broadband and Wireless Computing, Communication and Applications*, pp. 575–580 (online), 10.1109/BWCCA.2013.100 (2013).

[22] 岩本　舞，中村真人，小島俊輔，中嶋卓雄：不正コピー検出手法を備えたオンラインジャッジシステムの開発，情報処理学会論文誌教育とコンピュータ（TCE），Vol. 1, No. 4, pp. 38–47 (2015).

[23] 古谷勇樹，林　真史，山本隆弘，長尾和彦ほか：RK-003 オンラインジャッジシステムと連携可能な Moodle プラグインの実装と比較 (K 分野: 教育工学・福祉工学・マルチメディア応用, 査読付き論文)，情報科学技術フォーラム講演論文集， Vol. 14, No. 3, pp. 89–94 (2015).

[24] 則行祐作，中川尊雄，畑　秀明，松本健一：オンラインジャッジの履歴を対象としたプログラマの成長分析，電子情報通信学会技術研究報告. SS, ソフトウェアサイエンス，Vol. IEICE-116, No. 277, pp. 97–101 (2016).

[25] V. Karavirta, A. Korhonen and L. Malmi: On the use of resubmissions in automatic assessment systems, *Computer science education*, Vol. 16, No. 3, pp. 229–240 (2006).

[26] V. Karavirta, A. Korhonen and L. Malmi: Different learners need different resubmission policies in automatic assessment systems, *Proceedings of the 5th Annual Finnish/Baltic Sea Conference on Computer Science Education*, pp. 95–102 (2005).

[27] S. Wasik, M. Antczak, J. Badura, A. Laskowski and T. Sternal: A Survey on Online Judge Systems and Their Applications, *ACM Comput. Surv.*, Vol. 51, No. 1, pp. 3:1–3:34 (online), 10.1145/3143560 (2018).

[28] C. Douce, D. Livingstone and J. Orwell: Automatic test-based assessment of programming: A review, *ACM Journal of Educational Resources in Computing*, Vol. 5, No. 3, p. 4 (2005).

[29] P. Ihantola, T. Ahoniemi, V. Karavirta and O. Seppälä: Review of Recent Systems for Automatic Assessment of Programming Assignments, *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*, Koli Calling '10, New York, NY, USA, ACM, pp. 86–93 (online), 10.1145/1930464.1930480 (2010).

[30] A. Kurnia, A. Lim and B. Cheang: Online judge, *Computers & Education*, Vol. 36, No. 4, pp. 299–315 (2001).

[31] B. Cheang, A. Kurnia, A. Lim and W.-C. Oon: On automated grading of programming assignments in an academic institution, *Computers & Education*, Vol. 41, No. 2, pp. 121–131 (2003).

[32] M. Ben-Ari: Constructivism in computer science education, *Journal of Computers in Mathematics and Science Teaching*, Vol. 20, No. 1, pp. 45–73 (2001).

[33] P. Guerreiro and K. Georgouli: Combating anonymousness in populous CS1 and CS2 courses, *ACM SIGCSE Bulletin*, Vol. 38, No. 3, ACM, pp. 8–12 (2006).