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

令和６年度 専攻科特別研究論文

差分構文木を用いたプログラミング授業受講者のコーディング特徴の自動抽出

Difference Syntax Trees for Characterising Student in Programming Course

指導教員名　上野 秀剛 准教授
論文提出者名　青木 晃汰

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

差分構文木を用いたプログラミング授業受講者のコーディング特徴の自動抽出

Difference Syntax Trees for Characterising Student in Programming Course

青木 晃汰
Aoki Kouta

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: The Online Judge System (OJS) is well-used in programming courses at universities or for self-learning. The system compiles and executes a set of source codes submitted in response to an assignment and automatically grades them by comparing the output results and expectations. In university programming courses, especially courses for beginners, students repeatedly modify and submit source code until they receive a 100-point score. The OJS stores every source code until each student receives 100 points on an assignment. The differences through the first to last submissions contain helpful information to estimate students' understanding of syntax or learning units in the course. In this study, the authors propose difference flow, a series of syntax trees extracted from the differences between the final submission and every previous one. The difference flow contains the node where the difference with the 100-point source code and each parent node; hence, its features (such as the count of each syntax node throughout the flow) may indicate the students' understanding.

Keywords: Programming education, Difference flow, Syntax tree;

関連業績リスト

1. 青木晃汰, 上野秀剛, “差分構文木によるプログラミング授業受講者のコーディング特徴分類,” 第 22 回情報科学技術フォーラム（FIT2023）,FIT2023 講演論文集,2023.

2. 青木晃汰, 上野秀剛, “差分構文木を用いたプログラミング授業受講者のコーディング特徴の自動抽出,” ソフトウェアエンジニアリングシンポジウム 2024 (SES2024),2024.

3. 青木晃汰, 上野秀剛, “ Difference Syntax Trees for Characterising Student in Programming Course,” The Asia-Pacific Software Engineering Conference (APSEC2024),2024.

目次

図目次

表目次

1. はじめに

大学等のプログラミング教育では，受講者が与えられた課題に対してソースコードを作成し提出を行うプログラミング演習と呼ばれる授業が開講されている [1]．講義に用いられるシステムの 1 つに Online Judge System (OJS) がある．OJS は教員が提示する課題に対して，受講者が課題を解決するソースコードを作成し提出すると，自動でコンパイルと実行を行う．その後，教員が用意したテストケースによって採点され，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，実行時エラーがフィードバックとして提示される．受講者は採点結果をもとにソースコードの修正と提出を繰り返し，*score* が満点である 100 点となるソースコードを実装することを目指す．講義を受講する初学者の一部はソースコードが期待と異なる動作をするときにその原因が分からず，1 つの不具合に対して繰り返し修正を行う．そのため，同じ課題に対して提出した一連のソースコードの差分は，特定の構文要素や learning unit に対する未理解の有無の識別に有用である．

本論文では OJS を用いたプログラミング講義の課題を対象として，受講者が $score = 100$ を取ったソースコードとそれ以前のソースコード間の差分構文木を自動的に抽出するツールを提案する．差分構文木は差分箇所が属する AST で表され，ある 1 つの編集を演算式やブロック，メソッドなど異なる粒度に対する編集として表現することで編集行動の分析を容易にする．例えば，ある 3 回の連続した差分 1) i=i+1, 2) print(i), 3) i<10 が同じ for ブロックに属する行に対する編集として表現されるため，受講者が困難を感じているブロックや構文要素を検出し，それに沿ったサポートが可能になる．本論文では最終提出とそれまでの各提出間の差分構文木の列を差分フローと定義する．本論文では，2 つのソースコードを入力すると任意の形式で構文木差分を出力するツールである Gumtree [2] を用いて XML 形式で差分を出力し，そこに含まれる構文情報を抽出するシステムを開発し，複数のソースコードを対象とした出力例を示す．実装したシステムで生成した差分フローから各受講者のコーディング特徴を求めることで，受講者自身が間違いや行動への理解を促すフィードバックの動的生成に有用と考えられる．

2. 関連研究

2.1 Online Judge System

これまでに，OJS をプログラミング講義の支援に用いる研究が複数行われている [3,4]. Hui らはプログラミングの習得に必要なプログラミング言語の理解，問題解決能力などを養うために独自のオンライン判定システムである YOJ（Youxue Online Judge）を構築した [3]．YOJ は提出されたソースコードのコンパイル・実行・正誤判定を行う Online Judge モジュール等の 7 つのモジュールで構成されており，複数のソースコードを並列処理によって実行・評価できる．Wenju らはテストケースを用いた採点では評価できないプログラムの類似性や可読性などを評価するための OJS フレームワークを提案した [4]. 提案フレームワークは受講者への個別フィードバック，コード品質チェック，コード類似性チェック，教育調整アドバイスの 4 つのモジュールから構成され，提出されたソースコードの総合的な解析レポートを自動的に作成し，受講者に提供する．また，採点結果を元にした統計情報を基に教師は教育スケジュールを調整できる．

これらの研究は OJS を利用したプログラミング講義という点において本研究と関連があるが，ソースコード間の差分を用いて自動的なフィードバックを生成するための研究は行われていない．本研究では提出された一連のソースコードの差分からコーディング特徴を抽出することで，受講者自身が各提出において何を誤っていたか，見落としていたか理解させるきっかけとなるフィードバックを自動生成することを目指す．

2.2 ソースコード間の差分

ソフトウェア開発などにおいて，コード間の変更履歴を理解することは重要である．これまでにオープンソースソフトウェアなどを対象にパターンマイニングやコードレビューに関する様々な研究が行われている [5–8].

ソースコード差分に含まれるデータを用いた分析を行う研究は Koyuncu らや Nguyen らによって行われている [5] [6]．Koyuncu らはプログラム修復の自動化のために，ソフトウェア開発におけるパッチで繰り返される修正パターンのマイニングを自動化する手法である”FixMiner”を開発した．”FixMiner”は AST レベルのコードの変更内容をキャプチャする編集スクリプトに特化したツリー構造を用いる．オープンソースプロジェクトから収集したパッチで”FixMiner”を評価した結果，変更情報を効率的に利用して正確なパターンをマイニングできることが示され，これをプログラムの自動修正システムに統合することで Defects4J ベンチマークの 26 個のバグを正しく修正することができた Nguyen らは 2,841 の Java プロジェクトからなる大規模なデータセットを収集し, 変更のリビジョンを分析することでソフトウェア進化におけるコード変更の学習とそれを推奨することは有益であると分かった [6]．藤本らは Gumtree を拡張し，ファイルを横断するコード片の移動を検出する手法を提案した [7]．8 個のオープンソースソフトウェア (OSS) に対して提案手法を用いた実験の結果，88,848 個のコミットの中から 89,418 個のファイルを横断する移動を検出し，ファイルを横断するコード片の移動やファイル名の特徴が得られた．また既存ツールを上回る数の移動を検出した．松本らは編集スクリプトの長さが課題である Gumtree を改良し，より短く理解しやすい編集スクリプトを生成する手法を提案した [8]．7 個の OSS を対象とした実験の結果，全てのプロジェクトで編集スクリプトが短くなった．また，14 人の被験者に対する実験の結果，提案手法によって差分理解に費やす時間が減ることを確認した．

ソースコード差分に関するこれらの研究は，ソースコード間の変更内容の分析という点やソースコード間の差分の理解支援という点で本研究と関連があるが，プログラミング講義の課題に対して受講者が提出したソースコードの差分から理解を促すための研究は行われていない．本研究では講義の課題に対して提出されたソースコード間の差分から構文情報を抽出し，受講者のコーディング特徴を分析することで受講者の学習補助に利用する．

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 OJS 上に提示された講義資料を元に課題を解く形式である．受講者は OJS 上に提示された課題に対応するソースコードを，自身が普段使用するエディタで作成して OJS に提出する．OJS は提出されたソースコードをプログラムの動作に必要な他のソースコードとともにコンパイル・実行し，あらかじめ用意された入力を与えたときの出力が，予期された出力と一致するか判定する．あらかじめ用意された入力と，入力に対応した正解となる出力の組をテストケースと呼ぶ．OJS は受講者が提出したプログラムに対して，テストケースとして用意された入力を渡し，出力される標準出力の内容がテストケースの出力と一致した数に応じて以下の式で点数 *score* を計算する．

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

受講者は提出したソースコード 1 組に対するフィードバックとして，各テストケースの正誤，*score*，コンパイルエラーの有無，実行時エラーの有無を得る．各受講者は *score* が 100 になるまでフィードバックを基にソースコードを修正，再提出する．*score* が 100 となるソースコードを提出することで課題が完了し，それまでに提出したすべてのソースコードが提出日時，*score* とともにリビジョンとして記録される．

3.2 抽象構文木 (AST)

抽象構文木 (AST: Abstract Syntax Tree) とはソースコードの構文情報を表現した木構造である．図 3.1 に (a)Java ソースコードと (b) ソースコードに対応する AST を示す．AST は順序木であり，各頂点がプログラムの構文上の 1 つの要素に対応した ID とラベ

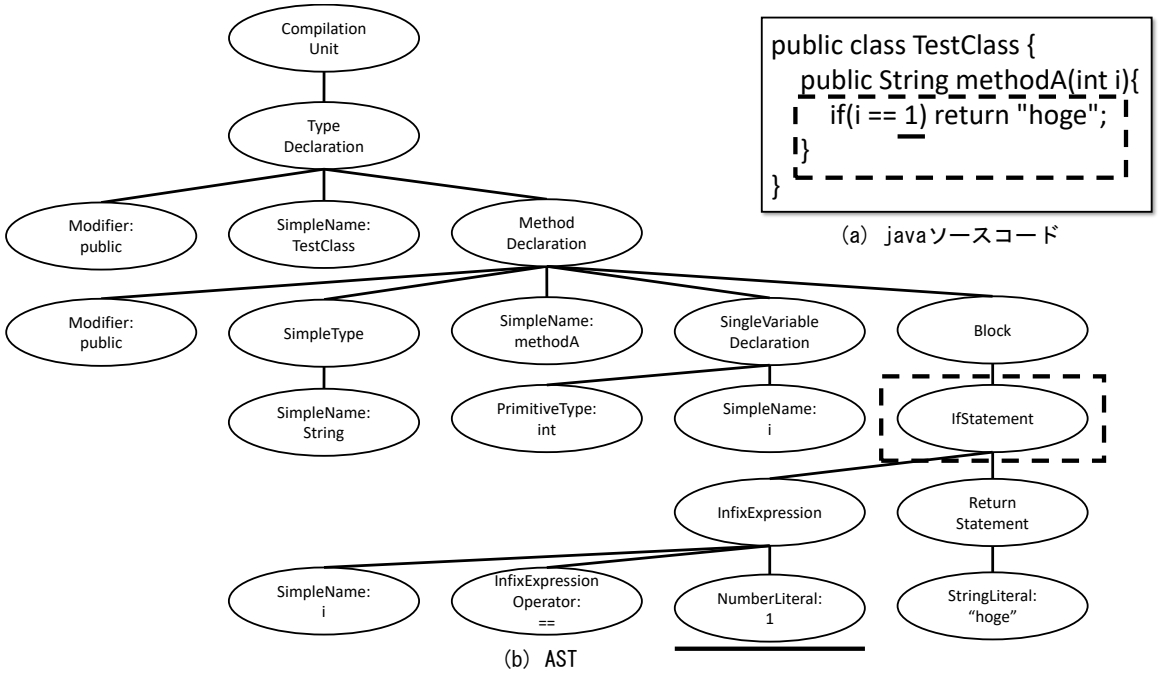

図 3.1　ソースコードと AST

ルを持つ．表 3.1 に構文情報の一部を示す．親頂点と枝で結ばれた子頂点は詳細情報を表す．例えば，図 1 の下線で示した頂点"NumberLiteral:1"は図 1(a) の 3 行目 7 文字目の要素に対応しており，ラベル NumberLiteral が数値定数，値 1 がその値が 1 であることを表す．また，図 1 の破線で示した頂点は図 1(a) の if 文全体に対応しており，ラベル IfStatement の構文情報が 2 つの子頂点 InfixExpression（if 文の条件式）および ReturnStatement（return 文）からなることを示している．

本研究の提案手法はソースコードから AST を求め，*score* が 100 となったソースコード（最終版）と最終版以外のソースコード間で AST の差分を出力する．提出された個々のソースコードを最終版と比較することで，各提出における修正が必要な箇所や，編集箇所の分布が容易に理解できる．また，各差分に構文情報を付与することで，編集内容に score や要修正というラベルを付与したデータを用いた機械学習で頻繁に間違える構文要素や処理内容を学習し，フィードバックの生成に利用できると考えられる．

表 3.1　タグと内容の対応表

構文情報	概要
CompilationUnit	対象ソースコード全体
TypeDeclaration	型宣言
Modifier	修飾子
MethodDeclaration	method 宣言
MethodInvocation	メソッド呼び出し
METHOD_INVOCATION_RECEIVER	メソッドを呼び出す対象（レシーバ）
METHOD_INVOCATION_ARGUMENTS	メソッド呼び出しの引数
SinpleName	単純名
QualifiedName	限定名
Block	method や Statement の範囲
ExpressionStatement	式文
forstatement	for 文
Ifstatement	if 文
WhileStatement	while 文
DoStatement	do 文
SwitchStatement	Switch 文
VariableDeclarationExpression	変数宣言式
InfixExpression	中置演算式
PostfixExpression	後置演算式
NumberLiteral	数値定数
StringLiteral	文字列リテラル

4. 差分フロー

4.1 概要

図 4.1 に差分フローの出力プロセスを示す．受講者が 1 つの課題で $score = 100$ を取得するまでに提出したソースコード $(V_1 \ldots V_N)$ に対して，Gumtree [2] を用いて 1) 各 Ver の構文木と，2)Ver.N とそれ以前の各 Ver$(V_1 \ldots V_{N-1})$ との差分情報として type (add, delete, move, modify), row, line, text を出力し，それらを差分フロー抽出ツールが統合して差分フローとして抽出する．

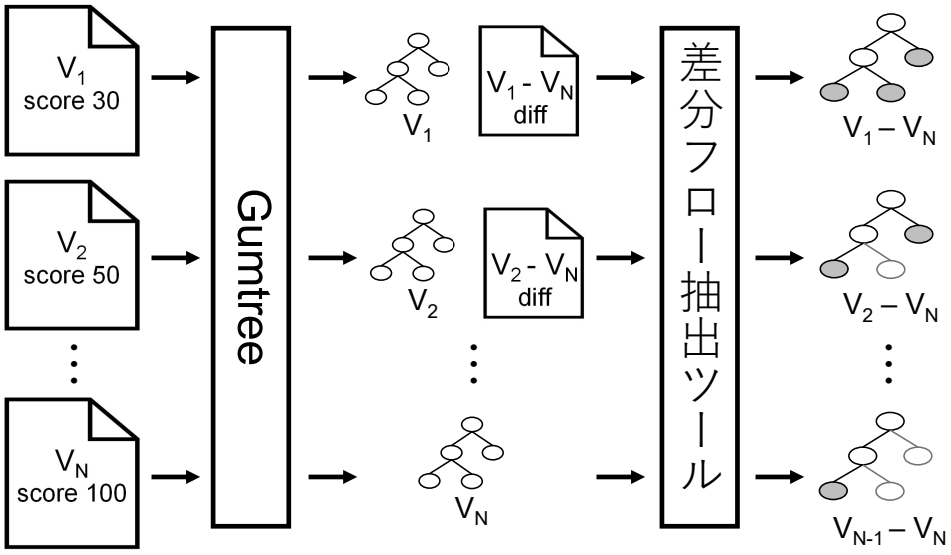

図 4.1　差分フローの出力プロセス

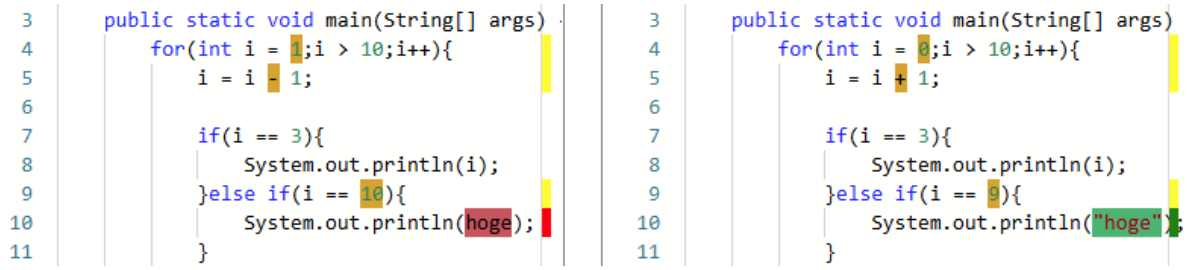

図 4.2　Gumtree を用いた最終提出との差分　提出 1 回目

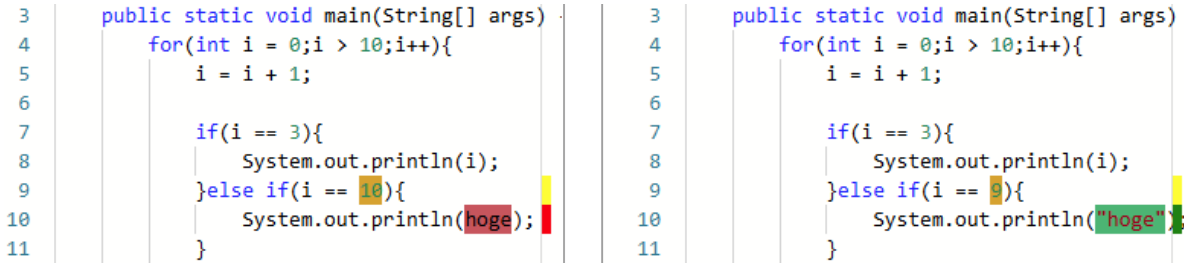

図 4.3　Gumtree を用いた最終提出との差分　提出 2 回目

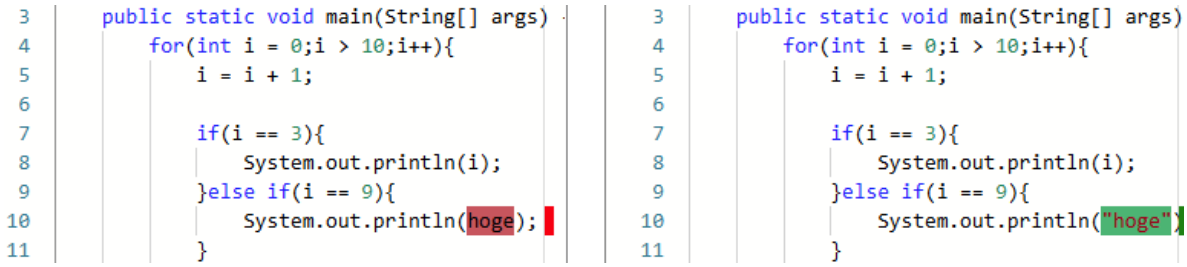

図 4.4　Gumtree を用いた最終提出との差分　提出 3 回目

図 4.2，図 4.3，図 4.4 に全 4 提出の課題から Gumtree を用いて出力した差分の例を示す．各図の右側が最終版のソースコード，左側がそれ以前の各 Ver のソースコードを示す．文字の背景色は緑が挿入，橙色が更新，桃色が移動，赤が削除を表す．提出 1 回目のソースコード（図 4.2 左側）は最終版と 4 箇所の差分があるが，2 回目（図 4.3 左側）で 2 箇所，3 回目（図 4.4 左側）で 1 箇所と各提出を経て徐々に差分箇所が減っている．その後，受講者は最終提出で表示文の文字列を修正して課題を完了した．

また，分析の際に用いる差分情報と木構造で出力したソースコードの XML の例を

```
<actions>
  <move-tree tree="InfixExpression [267,279]" parent="IfStatement [264,552]"
  <insert-node tree="Block [281,477]" parent="IfStatement [264,552]" at="1"/>
  <insert-node tree="Block [481,552]" parent="IfStatement [264,552]" at="2"/>
  <insert-node tree="IfStatement [286,469]" parent="Block [281,477]" at="0"/>
  <insert-node tree="ExpressionStatement [487,548]" parent="Block [481,552]"
  <move-tree tree="InfixExpression [284,317]" parent="IfStatement [286,469]"
  <insert-tree tree="Block [323,395]" parent="IfStatement [286,469]" at="1"/>
  <move-tree tree="Block [320,386]" parent="IfStatement [286,469]" at="2"/>
  <insert-node tree="MethodInvocation [487,547]" parent="ExpressionStatement
  <move-tree tree="METHOD_INVOCATION_RECEIVER [397,407]" parent="MethodInvoca
  <insert-node tree="SimpleName: print [498,503]" parent="MethodInvocation [4
  <insert-tree tree="METHOD_INVOCATION_ARGUMENTS [504,546]" parent="MethodInv
  <delete-node tree="INFIX_EXPRESSION_OPERATOR: || [280,282]"/>
```

図 4.5　差分情報.xml　一部抜粋

図 4.5，図 4.6 に示す．図 4.5 の差分情報は action タグの中に含まれており，各タグ内に構文情報とそれに対する編集の種類が記されている．図 4.6 の xml 形式に変換したソースコードはある構文の構成要素を子ノード（"children node"タグ）として記している．例えば for 文をこの形式で表現した場合，子ノードとして含まれるのは図 4.6 の"InfixExpression"の他に初期化式を表す"VariableDeclarationExpression"や後置演算式を表す"PostfixExpression"，ループ内容を表す"Block"などがある．

次に，図 4.7 に差分フローの一部を例として示す．例は 1) 一連の編集が同じ「for 文」ブロックに集中していること，2) 受講者は for 文の条件式に編集が数 Ver にわたって集中している様子が見られる．この差分フローの様子から受講者が for 文の条件式とループ回数の関係に対して理解不十分であること読み取れる．教師はこの情報を元に，該当の受講者に対して for 文の条件式とループ回数に特化した補足や，追加課題の提示を行う．2 バージョン間の行テキスト差分と比較すると，差分フローには変更された各ノードの親ノードが含まれている．そのため，差分フローは最終バージョンとの差分を用いることで，1) プログラム全体に対するどの箇所に変更が行われたのか把握しやすく，2) 同じブロックやメソッドの異なる行を対象とした編集を 1 まとめに扱える．

```
<?xml version="1.0" encoding="utf-8"?>
<root node="CompilationUnit [0,155]">
 <children node="nodeDeclaration [0,153]">
  <children node="MethodDeclaration [27,148]">
   <children node="Block [66,148]">
    <children node="ForStatement [71,144]">
     <children node="InfixExpression [85,91]">
      <children node="InfixExpressionOperator &lt;
       [87,88]" action="update" />
     </children>
    </children>
   </children>
  </children>
 </children>
</root>
```

図 4.6　ソースコード.xml　一部抜粋

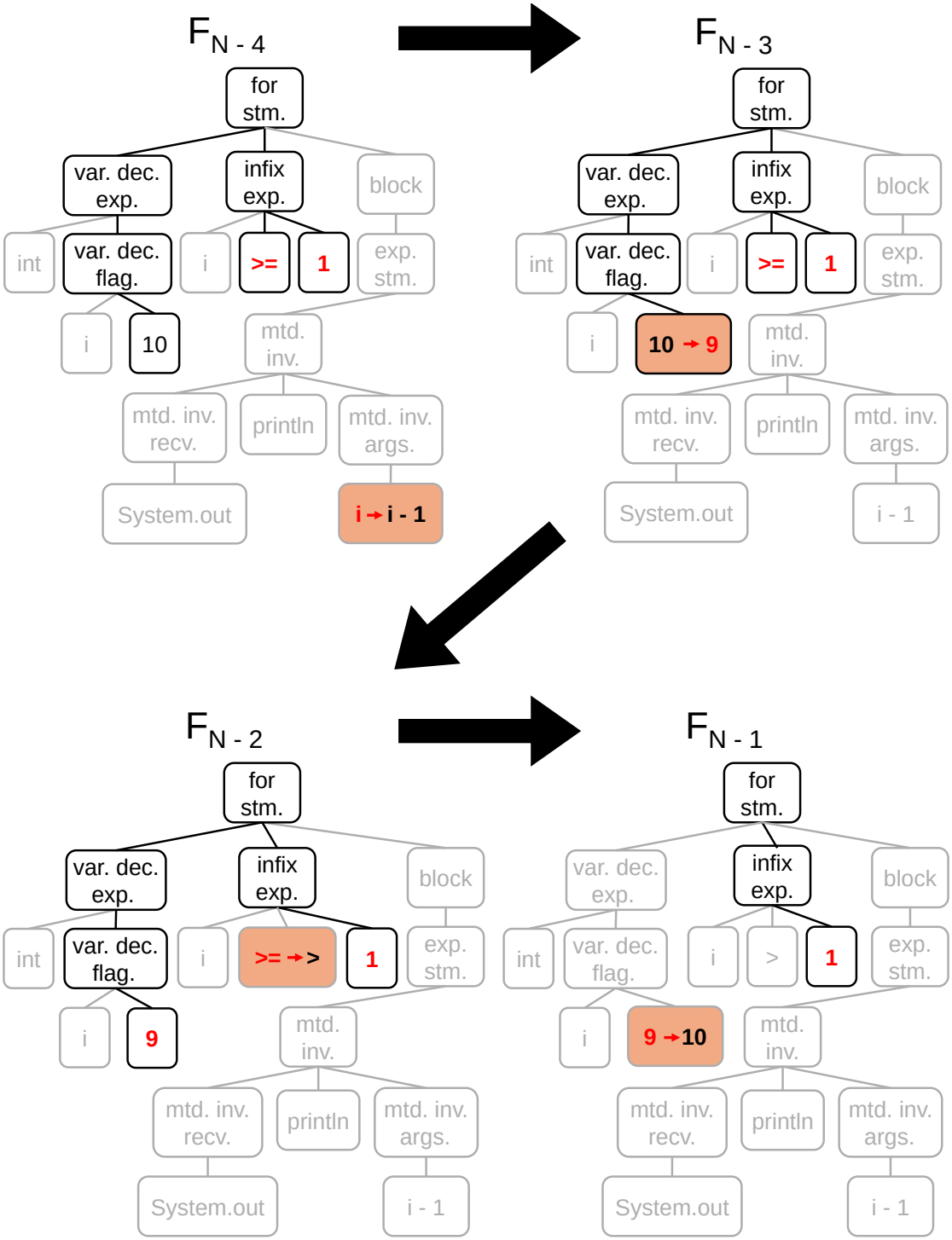
F N - 4
for stm.
var. dec. exp.
infix exp.
block
int
var. dec. flag.
i
>=
1
exp. stm.
i
10
mtd. inv.
mtd. inv. recv.
println
mtd. inv. args.
System.out
i → i - 1
F N - 3
for stm.
var. dec. exp.
infix exp.
block
int
var. dec. flag.
i
>=
1
exp. stm.
i
10 → 9
mtd. inv.
mtd. inv. recv.
println
mtd. inv. args.
System.out
i - 1
F N - 2
for stm.
var. dec. exp.
infix exp.
block
int
var. dec. flag.
i
>= → >
1
exp. stm.
i
9
mtd. inv.
mtd. inv. recv.
println
mtd. inv. args.
System.out
i - 1
F N - 1
for stm.
var. dec. exp.
infix exp.
block
int
var. dec. flag.
i
>
1
exp. stm.
i
9 → 10
mtd. inv.
mtd. inv. recv.
println
mtd. inv. args.
System.out
i - 1

図 4.7　差分フローの例

4.2 実装

受講者は OJS に掲示された講義資料と課題文を元にソースコードを OJS に提出する．提出したソースコードが題意を満たさずに 100 点未満の点数が付いた場合，受講者は修正したソースコードを再提出し，100 点になるまで繰り返す．OJS システムは提出されたソースコードをそれぞれ Gumtree [2] に入力し，Ver.i($i = 1, 2, ..., N - 1$) と Ver.N 間の差分情報とそれぞれのバージョンの AST を出力する．Gumtree は 2 つのソースコードを入力するとそれぞれを AST に変換した上で差分を出力する．差分情報は 2 つのソースコード間で行われた編集行動（挿入，更新，移動，削除）と構文情報，変数や文字，変更された範囲の座標（ソースコードの最初の文字から該当箇所までの文字数）が記されている．差分情報の座標をもとに各バージョンの AST から対応する編集箇所を抽出し，Ver.1 と Ver.N の差分とする．提案手法は上記の処理を最終バージョンを除いた全てのバージョンに対して実行し，その集合を差分フローとする．差分フロー抽出の処理は Python で作成し，抽出された差分フローは XML 形式で出力される．

4.3 差分フローの活用

差分フローは差分のあったノードに加え，親ノードの情報も XML 形式で出力するため，一連の編集履歴に共通するブロックやメソッド，文法要素を抽出，分析できる．また，差分フローに含まれる一連のノード情報は feature としても利用できる．各構文・ブロックに対する編集頻度や編集内容によるクラスタリングや機械学習によって，必要な支援の自動抽出や，learning unit に対する自動評価が可能になると期待される．ツールから得られる差分フローに含まれる各 Ver における構文情報の出現数をカウントすることで受講者の修正回数や編集が各 Ver でどの構文に集中しているかを数値的に見ることが出来る．カウントした数値を機械学習等に用いることによって，各構文の編集頻度や内容の関係性を見出すことが出来ると考えられる．また分析結果を用いて，OJS は受講者に述語に関する補習課題の提案などの適切なサポートを提供することができる．

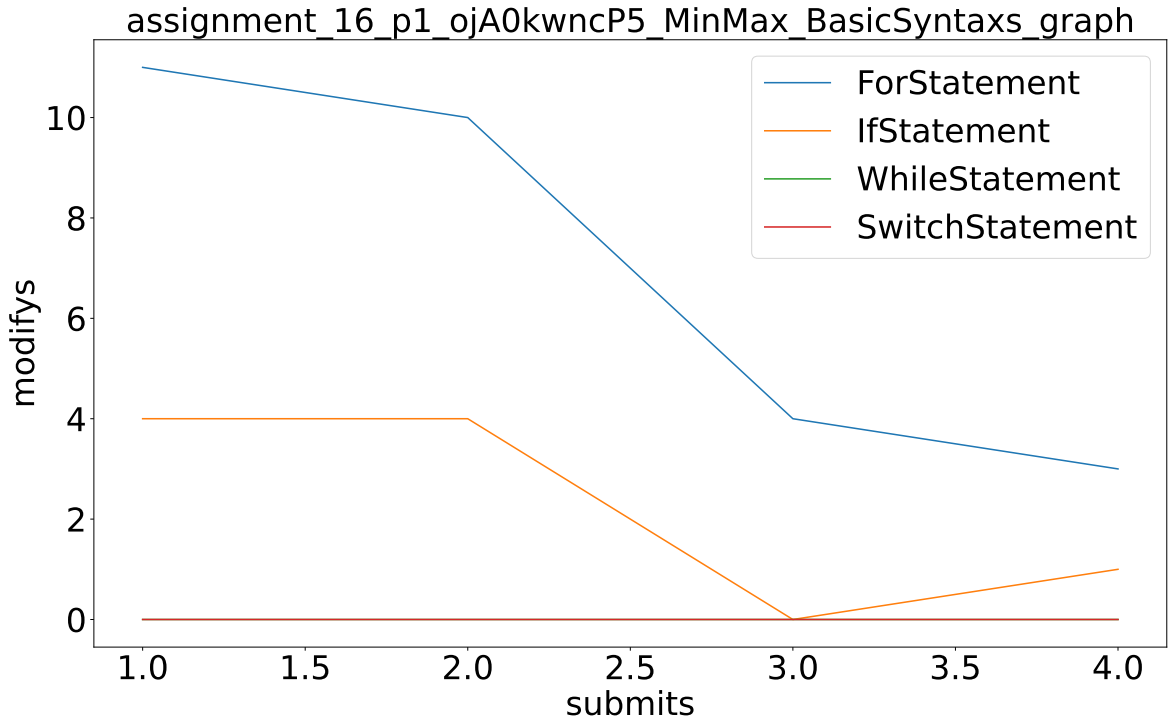

図 4.8　実装システムの出力例

4.3.1　実装システムの出力例

ある 1 人の受講者の取り組んだ 1 つの課題において提出されたソースコード群から差分フローを出力し，最終提出より前の各提出段階における差分から 4 つの基礎構文“ForStatement”（for 文）,“IfStatement”（if 文）,“WhileStatement”（while 文,do-while 文）,“SwitchStatement”（switch 文）の出現回数をカウントし，折れ線グラフにプロットした例を図 4.8 に示す．本システムは課題の java プロジェクトが複数のクラスファイルによって構成される場合，各クラスファイルごとに分けてグラフを作成する．折れ線グラフは横軸が提出バージョン，縦軸が差分量（編集回数）を表しており，各基礎構文の差分量の遷移から受講者が課題完了に至るまでの様子を視覚的に表現している．図 4.8 の例では初回の提出から最終提出までに"ForStatement"と"IfStatement"内で多くの編集があったことを示しており，最終提出に向けて徐々に差分が減って完答に近づいていく過程が分かる．

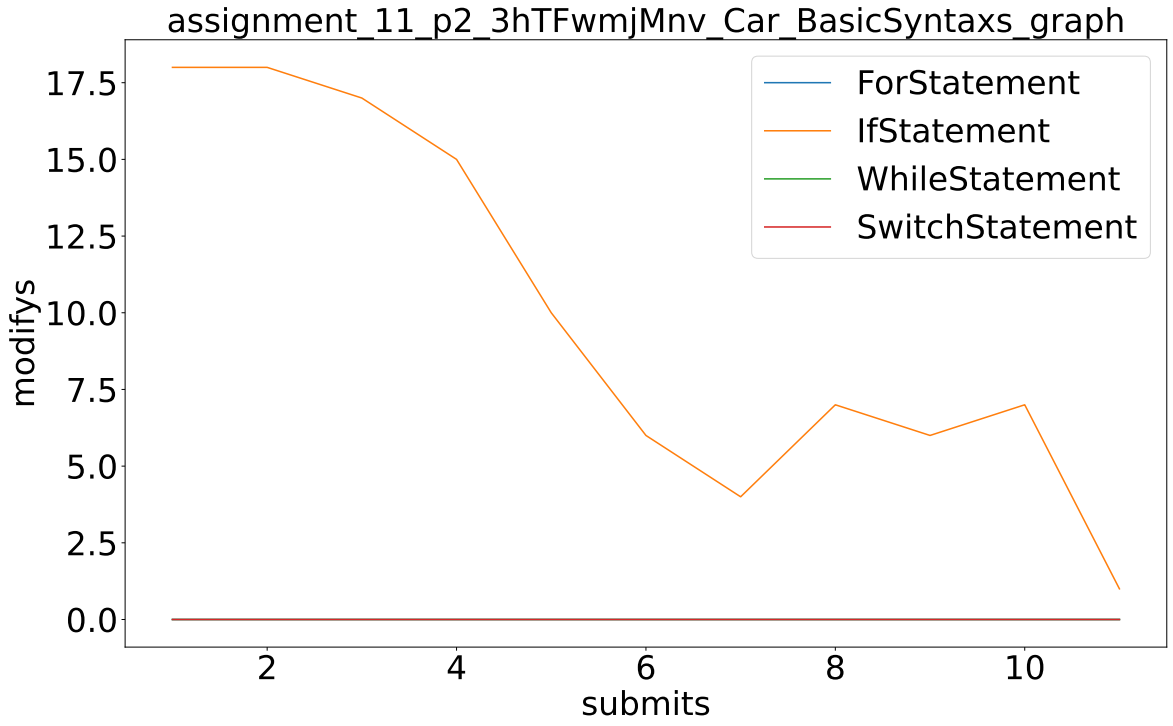

図 4.9　特定構文に対する編集の例

4.3.2　差分フローを用いた編集行動の理解

OJS を用いた講義の課題で提出されたソースコードを対象に，提案手法を用いて各構文の編集回数の推移を抽出し，編集行動の特徴を分析する事例を複数示す．分析に用いたデータは奈良高専 3 年生を対象としたプログラミングの講義「プログラミング Ⅱ」において，受講者が課題に対するソースコードの提出履歴である．提出履歴にはソースコードと提出日時，提出者のユーザー名や OJS が採点した課題の評価が含まれる．分析対象とするデータを取得する前に利用者全員に書面による説明を行い，同意した学生が提出したソースコードのみを分析に用いる．また，個人の特定に繋がりうる情報であるユーザー名に暗号化を施し匿名化を行う．

図 4.9 に受講者がある課題において特定の構文をよく編集した場合の編集回数の推移を示す．この受講者が最初に提出したソースコードは最終版までに if 構文に関する箇所が 18 回変更されており，提出全体を通して段階的に if 構文を編集しながら正答に近づいて

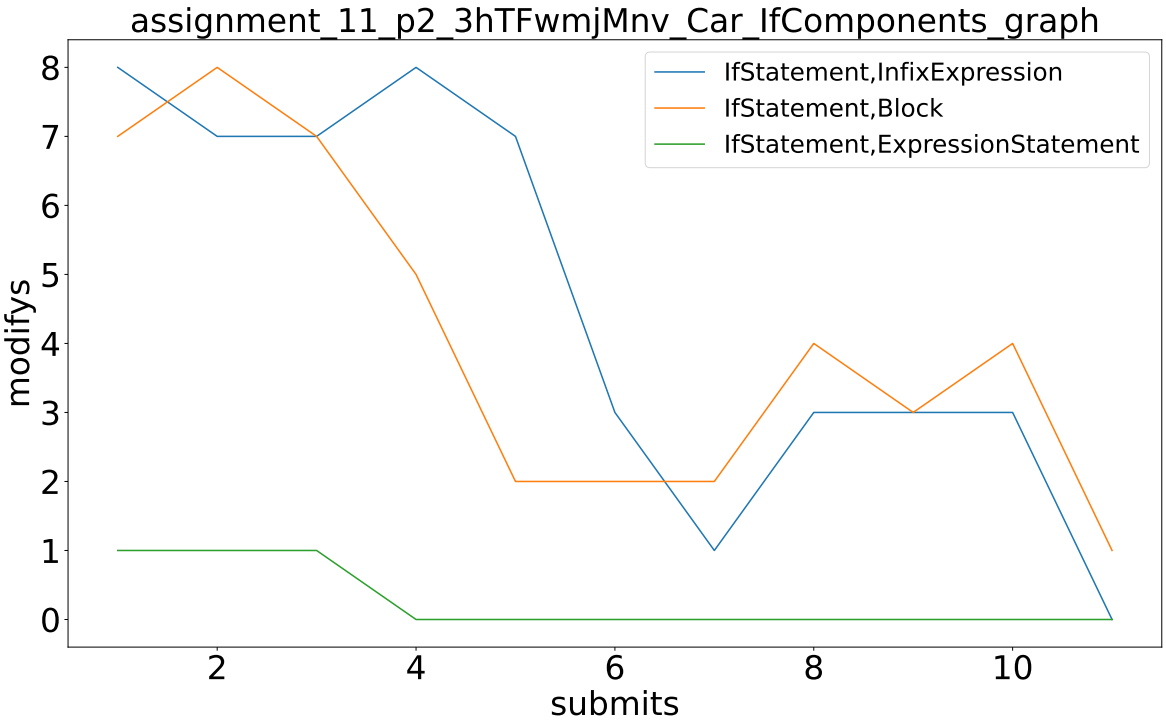

図 4.10　if 構文内要素に対する編集回数の推移

いることが確認できる．図 4.10 に if 構文内の異なる要素に対する編集として細分化した際の折れ線グラフを示す．この受講者は特に条件式（中置演算式：InfixExpression）と分岐後の処理（ブロック：Block, 式文：ExpressionStatement）を編集しており，ブロックを先に，条件式をその後に編集していることが分かる．図 4.9 と図 4.10 はいずれも差分フローを集計した結果であるが，修正フローが持つ修正箇所の構文木上の親要素である各構文の情報を分析目的に応じて集計することで異なる粒度に着目したときの編集の様子を分析することができる．ここで示した if 文の例と同様に，ループ構文において受講者の条件式とループ回数の関係に対する理解度や教師側が設定した課題テーマに関する構文に対する理解度，式文の簡素化や条件式の最適化の過程などに対しても同様に，異なる粒度での編集履歴を分析することで，個々の受講者のコーディング特徴に合わせたフィードバックが可能になる．例えば，図 4.9，4.10 に示した編集履歴が観察された場合，教員が if 構文や条件式などの該当箇所に対応する追加課題や補足資料を受講者に提示することで理解を促すことができる．また，各課題の差分フローから得られるデータを集め，機械学習を行

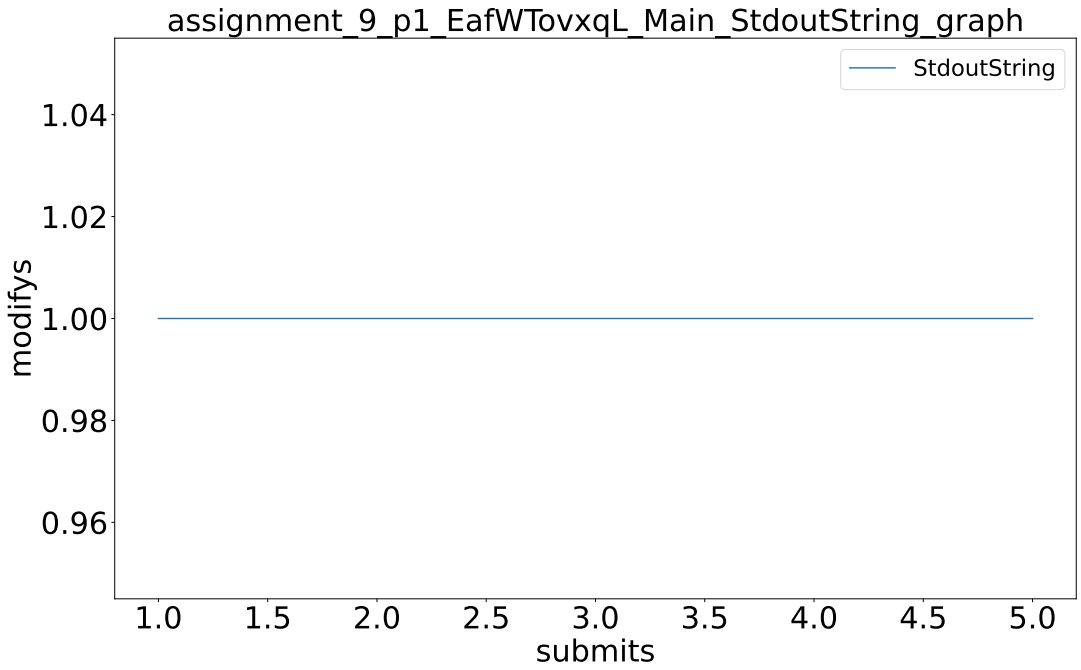

図 4.11　特定構文の誤りが残り続けている編集の例

うことで受講者ごとに各構文に対する理解度をスコアとして可視化できると考えられる．

図 4.11 に複数バージョンに渡って特定の構文の差分が消えない場合のグラフを示す．初回提出から最終提出にかけて“ExpressionStatement, MethodInvocation, METHOD_INVOCATION_ARGUMENTS, StringLiteral”の差分が編集されず存在している．図 4.12 に差分箇所を抜粋した元のソースコードを示す．本課題は入力した数値に相当する 2020 年の月の 1 日から月末までの日付と曜日を「年/月/日 (曜日)」のフォーマットで一覧表示するプログラムである．受講者が提出したソースコードは日と（曜日）の間に“/”が誤って入っており，表示フォーマットの誤りが初回提出から最終提出にかけて続いている．このような単純な間違いは繰り返される可能性が高いため，以降の課題に取り組む際の注意を促したり，ケアレスミスが多い受講者には複数の課題のデータを集計して，同様の誤りが頻発していることを示すフィードバックを行うことで，受講者の意識改善が期待できる．

図 4.12　差分箇所のソースコード抜粋

図 4.13，図 4.14，図 4.15 に複数クラスに渡って編集が繰り返されている編集の例を示す．Samurai クラス（図 4.14）と Socerer クラス（図 4.15）はいずれも 5 提出目で差分が無くなっているが，8 提出目で再度差分が出現し，最終提出までに差分が残っている．また，Character クラス（図 4.13）には初回から最後まで if 構文に差分がある．3 つのグラフから受講者は Character クラスに不具合が含まれていることに気づけず，既に完成している Samurai クラスと Sorcerer クラスに不要な修正を加えていることが読み取れる．複数クラスやメソッドを作成する課題において，一部が正しく完成しているにもかかわらず，他の部分の誤りが原因で不要な修正を加えている受講者は，プログラム全体の設計を把握できておらず，不具合箇所を正しく認識できていないと考えられる．同様の誤りは，提案手法を用いて算出した複数クラス／メソッドに対する編集履歴を同時に分析することで機械的に抽出可能である．このような編集が検出された際のフィードバックの例として，クラスやメソッドに対するユニットテストの重要性や具体的なデバック手法をまとめた資料や追加課題を提示し，各クラスやメソッドが入出力の仕様を満たしているか確認しながら作業を進める習慣を身に付けさせることが挙げられる．

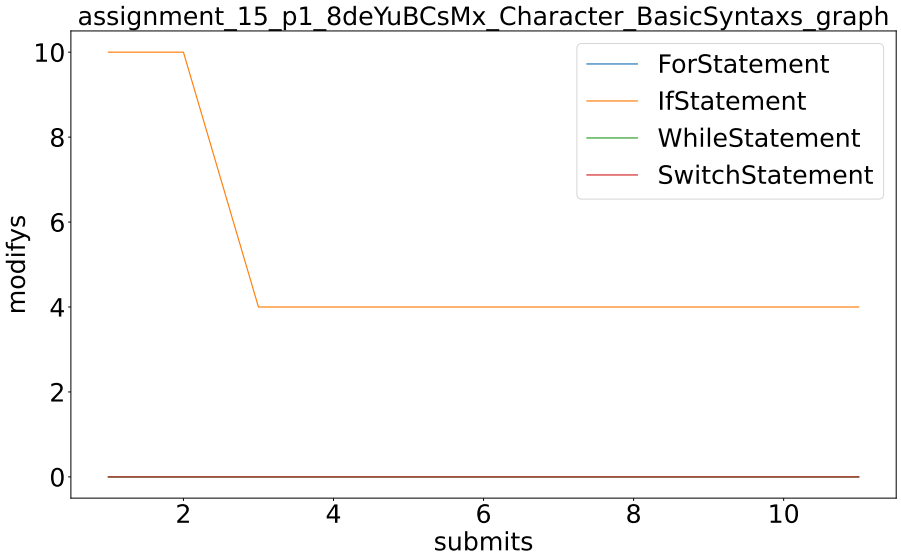

図 4.13　複数クラスに対する編集の例（Character クラス）

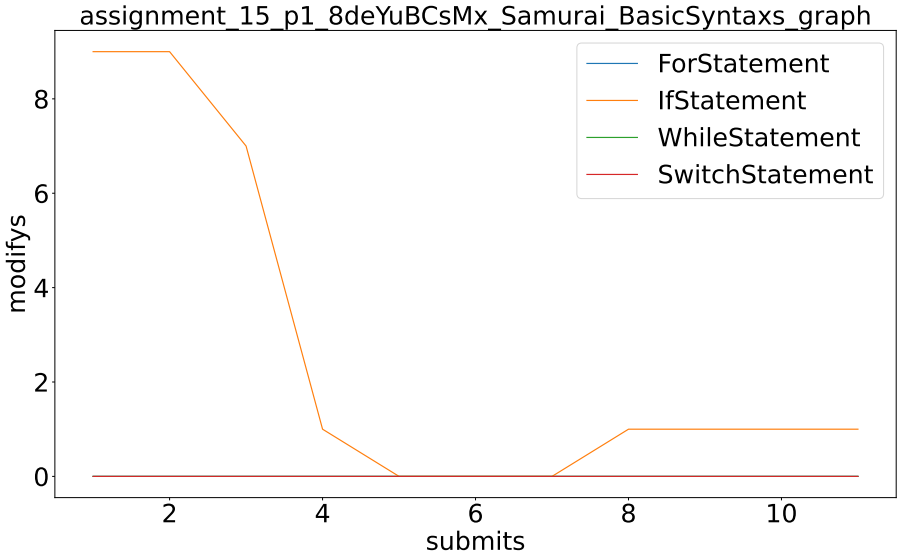

図 4.14　複数クラスに対する編集の例（Samurai クラス）

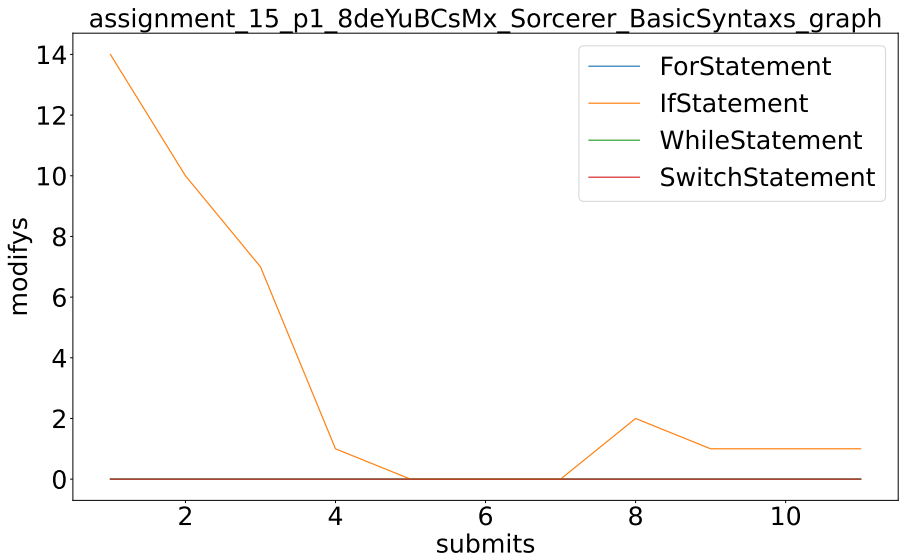

図 4.15　複数クラスに対する編集の例（Sorcerer クラス）

5. おわりに

本研究では OJS を用いたプログラミング講義において 1 人の受講者がある課題に完答するまでに提出した一連のソースコード群を対象として，最終版のソースコードとそれ以前に提出されたソースコードから差分の構文木を生成することで差分フローを抽出し，各提出段階における構文の編集回数を分析する手法を提案した．奈良高専本科の講義における課題の提出履歴を用いた複数の分析例を示し，提案手法の有用性を確認した．差分フローは受講者本人の完答ソースコードとの差分であるため，差分の変化を分析することでプログラム中のどの部分の編集を繰り返しているのか機械的な処理で得ることができる．また，差分フローはプログラミング講義の課題を完答するまでの編集を構文木の差分の列として出力することで，受講者が困難に感じている要素を発見し，それに沿ったサポートが可能になる．受講者が提出した一連のソースコードから差分フローを出力し，修正頻度の高い構文を分析することで，教師は講義資料の提示や追加課題の提供など受講者に適した指導方法を検討できる．受講者の取り組みから出力された差分フローを元に教師がフィードバックし，また受講者が学習を進めるというサイクルを通じて受講者の苦手な構文に対する理解が深まり，学習の効率化が期待される．

提案手法で得られる情報を応用することで, 以下のような学習支援の実現が期待できる．

- 不得意な学習単位の抽出
  講義の課題にて受講者が提出したソースコード群から差分フローを抽出し，各構文の編集頻度等の編集行動を特徴量として用いる事で受講者の理解不十分な要素や苦手な要素を抽出する機械学習に利用できる．
- 学習単位に対する理解のスコア化
  複数の課題に対して，完答したソースコードに使われている文法要素と差分フローを見ることで，課題内で使われている文法要素をどれだけ誤らずに利用できているか点数化する．教員による準備が必要な Unit Test に基づいた採点よりも簡易で，

より詳細な項目 (例えば else if 節の理解) に対する採点に応用できる．

- 生成 AI を用いたフィードバックの自動生成
  差分フローから得られる各構文に対する修正頻度などの情報をもとに，学習者の苦手とする文法要素などについて解説を求めるプロンプトを生成し，生成 AI に入力することで各受講者の理解度に沿った個別の資料などのフィードバックを自動生成できる．

謝辞

本研究を進めるにあたり，多くの方々のご助力をいただきました．この場を借りてお礼を申し上げます．指導教員である上野秀剛准教授には，常日頃から研究を進めるにあたってご指導いただきました．心から感謝申し上げます．

参考文献

[1] "IT 人材白書 2017", 独立行政法人情報処理推進機構, 2017.

[2] J. Falleri, F. Morandat, X. Blanc, M. Martinez, and M. Monperrus, "Fine-grained and accurate source code differencing. In Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering (ASE2014), pp.313-324, 2014.

[3] H. Sun, B. Li, and M. Jiao, ”YOJ: An online judge system designed for programming courses,” In Proceedings of the 9th International Conference on Computer Science Education (ICCSE2014), pp.812–816, 2014

[4] W. Zhou, Y. Pan, Y. Zhou, and G. Sun, "The framework of a new online judge system for programming education," In Proceedings of ACM Turing Celebration Conference (TURC2018), pp.9–14, 2018

[5] A. Koyuncu, K. Liu, T. F. Bissyandé, D. Kim, J. Klein, M. Martin, and Y. L. Traon, "FixMiner: Mining relevant fix patterns for automated program repair," Empirical Software Engineering Vol.25, No.3, pp.1980–2024, 2020

[6] H. A. Nguyen, A. T. Nguyen, T. T. Nguyen, T. N. Nguyen and H. Rajan, "A study of repetitiveness of code changes in software evolution," 2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp.180–190, 2013

[7] 藤本章良, 肥後芳樹, 松本淳之介, 楠本真二, プロジェクト全体の抽象構文木構築によるファイル間の移動コード検出, 電子情報通信学会論文誌 D, Vol.J104-D, No.4, pp.242-254, 2021.

[8] 松本淳之介, 肥後芳樹, 楠本真二, より短い編集スクリプトを目指して— 行単位の差分情報に基づく GumTree の拡張, 電子情報通信学会論文誌 D, Vol.J103-D, No.8,

pp.579-590, 2020.