Vol.2020-CE-154 No.4
2020/3/14

プログラミング学習者のつまずきの自動検出

浦上 理[1]　長島 和平[2]　並木 美太郎[2]　兼宗 進[3]　長 慎也[1]

概要：プログラミングの講義では，いくつかの課題が出されることが一般的である．プログラミング学習者は，エラーメッセージを理解できなくて修正できなかったり，課題で指定された題意を満たさない出力になってしまったときに何をしていいかわからなかったりすることがある．このように課題が正解に近づいていない，つまり課題の進捗が滞ることを『つまずき』として定義した．本研究では，プログラミング学習者のつまずきをソースコード編集履歴 (ログ) を用いて自動的に検出する方法を提案する．課題ごとに時系列でログをチェックし学習者が最終的に完成させた正解のソースコードと，ある時点での実行のソースコードとの差分をもとに，どのくらい正解に近づいているかを数値化した．この方法でつまずき検出ができれば，そのつまずきを自動的に教員に通知するシステムを構築することができるだろうと考えられる．

1. はじめに

プログラミングの講義では，いくつかの課題が出されることが一般的である．課題では，その日の講義で習った機能を用いて，指定された題意を満たすプログラムを書き，題意を満たしていれば正解となることが多い．学習者は，課題に取り組んでいる間，実行したときに文法などのエラーがあれば，開発環境が出したエラーメッセージをもとに修正し，エラーではないが，課題で指定された題意を満たさない出力になってしまったものについては，その原因を突き止め修正する必要がある．

しかし，プログラミング学習者は，エラーメッセージを理解できなくて修正できなかったり，課題で指定された題意を満たさない出力になってしまったものについては，開発環境はその原因を通知しないので何をしていいかわからなかったりすることがある．このように課題が正解に近づいていない，つまり課題の進捗が滞ることを「つまずき」として定義する．また，受講者が多いと教員はすべての学生のつまずきを見つけることが難しい実態がある．

本研究の目的は，学習者のつまずきを自動的に検出することである．検出したつまずきを教員に通知することで教員が指導したり，機械が自動的に指導したりするなどをし，学習者がつまずいている状態から復帰できるようになると考えた．

[1] 明星大学
Meisei University
[2] 東京農工大学
Tokyo University of Agriculture and Technology
[3] 大阪電気通信大学
Osaka Electro-Communication University

2. ログを用いたつまずきの検出

前述のとおり，つまずきとは課題の進捗が滞ることを指す．学習者の進捗を算出し，教員に把握させるシステムは，いくつか提案されている．

プログラム演習のための進捗モニタリングシステム [1] では，全クラスの進捗表示ページとクラス別進捗ページを出力し，評価日時，ファイル更新学生数，評価値の度数，座席番号，学生番号，各課題の評価値とファイル変更日時を表示して進捗状況を確認していた．ここでいう評価値は，事前テストや必須テスト，コンパイル，実行テスト，補足テストがどのくらいできたかを評価し値として出す．

授業支援システムにおけるプログラミング演習のための学習状況支援機能の設計と評価 [2] では，学習状況分析として，回答開始時刻，コンパイル時刻，実行時刻，正解時刻，提出時刻を現在の時刻と統計分析してつまずいている学生を分析していた．また，正解判定としてシステムが実行時に学生の回答と模範解答を比較して正解，不正解を画面に表示する．このシステムだと模範解答と違うやり方で題意を満たしていた場合でも不正解扱いにされる問題がある．

プログラミング入門教育を対象としたリアルタイム授業支援システム [3] では，評価項目としてコンパイル，インデント，クラスの形式的記述，ユニットテストで評価し，進行度の算出として学習者のソースコードが，コンパイル・インデント・クラスの形式的記述，ユニットテストの評価項目に対して正解している問題数を完全正解数と置き，その完全正解数が大きい学習者を上位として算出していた．

本研究では，先述したような課題に取り組んでいる間に

起きたエラーや題意を満たさないプログラムを総称して「間違い」のあるプログラムと呼び，学習者がその間違いの原因とは違うところを直している状態をつまずきの前兆とし，それをソースコードの編集履歴から検出する手法を提案する．

ソースコードの編集履歴を取得するためプログラム開発環境『Bit Arrow』[4] のログ収集機能を用いた．Bit Arrow のログは実行のたびに収集しており，実行内容の各項目には，次の情報が含まれている．

- ユーザ ID
- 実行時刻
- 実行時のソースコード
- 実行結果
- エラーメッセージ
- ファイル名

課題ごとにファイル名が指定されている場合[*1]は，そのユーザ名と課題ファイル名で学習者のある課題に対する行動を見ることができる．

3. 検出方法

それぞれの学習者がどの箇所でつまずいているかを把握するため，ある課題に対する学習者の進捗をグラフ化すると見つけやすいと考えた．そこで「正解行数」という値を導入する．

ログの各実行から特定の学習者のユーザ ID および調査対象となっている課題で指定されているファイル名を持つものを抽出し，その学習者が最終的に完成させた正解のソースコードとある時点での実行のソースコードとの差分を見て，同一であった行数を「正解行数」とする．正解行数がこれまでの最大値を上回っていなかったとき，その実行では完成に近づいていないと判断することとする．

また，正解行数を記録して，グラフ化することで，ある学習者がある課題のどのあたりでつまずいているかを見ることができると考えた．そのグラフの例を図 **1** に示す．

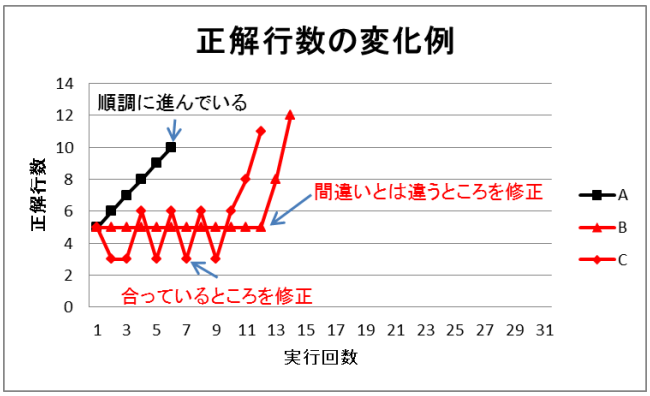

図 **1** 正解行数の変化例

[*1] Bit Arrow には教員によるソース配布機能があり，全受講者に課題用ファイルを同じ名前で配布できる．

図 1 で，順調に課題を進めている場合は，A の学習者のように順調に右肩上がりになっていると考えられる．間違いとは違うところの修正をしている場合は，B の学習者のように 1 回目の実行から 8 回目の実行まで正解行数に変化がないと考えられる．合っているところを修正している場合は，C の学習者のように 2 回目の実行が 1 回目の実行から下がってその後も上がったり下がったりしていると考えられる．という仮説を立てた．

4. 評価

正解行数がつまずきの有無を判断する指数として扱えるかを評価する．

4.1 調査対象

2018 年度に明星大学情報学部の 2 年次配当で開講された『プログラミング I』の講義を対象とした．この講義は，C 言語を用いた講義である．この講義には 32 名の受講者がいる．受講者は 1 年次に C 言語を用いた入門教育を受けている．調査したのは全 15 回中の 2 回目の講義で出された課題のうち「P0420_1.c」と「P0420_2.c」の 2 つである．この日の内容は繰り返し文で，課題も繰り返し文を使った課題が出されており，「P0420_1.c」は，繰り返し文を使い，1 から n までの総和を求める問題で，「P0420_2.c」は，繰り返し文を使い，1 から n までの総乗を求める問題である．

4.2 評価方法

最終的に正解になっていた学習者について正解行数のグラフ化を行い，実際のログを目視で見てつまずきを確かめた．そのつまずきの基準を次に示す．

- その課題を取り組んでいる間のログに残っている実行を古い順に 1 つずつ見ていく．
- エラーや題意を満たさない出力結果が出ている場合「間違い」が発生したとみなす．
- 間違いが解決されるまでの行動をログを見ながら追跡する．
- 間違いを解決する過程で，間違いの原因とは異なる場所を編集していたり，場所はあっているが的外れなプログラムを編集するような行動をしている場合，このユーザはこの箇所で「つまずいている」と判定する．

受講者 32 人の各課題のログの実行について，目視でつまずいているかを判定したものと正解行数のグラフの変化を集計した．

4.2.1 課題 P0420_1

「P0420_1.c」の目視によるつまずきの判定とグラフの変化を集計した結果を，表 **1** に示す．

表 1　P0420_1 の目視によるつまずきと，グラフの変化の件数

目視＼グラフ変化	上	下	平坦
つまずいている	6	10	15
つまずいていない	38	1	6

ユーザ 1　事例

ユーザ 1 のグラフを**図 2** に示す．

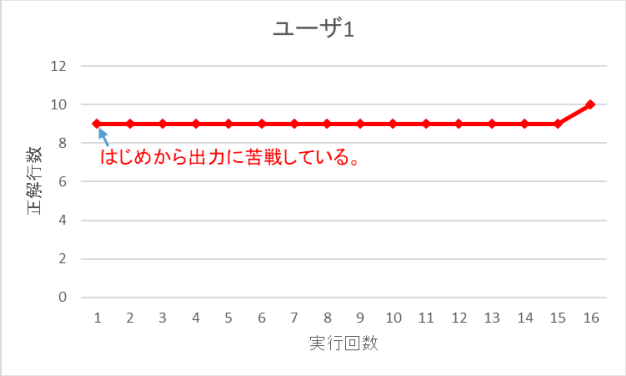

図 2　ユーザ 1 のグラフ

ユーザ 1 の学生は，図 2 を見ると 1 回目から 15 回目まで正解行数に変化がない．16 回目では実行結果の出力ができたので完成している．
ここでこの学生のプログラムの変化を示す．1 回目のプログラムを**図 3** に示す．

```
#include<stdio.h>
int main(void) {
    int sum=0,i,n;
    printf("imput n ");
    scanf("%d",&n);
    for(i=1 ; i<=n ; i++) {
        sum=sum+i;
    }
    printf("1 から n までの和=%d\n",sum);//A

}
```

図 3　ユーザ 1 の P0420_1.c（1 回目）

1 回目の実行から 3 回目の実行まで図 3 のとおりであった．次に 4 回目のプログラムを**図 4** に示す．なお，変更されていない部分は省略している．直前の実行からの正解行数の変化をキャプションの最後に示す．
図の凡例：△は進展した変更，▼は進展しない変更

```
printf("1 から&n までの和=%d\n",sum);//A：変更▼
```

図 4　ユーザ 1 の P0420_1.c（4 回目），平坦

5 回目のプログラムを**図 5** に示す．

```
printf("1 から%d までの和=%d\n",sum);//A：変更▼
```

図 5　ユーザ 1 の P0420_1.c（5 回目），平坦

```
printf("1 からまでの和=%d\n",sum);//A：変更▼
```

図 6　ユーザ 1 の P0420_1.c（7 回目），平坦

7 回目のプログラムを**図 6** に示す．
図 4 から図 6 まで図中 A の部分だけ変化があり，その後も出力に手間取っているように見られた．
16 回目のプログラムを**図 7** に示す．

```
printf("1 から%d までの和=%d\n", n,sum);//A：変更△
```

図 7　ユーザ 1 の P0420_1.c（16 回目），増加

ここでプログラムが完成している．
よって，正解行数の変化しない箇所では実際にこの学生はつまずいていると言うことができる．

ユーザ 2　事例

ユーザ 2 のグラフを**図 8** に示す．

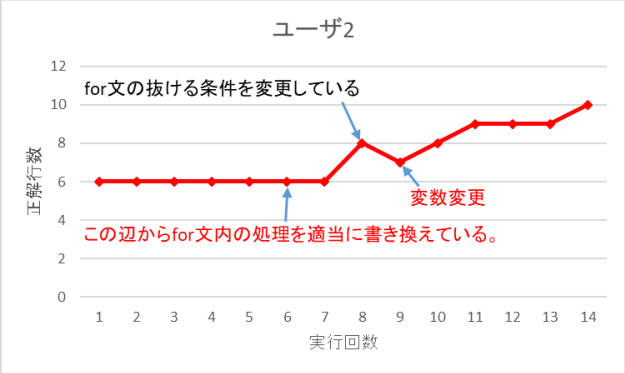

図 8　ユーザ 2 のグラフ

ユーザ 2 の学生は，図 8 を見ると 1 回目から 7 回目まで正解行数の変化がなかった．そのうえ 9 回目では正解行数が下がっていた．ただ，8，11，14 回目は正解行数が上がった．
ここでこの学生のプログラムの変化を示す．1 回目のプログラムを**図 9** に示す．

```
#include<stdio.h>
int main(void) {
    int x,i;//A
    printf("数値を入れてください：");
    scanf("%d",&x);
    for(i=0,i=x;i++){//B
        X=i+1;//C
    }
    printf("1 から 10 までの和＝%d",x);//D
}
```

図 9　ユーザ 2 の P0420_1.c（1 回目）

図中 B の部分が for 文の条件が違うのと図中 C の部分の x が大文字になっている．
次に 5 回目のプログラムを**図 10** に示す．

```
for(i=0;i>=x;i++){//B：変更▼
    x=i+1;//C
}
```

図 10　ユーザ 2 の P0420_1.c（5 回目），平坦

2 回目で図中 B の部分をカンマ（,）からセミコロン（;）に修正，4 回目で図中 C の部分を大文字から小文字に修正，5 回目では図中 B の部分を for 文の継続条件が i が入力した値を超えると抜ける条件にしている．ただ，このプログラムは，i に 1 足しているだけなので，総和のプログラムにならない．
6 回目のプログラムを図 **11** に示す．

```
for(i=0;i>=x;i++){
    x=x+1;//C：変更△
}
```

図 11　ユーザ 2 の P0420_1.c（6 回目），平坦

ここら辺から図中の C の部分をやみくもに変更しだす．
7 回目のプログラムを図 **12** に示す．

```
    int n,x,i;//A：追加▼
    printf("数値を入れてください：");
    scanf("%d",&x);
    for(i=0;i>=x;i++){
        n=i+1;//C：変更▼
    }
    printf("1 から 10 までの和＝%d",n);//D：変更▼
}
```

図 12　ユーザ 2 の P0420_1.c（7 回目），平坦

図 12 では，図中 A の部分で変数 n を追加し，図中 C と D の部分を書き換えている．
8 回目のプログラムを図 **13** に示す．

```
int sum=0,x,i;//A：変更△
printf("数値を入れてください：");
scanf("%d",&x);
for(i=0;i<=x;i++){//B：変更△
    sum=sum+1;//C：変更△
}
printf("1 から 10 までの和＝%d",sum);//D：変更△
```

図 13　ユーザ 2 の P0420_1.c（8 回目），増加

図 12 と比べ，図中 A の部分が変数 n を sum にし，0 で初期化したうえで図中 C と D を変更している．ただ，図中 B は for 文を抜ける条件を i>=x から i<=x に変更している点は正解に近づいており，これが正解行数の増加につながった．
9 回目のプログラムを図 **14** に示す．
図 13 と比べ，図中 A の部分が sum を消し，図中 C を変更している．この次の実行では，図 13 に戻している．
11 回目のプログラムを図 **15** に示す．
図 15 では，図中 C の部分が sum=sum+i という正しい

```
int x,i;//A：変更▼
printf("数値を入れてください：");
scanf("%d",&x);
for(i=0;i<=x;i++){
    i=i+1;//C：変更▼
}
```

図 14　ユーザ 2 の P0420_1.c（9 回目），減少

```
for(i=0;i<=x;i++){
    sum=sum+i;//C：変更△
}
printf("1 から 10 までの和＝%d\n",sum);//D：変更△
```

図 15　ユーザ 2 の P0420_1.c（11 回目），増加

文に変更している．その他にも D の部分が変更されている．計算は正しいが，入力した値が出力されていないので不正解となっている．
14 回目のプログラムを図 **16** に示す．

```
#include<stdio.h>
int main(void) {
    int sum=0,x,i;
    printf("数値を入れてください：");
    scanf("%d",&x);
    for(i=0;i<=x;i++){
        sum=sum+i;
    }
    printf("1 から%d までの和＝%d\n",x,sum);//D：変更△
}
```

図 16　ユーザ 2 の P0420_1.c（14 回目），増加

ここで完成をしている．
よって，正解行数が変化しない部分や正解行数が下がっている部分では，この学生はつまずいていると言うことができる．また，正解行数が上がった部分では，正解に近づいたと言える．

4.2.2　課題 P0420_2

「P0420_2.c」の目視によるつまずきの判定とグラフの変化を集計した結果を，表 **2** に示す．

表 2　P0420_2 の目視によるつまずきと，グラフの変化の件数

目視＼グラフ変化	上	下	平坦
つまずいている	1	9	13
つまずいていない	20	1	4

ユーザ 3　事例
ユーザ 3 のグラフを図 **17** に示す．

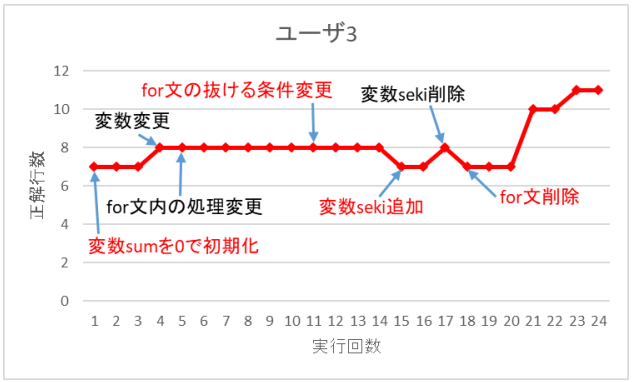

図 17 ユーザ 3 のグラフ

ユーザ 3 の学生は，図 17 では，4 回目から 14 回目まで正解行数に変化がない．そのためその部分がつまずいていると考えた．
ここでこの学生のプログラムの変化を示す．1 回目のプログラムを図 **18** に示す．

```
#include<stdio.h>
int main(void) {
    int n,sum=0,i;//A
    printf("input n:");
    scanf("%d",&n);
    printf("1 から%d まで積=",n);
    for(i=1 ; i<=n ; i++){//B
        sum=sum*i;//C
    }
    printf("%d",sum);//D
}
```

図 18 ユーザ 3 の P0420_2.c（1 回目）

図 18 では，図中 A の部分が sum が 0 になっているため繰り返し 0 をかけて出力している．その後は，図中 B の部分の for 文の変化式が sum=i*i や sum=i*i+1 にしていた．
4 回目のプログラムを図 **19** に示す．

```
    int n,s=0,i;//A：変更△
    printf("input n:");
    scanf("%d",&n);
    printf("1 から%d まで積=",n);
    for(i=1 ; i<=n ; i++){
        s = s * i;//C：変更▼
        i = i - 1;//E：追加▼
    }
    printf("%d",s);//D
```

図 19 ユーザ 3 の P0420_2.c（4 回目），増加

図 19 では，図中 A が sum だったところを s に変更している．それに伴い図中 C，D を変更している．また，図中 E が追加された．図中 C，E の処理をするとループが抜け出せなくなるので，エラーが出ている．[*2]ただ，このプログラムは正解行数が増えてはいるが，図中 D の部分が最後に実行したのと同じであるため増加した．これについては後程考察する．
5 回目のプログラムを図 **20** に示す．

```
for(i=1 ; i<=n ; i++){
    printf("%d x %d = %d\n", i, n, i*n);//C：変更▼
}
```

図 20 ユーザ 3 の P0420_2.c（5 回目），平坦

図 20 では，for 文内の処理が変わっている．この行動は，どういう処理が行われているかを確認するために実施したものと判断した．
6 回目のプログラムを図 **21** に示す．

```
for(i=1 ; i<=n ; i++){
    s=s*i*n//C：変更▼
}
```

図 21 ユーザ 3 の P0420_2.c（6 回目），平坦

図 21 では，図中 C の部分が変更されている．セミコロン（;）の入れ忘れは，次の実行で修正されている．その後は，図中 C の部分を s=s+i*n にしていた．
11 回目のプログラムを図 **22** に示す．

```
for(i=1 ; i<=n+1 ; i++){//B：変更▼
    s=i*n;//C：変更▼
    s=s*n;//E：変更▼
}
```

図 22 ユーザ 3 の P0420_2.c（11 回目），平坦

図 21 と比べ，図中 B の部分の for 文の継続条件と図中 C，E が変更されている．ただ，s が 0 のため繰り返し 0 で掛けてしまっている．
15 回目のプログラムを図 **23** に示す．

```
int n,s=0,seki=0,i;//A：追加▼
printf("input n:");
scanf("%d",&n);
printf("1 から%d まで積=",n);
for(i=1 ; i<=n ; i++){//B：変更△
    s=i*n;
    seki=s*n;//E：変更▼
}
printf("%d",seki);//D
```

図 23 ユーザ 3 の P0420_2.c（15 回目），減少

図 22 と比べ，図中 A の部分に変数 seki を追加し，図中 E，D を変更している．しかし，これは変数 s と同じ役割であり，余計である．その他にも図中 B の for 文の継続条件を元に戻している．
17 回目のプログラムを図 **24** に示す．
図 23 に比べ，図中 A の部分から変数 seki を削除し，図中 C，D を変更している．余計な変数を削除したため，正

[*2] Bit arrow では無限ループを感知すると実行を停止することを学習者に通知する仕組みがある．

```
int n,s=0,i;//A
printf("input n:");
scanf("%d",&n);
printf("1 から%d まで積=",n);
for(i=1 ; i<=n ; i++){
    s=n*n-1;//C：変更▼
}
printf("%d",s);//D：変更△
```

図 24 ユーザ 3 の P0420_2.c（17 回目），増加

```
printf("1 から%d まで積=",n);
s = n(n+1)/2;//F：追加▼
printf("%d",s);
```

図 25 ユーザ 3 の P0420_2.c（18 回目），減少

解に近づいた．
18 回目のプログラムを**図 25** に示す．

図 25 では，for 文を削除し，図中 F の部分の処理に書き換えている．その後は，s = n\(n+1\)/2 や s = n*n+1/2 に書き換えている．
21 回目のプログラムを**図 26** に示す．

```
int n,s=1,i;//A
printf("input n:");
scanf("%d",&n);
printf("1 から%d まで積=",n);
for(i=1;i<=n;i++){//B：追加△
    s = s*n//C：追加△
}
```

図 26 ユーザ 3 の P0420_2.c（21 回目），増加

図 26 では，図中 B の部分で for 文が復活し，そのうえ図中 A の変数 s が 1 で初期化されている．また，図中 C の部分を変更している．セミコロン（;）は次の実行で追加されている．s の初期値を変更したことで正解に近づいている．
23 回目のプログラムを**図 27** に示す．

```
#include<stdio.h>
int main(void) {
    int n,s=1,i;
    printf("input n:");
    scanf("%d",&n);
    printf("1 から%d まで積=",n);
    for(i=1;i<=n;i++){
        s = s*i;
    }
    printf("%d",s);
}
```

図 27 ユーザ 3 の P0420_2.c（23 回目），増加

ここで総乗のプログラムが完成している．

よって，この課題でも正解行数が変化しない部分や正解行数が下がっている部分では，この学生はつまずいていると言うことができる．また，正解行数が上がった部分では，正解に近づいたと言える．

このように正解行数のグラフと，目視によるつまずきの調査を照らし合わせた結果，平坦または下がっている部分ではつまずいている箇所が多く，上がっている部分では進展している箇所が多かった．このことから正解行数は，大方つまずきを検出できていることが分かった．

5. 考察

今回の評価で，正解行数の変化をグラフにし，グラフの平坦または下がっている部分を見れば，つまずいていることが一目で見ることができた．しかし中には正解行数が増えた部分でも進展していないことがあった．また，正解行数が変わっていなかった箇所でもプログラムが完成に近づいているケースもあった．これらのケースについて考察する．

まず，正解行数が増えた部分でも進展していないケースを**図 28** に示す．

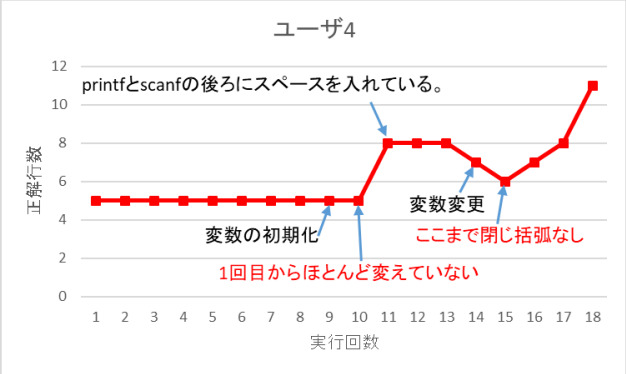

図 28 正解行数が増えた部分でも進展していないケース

図 28 では，P0420_1 でのユーザ 4 のグラフを挙げる．11 回目でそれまで平坦だった正解行数が増えた．
正解行数が上がる前に実行したソースコードと正解行数が増えた時のソースコードの比較を**図 29** に示す．

左側は正解行数が増える前に実行したソースコードで右側は正解行数が増えた時のソースコードである．黄色が付いた行は，変更があった箇所である．ここでの変更点は，printf と scanf の後ろにスペースを入れたことと，10 行目で改行コードを入れたことである．この変更をしても前に実行したときと実質同じ動作になるので，進展したとは言えないが，正解行数が増えていたためグラフ上では進展したように見える．このように動作が同じ書き方であれば，同じ行数として扱う必要があると考えた．前のプログラムと比較し、スペースを除いて同じであれば、同じ正解行数として扱うことで対処できると考えている．

次に正解行数が変わっていなかった箇所でプログラム完成に近づいているケースを**図 30** に示す．

図 30 では，P0420_1 でのユーザ 5 のグラフを挙げている．1 回目から 4 回目までの正解行数が変化がない．
1 回目に実行したプログラムと最後に実行したプログラム

Prev		Current	
1	#include<stdio.h>	1	#include<stdio.h>
2	int main(void) {	2	int main(void) {
3	int n,i,a;	3	int n,i,a;
4	printf("n= ");	4	printf ("n= ");
5	scanf("%d",&n);	5	scanf ("%d",&n);
6	a=0;	6	a=0;
7	for(i=1;i<=n;i++){	7	for(i=1;i<=n;i++){
8	a=a+i;	8	a=a+i;
9	}	9	}
10	printf("1から%dまでの和=%d",n,a);	10	printf ("1から%dまでの和=%d\n",n,a);

diff view generated by jsdifflib

図 29 正解行数が増える前に実行したソースコードと正解行数が増えた時のソースコードの比較

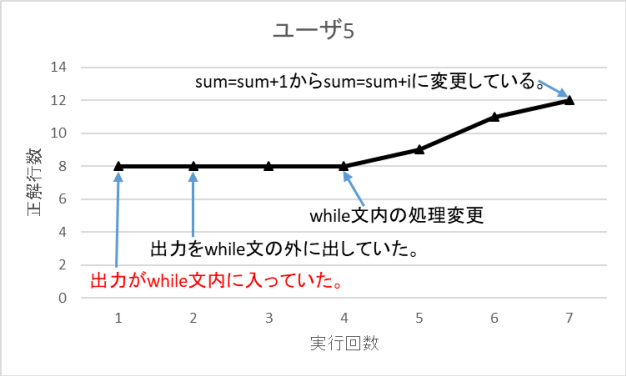

図 30 正解行数が変わっていなかった箇所でプログラム完成に近づいているケース

の比較を**図 31** に示す．図 31 では左側が 1 回目のプログラムで，右側が最後に実行したプログラムである．赤い線が引かれている部分は削除されている箇所．黄色い線は，変更があった箇所．緑は追加された箇所である．

次に 2 回目に実行したプログラムと最後に実行したプログラムの比較を**図 32** に示す．

図 32 では左側が 2 回目のプログラムで，右側が最後に実行したプログラムである．この時点で出力を while 文の外に出しているが，出力の表示が異なるので，正解行数に含まれていない．しかし赤や緑の行がなくなっているので 2 回目のほうが進展したと言える．

次に 3 回目と 4 回目に実行したプログラムの比較を**図 33** に示す．

図 33 では左側が 3 回目のプログラムで，右側が 4 回目に実行したプログラムである．3 回目まで sum+sum+1 だったのを sum=sum+1 に変更している．このことで sum を代入できるようになったので，進展したと言える．

このように適切な変更をしても正解行数が変わらない場合があるので正解と比較して，行全体が別の場所に移動している場合正解行数の値を低くしたり正解行数に含めることや同じ行で進展しているように見られる書き換え方をした場合に正解行数の値を増やしたりするなど，正解行数の算出方法を修正する必要があると考えた．

その他にも図 19 のように変数名を変えただけで正解行数が上がった事例がある．現時点ではプログラムのソースコードの比較を行っているが，変数名を適当なものに置換して，前の実行と同じプログラムであれば，同じ正解行数として扱う必要があると考えた．

6. まとめ

本研究では，つまずきを検出するため正解行数を導入し，その変化をグラフにした．その結果実行回数やグラフの平坦な部分を見れば，つまずいていることを一目で見ることができた．しかし，第 5 章のように正解行数が増えた部分でも進展していないケースや正解行数が変わっていなかった箇所でプログラム完成に近づいているケースが発生したので，今後はその事象を減らしつつ，つまずきを見つけるための検出方法を改善すべきだと考えた．また，検出にあたってグラフに出なかったつまずきを研究し，精度を高める．その他つまずきを教員に通知することや、機械が自動で指導することができるように開発をする．

参考文献

[1] 内藤広志他：プログラム演習のための進捗モニタリングシステム, 情報処理学会研究報告,2008-CE-93,pp. 33-40(2008)
[2] 加藤利康他：授業支援システムにおけるプログラミング演習のための学習状況支援機能の設計と評価, 情報処理学会研究報告,Vol.2012-CE-113,No.6,pp. 1-8 (2012)
[3] 長谷川伸他：プログラミング入門教育を対象としたリアルタイム授業支援システム, 情報処理学会論文誌,Vol. 52,No.12,pp. 3135-3149(2011)
[4] 長島和平他：Web ブラウザを用いたプログラミング学習支援環境 Bit Arrow の設計と評価, 情報処理学会論文誌教育とコンピュータ,Vol. 4,No. 1,pp. 57-69 (2018).

Current		Last	
1	#include<stdio.h>	1	#include<stdio.h>
2	int main(void) {	2	int main(void) {
3	int i=1,n,sum=0;	3	int i,n,sum=0;
		4	i=1;
4	printf("input:\n");	5	printf("input:\n");
5	scanf("%d",&n);	6	scanf("%d",&n);
6	while(i<=n){	7	while(i<=n){
7	sum+sum+1;	8	sum=sum+i;
8	i++;	9	i++;
9	printf("1から10までの和=\n");		
10	}	10	}
		11	printf("1から%dまでの和=%d \n",n,sum);
11	}	12	}

diff view generated by jsdifflib

図 **31**　1 回目と最後に実行したプログラムの比較

Current		Last	
1	#include<stdio.h>	1	#include<stdio.h>
2	int main(void) {	2	int main(void) {
3	int i=1,n,sum=0;	3	int i,n,sum=0;
		4	i=1;
4	printf("input:\n");	5	printf("input:\n");
5	scanf("%d",&n);	6	scanf("%d",&n);
6	while(i<=n){	7	while(i<=n){
7	sum+sum+1;	8	sum=sum+i;
8	i++;	9	i++;
9	}	10	}
10	printf("1から10までの和=\n");	11	printf("1から%dまでの和=%d \n",n,sum);
11	}	12	}

diff view generated by jsdifflib

図 **32**　2 回目と最後に実行したプログラムの比較

Prev		Current	
1	#include<stdio.h>	1	#include<stdio.h>
2	int main(void) {	2	int main(void) {
3	int i=1,n,sum=0;	3	int i=1,n,sum=0;
4	printf("input:\n");	4	printf("input:\n");
5	scanf("%d",&n);	5	scanf("%d",&n);
6	while(i<=n){	6	while(i<=n){
7	sum+sum+1;	7	sum=sum+1;
8	i++;	8	i++;
9	}	9	}
10	printf("1から%dまでの和=%d\n",n,sum);	10	printf("1から%dまでの和=%d\n",n,sum);
11	}	11	}

diff view generated by jsdifflib

図 **33**　3 回目と 4 回目に実行したプログラムの比較