Vol.2019-CE-151 No.6
2019/10/5

プログラミング演習における個別指導のための コーディング状況把握方法の提案

久保田　詩門[1,a)]　蜂巣 吉成[2,b)]　吉田　敦[3,c)]　桑原　寛明[4,d)]

概要：大学におけるプログラミング演習において，少数の指導者が限られた時間の中で，多数の学習者の個々のコーディング状況を把握する事は困難である．本研究では学習者のソースコードの模範解答への近付き具合を進捗度として定義し，単位時間あたりに変化したソースコード文字数を活動度として定義する．進捗度を横軸，活動度を縦軸とした平面上に学習者の状況をグラフで表し，サポートが必要な学習者を特定する方法を提案する．

キーワード：プログラミング演習，コーディング状況，サポート必要者の特定

A Method for Grasping Coding Status to Assist Students in Programming Exercise

Abstract:
In programming exercise at university, it is difficult for a small number of instructors to grasp how well each student have coded because there are many students and the exercise time is limited. In this research, we define degree of progress, which shows how much a student makes progress compared to a model answer. We also define degree of activity, which shows how many characters in source code are changed per unit time. We propose a way to identify students who need the support of instructors by expressing the situation of them in two dimension graph, whose horizontal axis is degree of progress, and the vertical axis is degree of activity.

Keywords: Programming Exercises, Coding Situations, Identifying students to need assistance

1. はじめに

大学におけるプログラミング演習では，学習者多数名，教員 1 名，TA 少数名という形式で授業が行われる事が多い．講義スタイルの 1 つに，学習者が共通の演習問題に取り組み，教員と TA(以下，まとめて指導者と表記) が教室を巡回しつつ適宜指導を行うものがあるが，学習者の課題の進捗には個人差がある．また，どうしたら良いか分からずに数分間にわたり手が止まっている学習者もいる．これらのような問題に対して少数の指導者で対応するには，学習者のソースコードがどれほど正解に近づいているか，また，どれだけ手を動かしてコーディングに取り組んでいるのかを把握する必要がある．しかし，演習時間内に指導者が教室内を巡回し全ての学習者の進捗を把握したり，どれだけ手が動いているのかを判断することは困難である．

石元ら [2] は，学習者のソースプログラムを制御構造や演算子，型に着目して同値類分割を行い，学習者全体のプログラムの傾向を指導者に示すことで，全体指導に活かす方法を提案した．多くの学習者が同じ時刻に同じような箇

[1] 南山大学 大学院理工学研究科 Graduate School of Science and Engineering, Nanzan University, 18 Yamazato-cho, Showa-ku, Nagoya-shi, 466-8673, Japan
[2] 南山大学 理工学部 Faculty of Science and Engineering, Nanzan University, 18 Yamazato-cho, Showa-ku, Nagoya-shi, 466-8673, Japan
[3] 南山大学 国際教養学部 Faculty of Global Liberal Studies, Nanzan University, 18 Yamazato-cho, Showa-ku, Nagoya-shi, 466-8673, Japan
[4] 南山大学 情報センター Center for Information and Communication Technology, Nanzan University, 18 Yamazato-cho, Showa-ku, Nagoya-shi, 466-8673, Japan
[a)] m19se003@nanzan-u.ac.jp
[b)] hachisu@nanzan-u.ac.jp
[c)] atsu@nanzan-u.ac.jp
[d)] kuwabara@nanzan-u.ac.jp

所でつまずいていればこの方法は有効であるが，実際には，進捗が大きく遅れている学習者や，進んではいるものの記述が足らずに完成に至らない学習者もいる．このような学習者を効率よく発見して，指導することも必要である．

本研究では，学習者個人レベルの指導を行うために，指導が必要な学習者を効率よく発見する方法を提案する．そのために，進捗度と活動度という 2 つの基準を定義する．進捗度は学習者のソースコードがどれだけ模範解答に近づいているかを表す．学習者のソースコードと模範解答を石元ら [2] が提案した方法で制御構造に着目して抽象化し，それらの差分により近さを求める．別解や記述のブレなどから，学習者のソースコードが必ずしも模範解答に近づくとは限らないので，実行結果が期待される結果と一致しているかも進捗度の評価に含める．活動度は学習者がどれだけ作業をしているかを表し，単位時間あたりに変化したソースコードの文字数を評価基準に用いる．例えば，進捗度が低く，活動度が高い学習者ほど指導の必要性は高く，プログラムの考え方などの基本的な指導が必要と考えられる．進捗度が高くても活動度が低い学生は，特定のプログラム記述でつまずいている可能性が高く，その箇所について指導すればプログラムが完成すると考えられる．活動度だけを見ても，進捗はしていて止まっているのか，進捗せずに止まっているかが分からず，サポート必要者の特定には繋がりにくいため，進捗度と活動度，両面からの分析が必要である．進捗度を横軸，活動度を縦軸とした進捗度活動度平面を定義し，指導が必要な学習者を効率よく発見する方法を提案する．

2. 関連研究

プログラミング演習において，学習者の進捗を把握するシステムがいくつか提案がされている．井垣ら [1] は学習者のオンラインエディタ上のコーディング過程を分析し，可視化して指導者へ提示するシステム C3PV を提案している．学習者が入力したソースコードの行数，課題ごとのコーディング時間，単位時間あたりのエディタ操作数，エラー継続時間の 4 種類のメトリクスを計測し，学習者内で順位づけしたものを表示することで，進捗の差を把握できる．相対的に遅れている学習者の座席情報に基づいて強調表示することで進捗状況の把握と指導が容易になっている．

石元ら [2] はプログラミングにおいて重要な，変数の値を変化させるという観点に着目し，ソースコード中の演算と変数の型について同値類分割を行う方法を提案している．同値類分割を行うことで読むソースコードの総数を減らすことができるので，少数のソースコードから全体の進捗状況の把握を行う．

長谷川ら [3] は授業中リアルタイムに課題を提示し，提出された学習者のソースコードに関してコンパイルやインデントなどを自動評価することができる，統合リアルタイム授業支援システム IDISS を提案している．また，提出履歴から分析を行う．これにより，人的リソース問題を解消でき，理解度に応じた授業進行が可能となる．

藤原ら [4] は学習者のコーディング過程を対象とした教育支援システムを提案している．学習者によるコンパイル回数，実行回数，コンパイルエラー数，行数などの履歴を計測及び分析する．藤原らのシステムでは，指導者が利用したいコーディング過程の各種履歴を組み合わせる．決められた組み合わせに合致した学習者は進捗が遅れているとして検出できる．

これらの研究と本研究との違いを述べる．[1][3] は学習者の進捗状況について，相対的に遅れている学習者や誤ったソースコードを記述している学生の特定をすることはできる．しかし，コーディング過程やどのような誤りをしているかを把握することは困難である．また，コンパイル時やファイル保存時などのソースコードを参考にするので，任意のタイミングでソースコードを見ることはできない．

石元ら [2] はソースコード中の演算と変数の型について同値類分割を行い進捗状況を把握しているので，演算が単純な問題については同値類分割はまとまるが，演算が長くなる問題については記述が複数あり，分割結果がまとまらないという課題がある．また，全体の進捗状況を把握することはできるが，学習者個人の進捗状況を把握できない．

藤原ら [4] は，各種履歴の分析と組み合わせの設定を指導者が事前に行う必要があり，指導者への負担が大きい．また，決められた組み合わせに合致した学習者内での指導の優先順位が判断できない．

3. 進捗度及び活動度を用いたコーディング状況把握方法の提案

本研究では学習者のソースコードと模範解答との近づき具合と，実行結果の正しさを進捗度として，単位時間内に変化した学習者のソースコードの文字数を活動度として定義する．進捗度と活動度を総合評価して，これまでの研究では発見が困難であった，あともう少しで正答できるが，その先の解法が思い付かず，手詰まりの状態にある学習者などを発見する方法を提案する．

3.1 進捗度の定義

3.1.1 編集距離の分析

学習者のソースコードと模範解答との編集距離で進捗を調べたいが，識別子名やコーディングスタイルの違いがある．石元ら [2] の方法で学習者のソースコードに対して閉じ括弧の補完，正規化，抽象化を行ったものを以後，評価コードと呼ぶ．制御構造を正しく記述することが重要と考えて，評価コードの編集距離 (レーベンシュタイン距離) で進捗度を計算する．その際，if, for, while の予約語を挿入，削除，置換の単位とする．閉じ括弧の補完では，編集途中

で足らない閉じ括弧を追加する．例えば，学習者のソースコードが図 1 ならば，閉じ括弧が 1 つ不足するので，図 2 のように最後に閉じ括弧を 1 つ追加する．正規化では if, for, while が単文のときに，{} を追加して複合文にする．図 2 では最初の for が単文であるので，図 3 のように複合文にする．抽象化では if, for, while の予約語と (), {} を抽出する (図 4)

```
for(j=1;j<=n;j++)
 for(i=1;i<=n;i++){
  x=j*i;
   printf("%d",x);
```

図 1 学習者のソースコード

```
for(j=1;j<=n;j++)
 for(i=1;i<=n;i++){
  x=j*i;
   printf("%d",x);
  }
```

図 2 閉じ括弧補完後のソースコード

```
for(j=1;j<=n;j++){
 for(i=1;i<=n;i++){
  x=j*i;
   printf("%d",x);
  }
 }
```

図 3 正規化処理後のソースコード

```
for(){
for(){
}
}
```

図 4 抽象化処理後のソースコード

以下に，2 重の繰り返しが必要な課題の編集距離を算出する例を示す．

for(){	for(){	for(){	for(){
for(){	while(){	if(){	}
}	}	}	for(){
}	}	}	}
模範解答	学習者 A	学習者 B	学習者 C

図 5 評価コードの例

単純な編集距離を算出すると，以下の通りになる．

- 模範解答と A の距離=1(2 番目の while を for に置換)
- 模範解答と B の距離=1(if を for に置換)
- 模範解答と C の距離=2(最初の } を削除，最後に } を挿入)

単純な編集距離では A=B<C となるが，プログラム処理を考慮すると，A<C<B とすることもできる．学習者 B は繰り返しが 2 つ必要なことがわかっていないと考えられる．学習者 C は繰り返しが 2 つ必要なことはわかっていると考えられるが，入れ子になっていない．そのため，編集作業に重みをつけ，編集距離を算出する．以下に編集作業への重みの付け方を示す．

- 予約語の置換の距離 (for<->while) は 0.5
- 他 (for<->if, while<->if) は 2
- { } の削除，挿入は 0.5

これらを考慮した編集距離は以下となる．

- 模範解答と A の距離=0.5(2 番目の while を for に置換)
- 模範解答と B の距離=2(if を for に置換)
- 模範解答と C の距離=1(最初の } を削除，最後に } を挿入)

いえる．

学習者 s の時刻 t における編集距離による進捗度 $progress_{edis}(s,t)$ の定義を次に示す．

$$progress_{edis}(s,t) = 100\frac{m-x_{s,t}}{m}$$

m：何も書いてないときと模範解答の編集距離
$x_{s,t}$：時刻 t の学習者 s の評価コードと模範解答の編集距離

学習者 A，B，C において進捗度を算出すると次のようになる．

- $progress_{edis}(A,t) = 100\frac{6-0.5}{6} \fallingdotseq 91.6$
- $progress_{edis}(B,t) = 100\frac{6-2}{6} \fallingdotseq 66.6$
- $progress_{edis}(C,t) = 100\frac{6-1}{6} \fallingdotseq 83.3$

3.1.2 実行結果の分析

別解の場合には編集距離による進捗度が 100 にはならない．しかし，解にはたどり着いているので，実行結果による分析により適切な進捗度を算出できる．実行結果による進捗度 $progress_{exec}(s,t)$ の定義を次に示す．

$$progress_{exec}(s,t) = \begin{cases} 30 & \text{(最初にコンパイルしたとき)} \\ 50 & \text{(最初にコンパイルが成功したとき)} \\ 50+50\frac{p}{n} & \text{(実行結果通りに動いたとき)} \end{cases}$$

p は期待した結果が得られたテストケースの数であり，n はテストケースの総数である．重みの付け方は課題が終了するまでの時間のかけ方の割合を基準としている．

3.1.3 総合的な分析

$progress_{edis}(s,t)$ と $progress_{exec}(s,t)$ の値を比較し，大きいほうを総合的な進捗度 $progress(s,t)$ と定義する．

$$progress(s,t) = max(progress_{edis}(s,t), progress_{exec}(s,t))$$

以降は $progress(s,t)$ を進捗度として扱う．

3.2 活動度の定義

コーディングの際に思うように進んでいない学習者を特定するために学習者がどのくらい作業しているのかを活動度として表す．活動度は 0〜100 の範囲で定義し，活動しているほど 100 に近くなる．学習者ごとの単位時間あたりに増加した文字数を計測する．計測にはソースコードファイルのバイト数を用いる．学習者 s の時刻 t における活動度 act(s,t) の定義を次に示す．

$$act(s,t) = 100 \sum_{i=1}^{N} w_i \frac{x'_{s,t-i+1}}{B}$$

$$\sum_{i=1}^{N} w_i = 1, \quad x'_{s,t} = \begin{cases} min(x_{s,t}, B) & (x_{s,t} \geq 0) \\ 0 & (otherwise) \end{cases}$$

$x_{s,t}$: 学習者 s の時刻 t から (t-1) 間で変化した
ソースコードバイト数

B：1 分間あたりに変化する文字数の基準値 (バイト数)

w_i: i 分前から (i-1) 分前の重み

N：活動度として計測する時間 (分)

直近の作業量が学習者への影響が大きいということが予想されるので，重み付けを行う．$x_{s,t}$ の値が負の場合には，ソースコードの編集が進んでいないものとし，$x'_{s,t}$ の値を 0 とする．

3.3 進捗度活動度平面の定義

進捗度を横軸，活動度を横軸として進捗度活動度平面を定義する．各問において進捗度活動度平面上における学習者の動きは図 6 のようになると予想される．

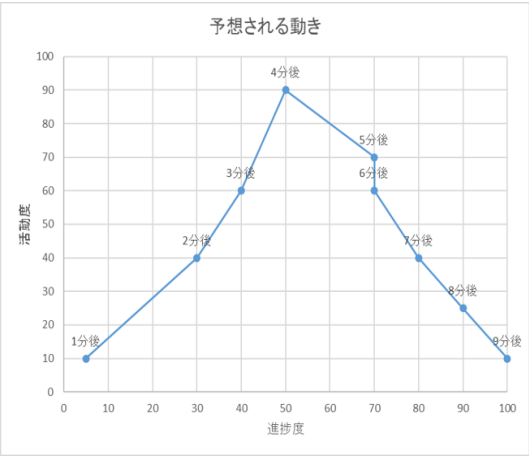

図 6 進捗度活動度平面において予想される動き

学習者は解法を考えるために，最初のうちは進捗度，活動度がともに低い状態で推移すると予想できる．その後，解法を思い付くとコーディングを進めるので，活動度は上昇し，進捗度もそれに伴って上昇し，終盤にさしかかるにつれて記述量が低下するので，活動度は低下し，進捗度は 100 に到達すると考えられる．

進捗度，活動度を総合評価することで，進捗度，あるいは活動度のみでは判断できなかった学習者の様子を把握できる．例えば，活動度の視点のみで演習の流れを見ていると，問題番号の移行があった際に学習者がそれまで解いていた問題が正解に辿り着いたのか判断ができない．予想として移り変わるタイミングで活動度が低ければ正解に辿り着いたので，正答しているという意味で取ることができる．一方で活動度が高ければ，その問題を解くことを中断し，他の問題に取り組み始めたと考えられる．しかし，これには途中でペースを急激に上げて正答した学習者などの例外が多く存在すると考える．そこで進捗度による観点から，進捗度が 100 を計測している場合にはその問に正答したという判断が可能となる．このように進捗度，活動度の両面からの観測によって，サポートが必要な学習者を特定する方法を提案する．

3.4 サポート必要学習者の特定方法の提案

進捗度と活動度の関係から，学習者がどのように問題を解き進めているのかを把握することができる．以下の図 7，8，9，10 は実際の演習データではなく，例として考えられる学習者の様子を進捗度活動度平面に示したグラフである．

3.4.1 方法 1

各学習者に対し，演習問題に解答するために必要となる上限時間を目安時間とする．目安時間は過去の類似の演習問題の経験や学習者の習熟度などから，あらかじめ指導者が設定するものとする．各問に取り組み始めて目安時間の半分が経過した時点で，進捗度が半分の 50 以上である場合は順調に正解へと近付くことができているといえる．この視点から，目安時間の半分以上経過しているとある時点で進捗度が 50 以下であり，かつ活動度が基準値である 30 を下回っている場合を想定する．この状態にある学習者は思うように正解に近付けておらず，なおかつ試行回数も減っており，指導が必要な可能性が高いと予測を立てた．

進捗度活動度平面で表すと，図 7 の赤線で囲んだ範囲に目安時間の半分以上が経過してから入った学習者が対象となる．

事象 1 その問を解き始めてから目安時間の半分以上が経過している

事象 2 進捗度が 50 以下である

事象 3 活動度が基準値の 30 以下である

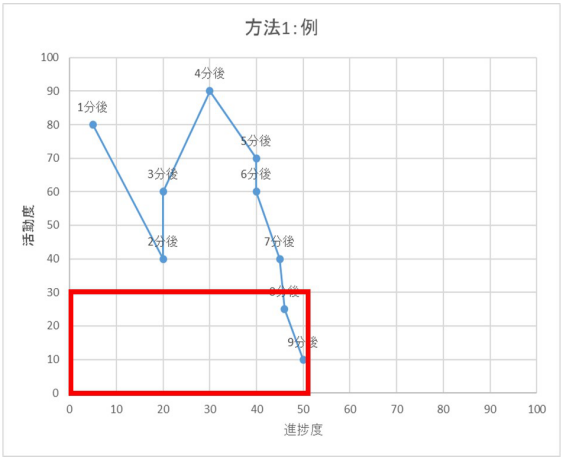

図 7　進捗度活動度平面における方法 1 の該当範囲

3.4.2　方法 2

活動度が基準値よりも低い状態が 2 分以上継続しており，かつその間に進捗度が変化しないもしくは低下している学習者は編集距離の遠近は問わず，その先の解答が思い付かず，かつ試行回数も減り，手詰まりの状態にあるためサポートが必要な可能性が高いと予測を立てた．

進捗度活動度平面で表すと，図 8 の赤線で囲んだ範囲に 2 分以上とどまっていた学習者が対象となる．

事象 4　活動度が基準値 (30) 以下の状態が 2 分以上継続している

事象 5　進捗度がその区間で変化なし，もしくは低下している

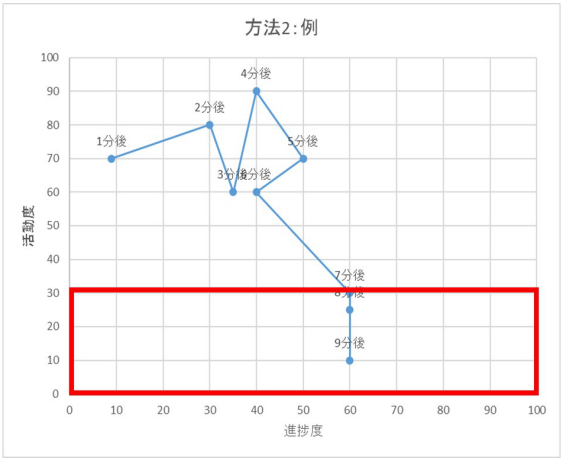

図 8　進捗度活動度平面における方法 2 の該当範囲

3.4.3　方法 3

進捗度が 80 以上であるということは模範解答に近い状態まで進捗していると判断できる．しかしその状態で活動度が継続して下がっているということは，あと一歩のところで行き詰まっていると予測を立てた．

進捗度活動度平面で表すと，図 9 の赤線で囲んだ範囲に 2 分以上とどまっていた学習者が対象となる．

事象 6　活動度が基準値 (30) 以下の状態が 2 分以上継続している

事象 7　進捗度が 80 以上である

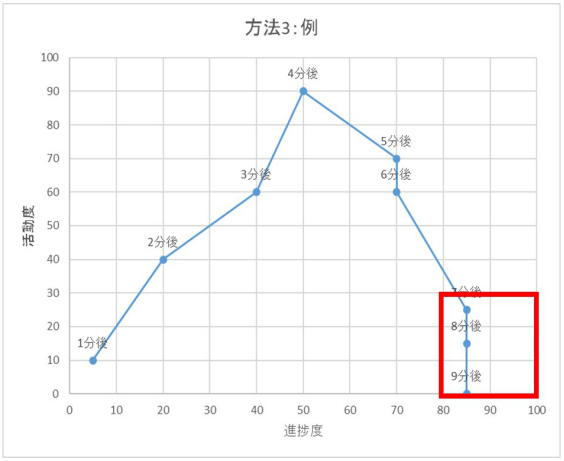

図 9　進捗度活動度平面における方法 3 の該当範囲

3.4.4　方法 4

進捗度が非常に低く (10 以下)，活動度が高い (80 以上) 状態が継続している場合，その学習者は試行回数を踏んでいるにも関わらず問題を解き進められていないと考えられる．よってサポートが必要だと予測を立てた．

進捗度活動度平面で表すと，図 10 の赤線で囲んだ範囲に 2 分以上とどまっていた学習者が対象となる．

事象 8　活動度が 80 以上の状態が 2 分以上継続している

事象 9　進捗度が 10 以下である

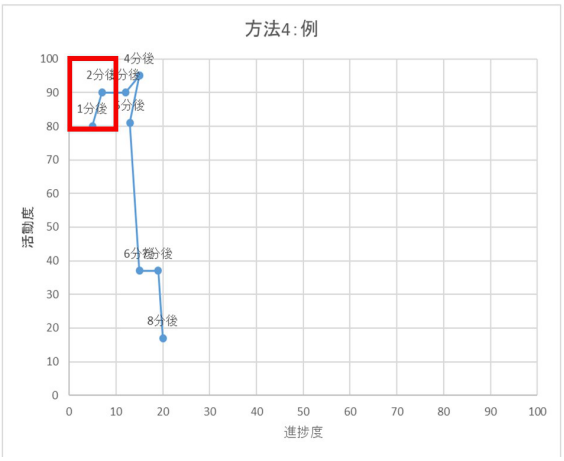

図 10　進捗度活動度平面における方法 4 の該当範囲

4.　評価

4.1　概要

進捗度と活動度を用いてサポートが必要な学習者を特定するにあたり，それらの評価が妥当かどうかを，実際の演習データを使って確認する．妥当であると確認した後に，

3 章で提案した方法でサポートが必要な学習者を特定できるかを評価する．

4.2 実験環境

本研究の演習データには，石元ら [2] の演習で得られたデータを用いる．C 言語を学んだ学部 3 年生 11 名に協力してもらい，60 分の演習形式で下記の関数を作成する問題を実行例とともに出題した．これらの問題では条件分岐，繰り返し，配列，ポインタ，再帰関数などの C 言語学習の主要な単元を網羅している．

問 1 整数の n 乗 (n>=0 の整数値) を計算する関数
問 2 実数の n 乗 (n は整数値) を計算する関数
問 3 文字列を変換し，ポインタで返す関数
問 4 ユークリッド互除法を用いた再帰関数
問 5 実数配列を逆順にする関数，2 つの実数値を入れ替える関数

実験の制約により，main 関数はあらかじめ作成されている．進捗度評価では学習者が記述した関数のみを分析対象とする．活動度評価ではソースコードファイル全体分析の対象とする．

4.3 進捗度について

$progress_{edis}$ はコンパイルする前から進捗度が計算できるが，別解を記述していると進捗が進んでいないと判断される恐れがある．$progress_{exec}$ はコンパイルするまでは進捗度が計算できないが，コンパイルした後は別解の記述をしていても進捗度が計算できる．これらの 2 つを組み合わせることで進捗度が 100 に近づくことを確認するために，文献 [2] のデータを用い，次の実験を行う．

実験 総合的な進捗度が 100 に近づくかを確認する．
結果

図 11 に学習者 B の編集距離による進捗度を示す．図の縦軸は進捗度であり，横軸は演習経過時間 (分) を表す．青が問 1，赤が問 2，黄が問 3，緑が問 4，紫が問 5 である．図 12 に学習者 B の実行結果による進捗度を示す．図 13 に学習者 B の総合的な進捗度を示す．

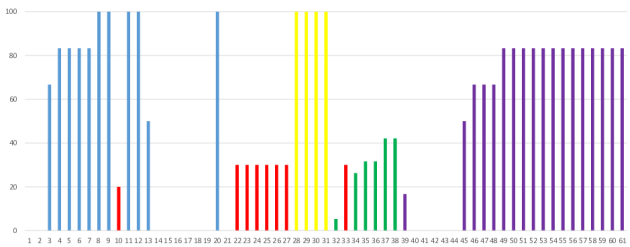

図 11 学習者 B の編集距離による進捗度

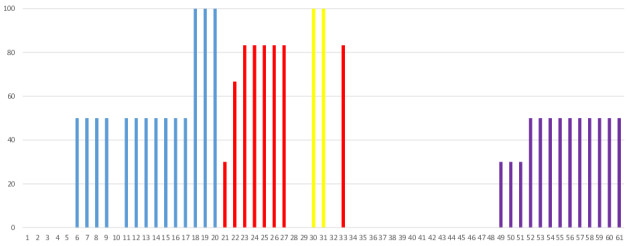

図 12 学習者 B の実行結果による進捗度

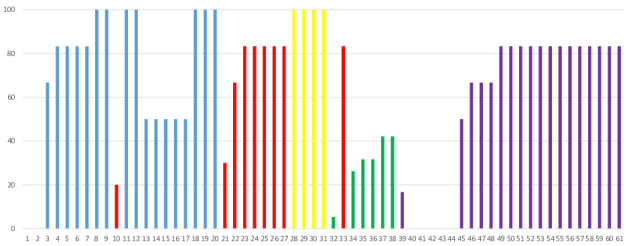

図 13 学習者 B の総合的な進捗度

これらの結果より，最終的な進捗度は 100 に近付くといえることが分かった．以後は上記の総合的な進捗度を進捗度として扱う．

4.4 活動度について

活動度の定義における N と w の妥当な値を確認するために文献 [2] の演習データを用い，次の実験を行った．

予備実験 変化したソースコード文字数の平均値及び各問の解答時間の平均値を求める．
実験 1 活動度の妥当な計測時間 N を確認する．
実験 2 活動度への重み付けにより，指導が必要な学習者を見つけやすくなるかを確認する．

演習問題は 5 問あり，問 2 は実数のべき乗を計算する関数を作成するものである．実験 1，2 では予備実験より B=5.8 を用いる．実験 1 では，N=1 のとき $w_1 = 1$，N=5 のとき $w_i = 0.2 (i = 1, 2, \cdots, 5)$ を用い，実験 2 では実験 1 より N=5 を用いる．活動度が 30 以下の場合を活動度が低いものとする．実験 2 における重み付けとして，より直近の情報が学生への影響が大きいと考え，次を用いる．

重み A 各 0.2 に重み付け．
重み B 直近から 0.3，0.25，0.2，0.15，0.1 に重み付け．
重み C 直近から 0.5，0.3，0.1，0.075，0.025 に重み付け．

結果

予備実験では，ソースコード文字数の変化の平均値は 5.8 であり，標準偏差は 16.2 であった．また，学生 11 名の各問の解答時間の平均値は問 1 から順に 8.4 分，8.6 分，7 分，13 分，13.2 分であった．

実験 1 では，図 14，15 に各学生の問 2 の解答時間を示す．縦軸は学生である．横軸は問 2 を解き始めてからの経過時刻 t である．棒グラフの長さは解答時間であり，活動度が 30 以下となった時間から赤色とする．

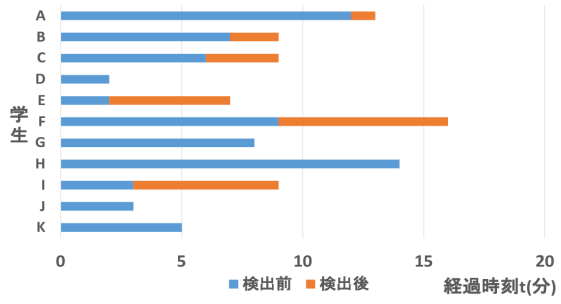

図 14　実験 1: 学生の問 2 の解答時間 (N=1)

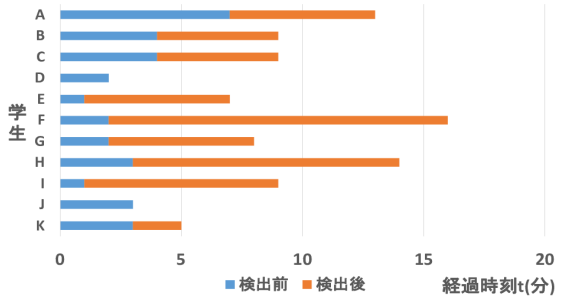

図 15　実験 1: 学生の問 2 の解答時間 (N=5)

実験 2 では，図 16 に学生 A の解き始めから 20 分経過時点までの活動度への重み付けを示す．縦軸は活動度，横軸は経過時刻 t である．

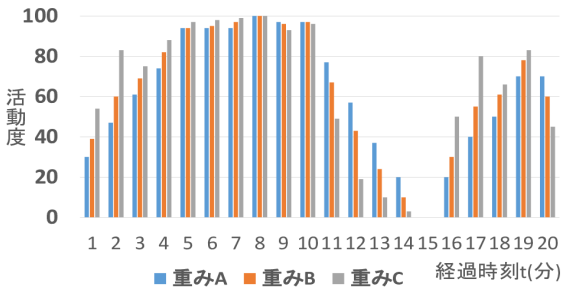

図 16　実験 2: 学生 A の活動度への重み付け

これらの結果から，今回用いた演習データにおいて重み付けによる効果的な差はないことが分かった．よって以下の総合評価では，上記の重み A で計算した数値を活動度として扱う．

4.5　進捗度及び活動度の総合評価

3 章で提案した方法により，サポートが必要な学習者が特定できるか過去に行った演習データを用いて確認する．過去の演習データから 3 章の方法 1〜4 に適合する学習者を抽出する．抽出した学習者が各問を解き終えるまでに要した時間を調べ，目安時間より長い，もしくは解き終えていない場合をサポートが必要であったと判断する．60 分で計 5 題出題しているので，目安時間を 12 分とした．

4.5.1　方法 1 について

11 人の学習者データを示した上記グラフから事象 1，2，3 を満たす部分は計 10 箇所検出された．このうち 7 箇所分，つまり 70 %は解き終えていない問が含まれていた．この際，同学習者の同一問題に複数含まれていた場合は重複と見なし 1 箇所とカウントしている．計 10 箇所の問を解き終わるまでにかかった時間，もしくは解き終えていない問についてはその問を解いていた時間の平均は，目安である 12 分を越える約 18 分となった．上述の通りこの中には最後まで解き終えていない問が 7 箇所分含まれており，正答するまでに要する平均時間はさらに増えると考えられる．すなわち事象 1，2，3 を満たした学習者はサポートが必要な可能性が高いと言えることが分かった．

4.5.2　方法 2 について

学習者 11 人分のグラフから事象 4 に該当する箇所は計 36 箇所検出された．このうち事象 5 にも該当する箇所は 22 箇所であり，その割合は約 61 %であった．その箇所が含まれる問を解き終えるまでに掛かっている平均時間を求めると，目安時間を越える約 16 分となった．ただし，この算出には最終的に解き終えていない問に取り組んでいた時間も含まれているので，実際に解き終えるまでには更に時間がかかると考えられる．事象 4 かつ事象 5 を満たす区間で，最終的にその問を解き終えていない箇所は約 63 %となった．これにより本研究で扱ったデータからは，事象 4 かつ事象 5 を満たした学習者は半分以上の確率でサポートが必要だと判断できることが分かった．

4.5.3　方法 3 について

学習者 11 人分のグラフから事象 6，事象 7 を共に満たす箇所は計 16 箇所検出された．このうちその箇所が含まれる問を解き終えるまでに掛かっている平均時間を求めると，目安時間を越える約 14 分となった．この算出には最終的に解き終えていない問に取り組んでいた時間も含まれているので，実際に解き終えるまでには更に時間がかかると考えられる．事象 6 かつ事象 7 を満たす区間で，最終的にその問を解き終えていない箇所は 12 箇所となり，約 75 %が解き終えていないことが分かった．これにより本研究で扱ったデータからは，事象 6 かつ事象 7 を満たした学習者はサポートが必要な可能性が高いと判断できることが分かった．

4.5.4 方法 4 について

学習者 11 人分のデータから事象 8 と事象 9 を満たす箇所は 1 箇所しか検出されなかった．しかし，初学者を対象とした場合はこの領域に入る学習者も増加すると考えられる．

5. 考察

実際の演習データを進捗度活動度平面に表すと，当初の予想とは異なり，図 6 のような山を描くグラフは一つもなかった．進捗度活動度平面での学習者のグラフの形などから，サポートが必要な学習者を特定する方法も今後検討する．以下に特徴的なグラフを複数あげる．

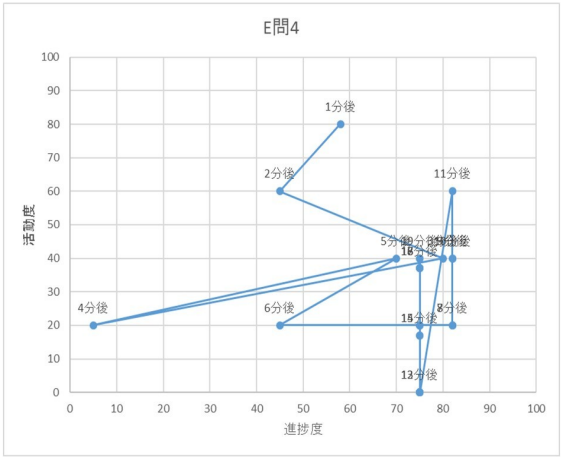

図 17 学習者 E の問 4

図 17 は問題開始直後の状態では，活動度の情報から手を動かしてコーディングしていると判断できるが，進捗度は何度も低下して最終的に正解に辿り着けていないため指導が必要であったといえる．

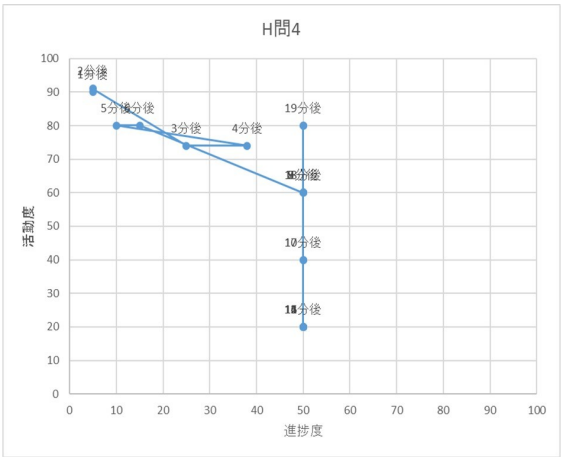

図 18 学習者 H の問 4

図 18 は活動度は高いが，進捗度が思うように上昇していない．結果 19 分経過時点でも進捗度は 50 となっており，指導が必要であったと考えられる．

6. おわりに

今回の実験により，進捗度と活動度の総合評価によってサポートが必要な学習者を発見することができた．本研究では実際の演習中に使用するシステムの環境構築は行っていないため，実際の演習でどのように提案方法を活用していくのかが課題として残っている．

さらに本研究のシステムの実用化についても考察を行う．プログラミングを学ぶ上で，学習者がプログラムの記述過程でつまづき，正解に向けて試行錯誤するという状況は，プログラミング技術向上のために不要とはいえない．本研究では，サポートが必要な学習者の評価方法を提案したが，サポートが必要な学習者をすぐに指導すべきだとは必ずしもいえない．そのため，今後は本研究のシステムを用いた場合の指導方法についても検討が必要になる．

謝辞

本研究を進めるにあたり，卒業研究を一緒に取り組んだ川出涼雅君と五十里貴斗君，実験に協力していただいた蜂巣研究室の学生の皆様に深く感謝致します．

本研究の一部は，JSPS 科研費 17K00114，17K01154，2019 年度南山大学パッヘ奨励金 I-A-2 の助成を受けた．

参考文献

[1] 井垣宏, 井上亮文, 齊藤俊, 山田誠 :『プログラミング演習における学生のコーディング過程可視化システム C3PV の提案』. 情報処理学会論文誌 Vol.54, No1, pp.330-339(2014).
[2] 石元慎太郎, 蜂巣吉成, 吉田敦, 桑原寛明, 阿草清滋 :『プログラミング演習における構文要素の種類毎のビューによるコーディング状況把握方法の提案』. 情報処理学会, 情報教育シンポジウム, 8pages(2018).
[3] 長谷川伸, 松田承一, 高野辰之, 宮川治 :『プログラミング入門教員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, No12, pp.3135-3149(2011).
[4] 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』. 電子情報通信学会第 18 回データ工学ワークショップ, pp.1–6, March(2007).
[5] 吉田敦, 蜂巣吉成, 沢田篤史, 張漢明, 野呂昌満 :『属性付き字句系列に基づくソースコード書き換え支援環境』. 情報処理学会論文誌, Vol.53, No.7, pp.1832-1849(2012).