プログラミング進捗把握のための学習者ソースコードのクラスタリング手法とクラスタ提示方法の提案
―制御文と演算子の並びに着目して―

2018SE019 池富蓮　2018SE095 上田遼太
指導教員：蜂巣吉成

1　はじめに

プログラミング演習では，指導ができる時間は有限であるので，指導者は学習者全体の進捗を把握し，全体指導することが好ましい．進捗把握する手段として，コンパイルや実行の結果を参考にする手法がある．しかし，演習時間内の編集途中であるソースコードはコンパイル可能な状態に達していない場合があり，コンパイル可能になるまでの支援も含めて進捗把握が必要となる．コンパイルや実行の結果以外には，ソースコードを直接見て比較しながら，学習者全体の進捗を把握していく手法がある．しかし，演習では指導者に対して学習者の数が多く，演習時間内で学習者のソースコードを見て進捗把握するのは難しい．

本研究の目的は，指導者がソースコードを直接見て行う方法での，学習者全体の進捗把握を支援する方法の提案である．進捗や中身が類似しているソースコードごとにクラスタリングを行い，クラスタ単位でクラスタ内の差分を考慮したソースコードを生成する．クラスタ単位でソースコードを指導者に提示し，指導者の見るソースコードの総数を減らす．

本研究の技術的課題は以下の通りである．

1. 類似する学習者のソースコードをどのような情報を利用することでクラスタリングすることができるか．
2. クラスタ内での差分抽出・提示をどのように行うか．

アプローチとしては，同値類分割や N-gram を特徴量とした Repeated-Bisection 法を用いて，ソースコードのクラスタリングを行う．このとき，学習者のソースコードを制御文と呼び出し関数名，演算子に注目しクラスタリングする．その後，クラスタごとに 1 つの代表的なソースを指導者に提示する．提示時には文字の濃淡を利用してクラスタ内の差分を表現する．

2　関連研究

石元ら [1] はソースコードから制御文，演算子，型，関数呼び出しを抽出し，同値類分割することで指導者が確認するソースコード数を減らしている．しかし，すべての要素を同値類分割すると，中身の差異が小さいソースコードも別物として扱われる．本研究では，制御構造で同値類分割，他の要素には N-gram を特徴量としたクラスタリングを用いることで，進捗や中身が類似するソースコードを同じクラスタとしてまとめ，指導者に提示するソースコード数を減らす．

3　学習者ソースコードの分析

本研究で扱う言語は C 言語とする．また，指導者から与えられた関数を記述する問題形式とする．他関数・関数名と型・引数については指導者から与えられていることとする．

指導者がどのような情報を利用し，ソースコードを分類するかを確認するために，サンプルを学生 2 人が指導教員 1 人と相談しながら手動で分類した．サンプルには，加藤ら [2] のものを利用した．学部 3 年生 14 人を対象とし，本研究と同一の問題形式である．内容は「実数の n 乗を計算する関数 (n は整数)」を作成する問題となっている．

手動での分析はソースコードをトップダウンで分類した．最初に分岐や反復といった制御構造の差異で分類した．次に計算手順の差異で分類を行った．今回の例題では主に n が正，0，負の場合分け (Listing 1)，n が 0 以上とそれ以外での場合分け (Listing 2, 3)，n が 0 以上のときのみ考え累乗計算を行うソースコード (Listing 4，5) に分類した．その後，n が負の場合，乗算で計算するもの (Listing 3) と除算 (Listing 2) で計算するものを分けた．大まかな計算方法の違いは，制御文本体で記述される演算子で見られた．Listing 4 に対して，Listing 5 は for 文の初期化式で用いられる演算子にミスが見られるが，どちらも n が正の場合の記述を行っていることになるので，進捗把握としては同じ組として扱う．

Listing 1　ソースコード A

```
double power(double x,int n)
{
  int i;
  double answer = x;
  if(n > 0){
    for(i = 0;i < n-1;i++){
      answer = answer * x;
    }
  } else if(n == 0) {
    answer = 1.0;
  } else {
    for(i = n+1;i < 0;i++){
      answer = answer * x;
    }
    answer = 1.0/answer;
  }
  return answer;
}
```

Listing 2　ソースコード B

```
double power(double x,int n)
{
  double ans = 1;
  int i;
  if( n >= 0) {
    for(i=0;i<n;i++) {
      ans *= x;
    }
  } else {
    for(i=0;i<n*(-1);i++) {
      ans /= x;
    }
  }
```

```
    return ans;
  }
```

Listing 3 ソースコード C

```
  double power(double x,int n)
  {
    int i;
    double y, ans=1.0;
    if(n>=0){
      for(i=0; i<n; i++){
        ans=ans*x;
      }
    } else {
      y=1/x;
      for(i=0; i<-n; i++){
        ans=ans*y;
      }
    }
    return ans;
}
```

Listing 4 ソースコード D

```
  double power(double x,int n)
  {
    int i;
    double ans;
    a=1;
    for(i=0;i<n;i++){
      ans=ans*x;
    }
    return ans;
  }
```

Listing 5 ソースコード E

```
  double power(double x,int n)
  {
    int i;
    double a;
    a=1;
    for(i<0;i<n;i++){
      a=a*x;
    }
    return a;
  }
```

最初に制御構造に注目したのは，制御文が 1 つあるかないかといった差異だけで，進捗や出力結果に大きな差異があると考えたからである．記述すべき if 文が 1 つ記述されていないだけで，そのソースコードは正答と比較するとまだ未完成であると判断できる．次に制御構造が同一であるものを詳しく見ると，反復文の本体で扱われている演算子によって，計算手順が異なる場合があった．そのため，制御構造に注目した後に，計算手順に注目して分類を行った．

我々が行う分類は客観性や妥当性が保証されていない．本研究で示す分類にある程度の妥当性があることを示すために，問題ごとに達成項目と達成度を考えた．用いたサンプルでは，「for 文を使って反復計算を行っているか」といった学習項目が考えられる．学習項目と達成度（記述途中，ミス，正答）をソースコードごとに分析し，達成項目とその達成度に加え，アプローチや使用する制御文を踏まえてサンプルを分類した．結果は，我々が行った手動分類の結果と一致した．よって，トップダウンで制御文から演算子へ注目しソースコードの分類を行う手法は，進捗把握のための分類として，ある程度妥当性があると判断した．

編集途中のソースコードについても分析を行った．編集途中のソースコードでは制御式のみで制御文本体が記述されていない例や，中括弧の対応が取れていない例が存在した．制御式のみの例と制御文本体まで記述されている例を見分けるには，制御式本体の演算子に注目する必要がある．中括弧の対応については，一定のルールで補完が可能だが，様々なルールで補完を試みた結果，すべてのソースコードに適用できるルールは見つからなかった．

4 提案手法

本研究で提案する手法の概略は以下のとおりである．

1. ソースコードのクラスタリングを行う．
2. クラスタ単位でソースコードを生成し指導者に提示する．

学習者のソースコードをクラスタリング後，クラスタごとに代表的なソースコードを生成し，生成したソースコードを指導者に提示する．最初にクラスタリング手法を説明し，その後クラスタ単位でソースコードを提示する方法を説明する．

4.1 ソースコードのクラスタリング手法

4.1.1 クラスタリングの方針と概略

本節では 3 節での分析を踏まえ，ソースコードをクラスタリングする際の方針を記述する．トップダウン形式でソースコードの制御文から演算子の順に着目する分類手法は，問題の達成項目や達成度の観点から，進捗把握のための分類として，ある程度の妥当性があると判断できる．このことからトップダウン形式でソースコードの制御文から演算子の順で注目していくことで，進捗把握に必要な分類ができる．本研究では最初に制御文に注目し，その後演算子に注目する 2 段階のクラスタリング手法を提案する．編集途中のソースコードでは中括弧の対応が取れない例が存在した．よって，構文木の構造ではなく，ソースコードの字句の並びに注目してクラスタリングを行う．

クラスタリングの概略は以下のとおりである．

1. 制御文の予約語と関数名の並びを利用した同値類分割
2. 制御式や制御文本体で扱われる演算子・EOF・NULL の並びを利用したクラスタリング

4.1.2 前処理

指導者が把握する必要のない差異を前処理で吸収する．変数名は統一し，コメントや余分な空白や改行は削除する．i++ や i=i+x，i=i+1 や i+=x といった記述は前者に統一する．石元 [3] の成果により，本研究ではループのカウンタ変数，返却値として扱う変数，配列の添字として扱われる変数名が統一されることとする．

4.1.3 制御文の予約語・関数名を利用した同値類分割

トップダウン形式で行う分類の第 1 段階として，制御文と呼び出されている関数名に注目した分類を行う．前処理後のソースコードから制御文の予約語と呼び出されている関数名の並びを抽出する．抽出した制御文の予約語と呼び出し関数名の並びが完全一致するものを同値とし，同値類分割を行う．例えば Listing 2 と Listing 3 は，制御文の予約語の並びを抽出すると，どちらも `if-for-else-for` となり，同じクラスタとなる．return 文については，編集途

中のソースコードでは記述を後回しにする場合や，単に書き忘れている場合が考えられる．そのため，予約語 return については並びを抜き出すことはしていない．

4.1.4 制御式・制御文本体の演算子・EOF・NULL の順序を利用したクラスタリング

同値類分割後は，ソースコードに記述される制御式や制御文本体の演算子に注目する．ソースコード中の制御式や制御文本体で扱われる演算子を N-gram として抽出し，Repeated-Bisection 法を用いてクラスタリングを行う．本研究ではクラスタリングツール bayon[4] を用いる．bayon は軽量で高速実行を目的とするツールで，学習者のソースコードを素早く分類したい本研究のユースケースにあっている．本研究では事前にプログラム群がどれくらいのクラスタに分かれるか把握できない．bayon では分割ポイントを設定することで，クラスタ数を指定せずクラスタリングを行うことができる．分割ポイントが小さいほどクラスタ数は多くなり，0 だと入力が完全一致するデータごとのクラスタが出力される．

全体の概略としては，最初にソースコードの演算子や NULL などから，N-gram を生成する．その後，N-gram で扱われている字句以前に使用されている制御文の予約語の並びを，N-gram の先頭に付与する．予約語が付与された N-gram を特徴量として，重みを付けて bayon への入力データとする．N-gram には以下の物を用いる．

- 制御式の演算子 (ただし for 文は条件式のみ)
- 制御文本体の演算子
- 制御式に記述される NULL や EOF
- 関数呼び出しの実引数で扱われる演算子

NULL や EOF については，記述を避けられない問題がある他，プログラミング演習において学習者が不適切に使用するケースが見られるので，N-gram の単位として抽出する．他にも変数の型や通常の定数を抜き出すことも可能だが，変数の型まで見ていると編集途中のソースコードでは，1 ソース 1 クラスタとなる可能性がある．また，制御式における定数の差異については提示方法を工夫することで対応可能である．

本研究では，制御式に記述される演算子や NULL・EOF は uni-gram として，他は bi-gram として抽出する．なお，bi-gram の先頭にはダミー (dummy) を設ける．Listing 1 の if 文内にある for 文から作成できる N-gram は表 1 の 2 列目である．N-gram がどの位置から抽出されたのかを区別するために，抽出以前に使用されている制御文の予約語の並びを，N-gram の先頭に付与する．先頭に予約語を付与したものは，表 1 の 3 列目である．

N-gram の重みについては，uni-gram は 1，bi-gram は 3 とする．ただし，N-gram に NULL や EOF の記述が含まれる場合は 5 とする．自由入力やリストを扱う問題では，EOF や NULL が記述されているか否かは進捗を把握する上で重要な要素である．よって，NULL や EOF は重みをつけて，重要度の高い分類としている．この重みに，登場回数を乗算したものを最終的な重みとする (表 1 の 4 列目)．

表 1 作成できる n-gram の例

演算子の位置	n-gram	n-gram(完成)	重み
制御式	<	if,for, <	1
制御文本体	dummy,=	if,for,dummy,=	3
	=,*	if,for,=,*	3

4.2 クラスタ単位の提示手法

クラスタ内に残った差分を抽出し，クラスタごとに代表ソースコードを提示する方法を説明する．予測される差分は，クラスタリング時に重みを小さくしている制御式周りだと考えられる．よって，制御式が記述されている行に注目して差分を提示する方法を説明する．

最初にクラスタ内から 1 つソースコードを選択し，制御式の行に記述されている字句を順番に記録していく．次に，記録した行とクラスタ内の残りのソースコードで対応する行を比較していく．記録された字句が，クラスタ全体で何回登場するかを記録する．字句の種類が同一でも別物としてカウントする場合がある．例えば for 文の初期化式に記述されている = と，条件に記述されている = は別物である．異なる字句が記録された字句と同じ役割・位置にあるときはその対応関係も記録する．例えば > が記録されたクラスタ内で，他のソースコードで同じ位置に < が記述されていたら > と < が同じ位置にあることと，その字句を使用している人数を記録しておく．

次に，記録からソースコードの提示を行う．クラスタからソースコードを 1 つ選択し，ソースコードをそのまま出力する．制御式の行のみ，記録した字句の並びを出力する．ただし，同じ位置に記録されている字句については，登場回数が多い字句のみを出力する．登場回数が多いほど濃く出力する．登場回数が少ない字句は補足情報として提示する．Listing 4，5 が同じクラスタになる前提で，提示例は図 1 のようになる．初期化式の=が薄く表示されていることから，=以外の演算子を誤って使用している学習者がいることが把握できる．補足情報として”<:1”といった文字列を表示している．これは該当箇所で=の代わりに < を記述している学習者が 1 名いることを示しており，初期化式で誤って < を使う学習者がいることを把握できる．

5 実験

5.1 実験方法

クラスタリング手法の評価と代表ソースコードが適切に生成されるかを確認するために，実際にサンプルに提案手法を適用した．サンプルは，「実数の n 乗を計算する関数を作成する問 1」，「ユークリッドの互除法を再帰関数で作

```
double power(double x,int n)
{
  int i;
  double a;
  a=1;
  for(i=0;i<n;i++){
      a=a*x;
  }
    return a;
}
```

図 1　提示例

成する問 2」,「文字列の走査をする問 3」,「文字列の長さを返す再帰関数を作成する問 4」である．解答者は学部生 14 から 39 人である．それぞれ完成版と編集途中のソースコードを用いて実験した．bayon の分割ポイントは経験則から 0.2 とした．

問題ごとに達成項目と達成度・アプローチを分析し，手動分類を行った．クラスタリング手法の評価は，達成項目に基づく手動分類の結果に，提案手法の結果が近づくかどうかで評価する．ソースコードのペアが同じクラスタに含まれているかを調べ，手動分類の結果を「正解」，提案手法による結果を「予測」として扱い，再現率・適合率を求める．ソースコードのペアについて，正解と予測で共通するペアを TP，正解と予測ともに存在しないペアを TN，予測のみに登場するペアを FP，正解のみに登場するペアを FN として，再現率・適合率を以下の式で算出する．

$$\text{再現率} = \frac{TP}{TP+FN}, \text{適合率} = \frac{TP}{TP+FP}$$

5.2　実験結果

評価指標の計算結果を表 2 に示す．なお，3 節で示した例は問 1 に対応しており，Listing 1，2，3，4 は別クラスタ，Listing 4，5 は同じクラスタになっている．

表 2　実験結果

問題	抽出時間	再現率	適合率	F 値 (調和平均)
問 1	編集途中	0.95294	0.85263	0.90000
問 1	完成版	1.00000	1.00000	1.00000
問 2	編集途中	1.00000	1.00000	1.00000
問 2	完成版	0.66667	1.00000	0.80000
問 3	編集途中	1.00000	0.87500	0.93333
問 3	完成版	1.00000	0.88889	0.94117
問 4	編集途中	0.94118	0.88889	0.91429
問 4	完成版	0.71429	0.71429	0.71429

6　評価と考察

6.1　クラスタリング手法の評価と考察

クラスタリング手法については，評価指標がおおむね 0.8 を上回っており，進捗把握の観点を用いた手動分類に近い分類を提案手法で行うことができた．一部，評価値が低くなっている問題があるが，これは単一の要素のクラスタが分類結果の多くを占めたことが原因である．用いた評価指標は単一の要素であるクラスタが多いと，数値が低く出るので，一部結果が悪く見えている．詳細に分析すると，進捗把握の観点を用いた結果と近い分類ができている．

クラスタリング手法の問題点として，デッドコードの演算子も特徴量として抽出する問題や，予約語の並びのみを見ているので，`if(){for(){}}`と `if(){}for(){}` のような例の差異を把握できない問題がある．今回，演習で用いる問題が複雑でないことから uni-gram と bi-gram を用いている．しかし，複雑な問題においては，同一の N-gram が複数回登場する場合があるので，tri-gram を特徴量としてクラスタリングすべきである．

6.2　クラスタごとの提示手法の評価と考察

クラスタごとの代表ソースコードを分析した結果，「反復回数の差異」,「初期化式の差異」といったクラスタリング時に重みの小さい部分で発生したクラスタ内の差分を表現できることを確認した．一方で，問題によってはクラスタ内の return 文の行に差異が存在した．問題や指導者の方針によって，return 文の行にも提示手法を適用するべきかどうかを，調整できるようにすべきである．

7　おわりに

本研究では，プログラミング演習におけるソースコードを直接見る進捗把握方法の支援を目的とし，ソースコードに対しクラスタリングを行う手法とクラスタごとの提示方法を提案した．実際の演習現場で適用した時の進捗把握への効果や，tri-gram の検討については今後の課題である．

参考文献

[1] 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, 情報教育シンポジウム論文集, vol.2018, no.22, pp.158-165 (2018).

[2] 加藤祐樹，加藤芳基：プログラミング演習における複数の観点を用いた指導が必要な学習者の特定方法の提案，2019 年度南山大学卒業論文 (2020).

[3] 石元慎太郎：プログラミング学習者の編集途中のソースコードに対するフィードバック方法の提案, 2019 年度南山大学修士論文 (2020).

[4] 軽量クラスタリングツール bayon, mixi engineer blog, 入手先 〈https://mixiengineer.hatenablog.com/entry/2009/10714/〉 (参照 2021-09-29).