# 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

M2018SE004 石元慎太郎
指導教員：蜂巣吉成

## 1 はじめに

大学などのプログラミング演習中に解法が分からずプログラムの作成を継続できなくなり，行き詰まり状態になる学習者がいる．そのような学習者に対して教員や TA が適切なフィードバックを行うことができればよいが，支援できる学習者の人数は限られており，学習者によっては長時間フィードバックを受けられないことがある．

解決策として，学習者への自動フィードバックシステムを利用する方法がある．システムは演算子が間違っている，文が不足しているなどの学習者特有の書き間違いの検出や，間違いの内容や状況に合わせ直接の答えではなくヒントを学習用のフィードバックとして提示する．プログラム依存グラフ (以下， PDG) を利用し正解パターンとの比較を行う方法 [1] や，機械学習によりデータフローの観点からソースコードを分析しフィードバックする方法 [2] [3] が提案されている．しかし，これらは提出した学習者自身は正答であると思っている完成に近いソースコードが対象であり，複数の文が欠け，間違った記述を含む編集途中のソースコードを用いた行き詰まりの状態の学習者への支援は考えられていない．

本研究では編集途中のソースコードにも対応する自動フィードバック方法を提案する．構文自体は理解しているものの，問題の解き方が分からない，どこを間違えているか分からない学習者を対象に解法のヒントとなるメッセージを提示する．学習者の記述の正誤を判断する材料として模範解答を利用する．模範解答は演習問題を作成する際にほとんどの場合で作成されるので，教員の負担が少ない．学習者のソースコードと模範解答の差分を求めることで，何が足りないか，余分か，間違っているかがわかる．また，差分より編集スクリプトを生成することで，どのように編集すれば正しいプログラムになるかがわかる．差分を求める方法としてバージョン間の差分理解支援ツール [4] があるが，学習用ではないので，これらのツールを用いるには以下の問題がある．

**問題点 1** バージョン間の差分理解支援ツールでは同じソースコードのバージョン違いなので変数名が一致するという前提があり，変数名を踏まえて差分をとるが，学習者と模範解答のソースコードでは変数名は異なるので期待する差分が得られないことがある．

**問題点 2** 生成される編集スクリプトは具体的な修正方法なので，学習者は機械的に間違いを直せるので学習には向かない．編集スクリプトはその短さを重視して生成するで，学習用のフィードバックには向かない操作が含まれることがある．

問題点 1 の解決のために学習者と模範解答の変数名をあらかじめ統一し差分を求める．問題点 2 の解決のために得られた差分から文を単位とする学習に適した編集スクリプトを生成し，あらかじめ模範解答の文に自然言語でその説明を付与しておくことで，どの文がどのように間違っているかをメッセージで提示できる．なお，行き詰まり状態の検出方法については本研究では対象外とし，学習者が支援を求めてきた場合にフィードバックを行う．コンパイラで支援できることから文法的なエラーは支援対象外とする．また，関数を作成する演習問題を想定し，関数名と引数は与えられるものとする．

## 2 関連研究

Marin ら [1] は演習問題に頻出するコード片にメッセージを付与した拡張 PDG をパターンとして用意し，提出された解答とのマッチングを行い，結果に応じてメッセージの出力を行う方法を提案している．パターンは問題間での再利用が想定され,「変数の代入と Return」,「累積加算」,「十進数の桁数を取り出す」などがある．

Lazer ら [2] は提出された Prolog のソースコードの AST から正解，不正解に寄与するパターンを機械学習を用いて見つけ，フィードバックに利用する方法を提案している．パターンは変数，もしくは定数間をつなぐ AST のパスとして表現される．複数のパターンを組み合わせたルールを用いて間違いと思われる箇所，足らない箇所を検出し，指摘する．Možina ら [3] はこの手法を拡張し，プログラムの制御構造や関数呼び出しも対象として Python でも同様のフィードバックを行う方法を提案している．

これらの研究に共通する問題点として，学習者の記述がほとんど終わっていることを前提としている点が挙げられる．データフローに依存する手法なので，複数の文が欠けている場合や，想定しない文が余分にある状態でのフィードバックは困難である．

Falleri ら [4] は人がソースコードを比較するときの行動に着目し，AST を用いて短い編集スクリプトを生成することでバージョンが違うソースコードの差分を得るツール，GumTree を提案している．バージョン間の差分理解支援なので学習用のフィードバックを行うには問題点 1, 2 で述べた問題がある．

## 3 学習者へのフィードバックの分析

### 3.1 学習者のソースコードの特徴

#### 3.1.1 変数

ソースコードを比較するには共通の変数に共通の処理を行っているか，という情報が有効である．バージョン間の差分理解支援 [4] でも変数名を考慮した差分検出が行われている．しかし，学習者が自身で宣言した変数名が模範解答と一致することは稀である．型を間違えること

も少なくない．また，本研究が対象とする演習問題は関数の実装を求めるものなので，学習者が自身で宣言する変数はせいぜい十数個である．

#### 3.1.2 学習者の間違え方の分類

学習者の編集途中のソースコードには演算子の間違いやまだ記述できていない文がある．学習者のソースコードと模範解答を文に着目して比較すると，学習者のソースコードの間違いは次の 4 種に分類できる．

1. 必要な宣言，文がない
2. 不要な宣言，文がある
3. 文で記述すべき式に間違いがある
4. 宣言，文の記述箇所が異なる

例えば，模範解答が Listing 1，学習者の解答が Listing 2 であるとき，この解答は，「sum=sum+arr[i]」などの「必要な宣言，文がない」間違いが 3 つ，「sum=0」と考えられる「b=0」と for 文の記述場所が逆になっているので「宣言，文の記述箇所が異なる」間違いを 1 つ含むと言える．

Listing 1 平均の計算 模範解答

```
1  double Average(int arr[], int size){
2      int i;
3      double sum;
4      double avg;
5      sum = 0;
6      for(i=0;i<size;i++){
7          sum = sum + arr[i];
8      }
9      avg = sum / size;
10     return avg;
11 }
```

Listing 2 文の欠落が複数含まれるソースコード例

```
1  double Average(int arr[], int size){
2      int a;
3      double b;
4      for(a=0;a<size;a++){
5      }
6      b = 0;
7  }
```

### 3.2 フィードバック

学習者に間違い箇所とその内容をフィードバックするときは，直接の修正方法ではなく解答のヒントを提示したい．Listing 2 の場合，「sum=sum+arr[i] を 5 行目に追加してください」というメッセージでは学習者は指示された文をそのまま追加するだけで修正できるので，学習用としては不適切である．「合計を計算する文がありません」のような，文と修正方法を抽象化したメッセージが必要である．本研究では，このような文の説明とその修正方法という形でのフィードバックを行う．このメッセージは間違いを含む文，その文がどのように間違っているかとその文を表す自然言語による文の説明の 3 つの要素から成り立っている．

文の位置を入れ替えるときは構成要素の少ない文を移動させたい．Listing 2 の「sum=0」場所の間違いを直すには，「sum=0」を移動させる，または for 文を移動させる方法があるが，for 文とその中身を移動させるよりも単純な「sum=0」を移動させたい．

## 4 フィードバック方法の提案

### 4.1 手法の概要

本研究では，模範解答と学習者の編集途中のソースコードを比較し，学習者の行き詰まり状態を解消できるフィードバックを行う方法を提案する．手法の概要を図 1 に示す．

学習者のソースコードと模範解答の差分検出には GumTree[4] を利用する．GumTree の小さな共通部分から全体の対応を考えるという方法は教員や TA が学習者のソースコードから間違いを探す方法と類似しており，学習用でも有効である．しかし，学習者のソースコードは「}」がないなどの途中で終わっている場合があるので，GumTree が解析できる形にソースコードを補完，正規化する．問題点 1 を解決し，GumTree の差分検出精度を上げるために学習者のソースコードと模範解答の変数名の対応をとり，模範解答の変数名を置き換える．変数の対応付けは原則として 1 対 1 の対応となるが，学習者が不要な変数を宣言したり，必要な変数がまだ宣言できていないことがあるので，1 対 0，0 対 1 の場合もある．

問題点 2 の解決のために編集スクリプトの生成は GumTree でも利用されている Chawathe のアルゴリズム [5] を基に学習用に適したアルゴリズムに改良したものを利用する．文単位のフィードバックを行うので，GumTree の出力結果である AST とその差分から文を表すノードを抽出し，文単位の編集スクリプトを生成する．

文単位の編集スクリプトと対応した自然言語で書かれた文の説明を合わせることで，どの文をどのように修正するかというメッセージを提示することができる．

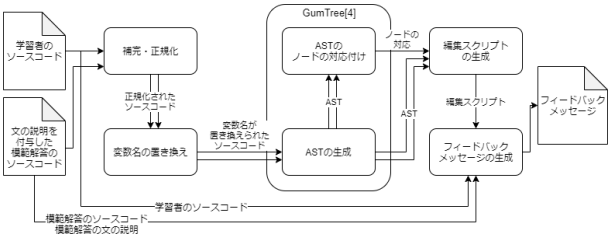


図 1 手法の概要

### 4.2 変数名の置き換え

3.1.1 節で述べたように学習者の変数名，型は間違っていることがあるので，変数の役割から変数を対応付ける．変数の役割はその変数が利用される文脈から，制御文との関係，代入される値，上記以外で特徴的な役割の 3 つに分類できる．様々な変数の役割が考えられるが，C 言語の入門書 [7] で紹介されているソースコードから特に多かった次の 6 つの役割を抽出した．

1. 制御に関連する役割
   (a) ループのカウンタ変数
   (b) if 文の条件式で参照し，if 文内で利用
2. 代入式から判断される役割
   (a) 累積計算結果で更新される
   (b) なにかを数えた結果で更新される
3. その他

(a) return 文で参照される
(b) 配列の添字である

入門書の 185 のソースコードで使われている変数は 477 個あり，その 47.6%の変数に上記の特徴のいずれかが当てはまった．これらの特徴を含まない変数の多くは main 関数での入出力に用いるものであり，本研究で想定する関数の作成を求める問題ではそのような役割を持つ変数は少ない．

役割による変数の対応付けは 2 段階で行う．1 段階目として変数を大まかな役割の方向性で特徴付け，対応を取る．例えば，ループのカウンタ変数という役割を持つ変数がどちらのソースコードにも 1 つだけならば，式に間違いがあったとしても対応付けられる．同じ役割の変数が複数あった場合，2 段階目としてより細かい粒度で特徴付け，対応をとる．例えば，多重ループのカウンタ変数という役割を持つ変数が複数ある場合は，その変数はどの深さの for 文で利用されているか，複数の for 文の制御に使われるかで比較し，対応付ける．

変数の役割はその特徴を表現したパターンが当てはまるかで評価する．例えば,「累積計算結果が格納される変数」であることは，図 2 の左のパターンで確認する．#はワイルドカードを表し，変数が□に当てはまるほど累積計算を保存する役割に近いといえる．2 段階目の評価パターンには「累積計算の初期化時にリテラルで初期化される」「入れ子になったループの外から 2 つ目のループのカウンタ変数である」などがある．

```
□=#;
#
for(#){
    #
    □=#□#;
    #
}
```

役割 2(a) の場合

```
for(#□#;#□#;#□#){
    #
}
```

役割 1(a) の場合

図 2 変数の役割を調べるパターン

各変数をその役割の評価結果を要素とするベクトルとして，ベクトルの向きの近さを評価するコサイン類似度を用いて学習者と模範解答の変数の類似度を求め，類似度を基に変数を対応付ける．$d$ 次元ベクトル $a, b$ の各次元の要素が $a_i, b_i$ で表されるとき，類似度は式 (1) で求められる．今回利用するベクトルは 0 以上の値を取るので，類似度は $0 \leq similarity_d(a,b) \leq 1$ の範囲となり，1 に近いほど類似していることを示す．1 段階目では各変数の役割を要素とする 6 次元のベクトルを用いて類似度を計算し，他に同じ変数を含む近い類似度の組がなければそれらの変数を対応付ける．複数の組の候補がある変数は 2 段階目として，詳細な特徴を評価した 28 次元のベクトルを用いた類似度により対応付ける．2 段階目の評価でも類似度が近い組があれば，2 段階目のベクトルのユークリッド距離を，それも一致したらソースコード中での変数の出現位置の近さにより対応付ける．1 段階目の類似度が一定値に満たない組は，不要な変数，もしくはまだ学習者が利用していない変数として対応付けない．

$$similarity_d(a,b) = \frac{\sum_{i=1}^{d} a_i b_i}{\sqrt{\sum_{i=1}^{d} a_i^2}\sqrt{\sum_{i=1}^{d} b_i^2}} \quad (1)$$

得られた変数の対応付けを基に模範解答の変数名を学習者の変数名に置き換える．

### 4.3 編集スクリプトの生成

GumTree により得られた AST とその差分から文を対象とするノードを抽出し，編集スクリプトを生成する．

編集スクリプトの生成には Chawathe のアルゴリズム [5] を改良したものを利用する．Chawathe のアルゴリズムは構造化されたデータを対象に「挿入」「削除」「更新」「移動」「整列」の 5 つの操作から編集スクリプトを生成する．ノードの移動操作を，異なる親へ移動させる「移動」と同じ親で順番を入れ替える「整列」の 2 つに分けて検出し，最終的には共に「移動」として編集スクリプトに追加する．整列を行う「整列フェーズ」ではノードの順番を編集後のソースコードの子に合わせて入れ替える．このとき，入れ替える回数を最小にするために編集前，編集後の子の並びの最長共通部分列 (以下，LCS) を 1 つ求め，編集前の子を先頭から走査し LCS に含まれないノードが見つかったらそれを適切な場所に移動させる．

「整列フェーズ」で移動させるノードは常にソースコード上で先に出現するノードなので，Listing 2 では「sum=0」ではなく for 文を移動させる編集スクリプトが生成される．そこで，本研究では「整列フェーズ」を変更し，ノードの構成要素が少ない文を優先的に移動させる．各ノードを根とする部分木に含まれるノードの数をコストとして，すべての LCS を用いてそれぞれ仮の編集スクリプトを生成し，最もコストが低い編集スクリプトを選択する．先の例では for 文よりも「sum=0」のほうがコストが低いので「sum=0」を移動させる．

## 5 評価

### 5.1 概要

以下の 2 点より問題点 1，2 が本研究の手法で解決し，学習者へのフィードバックが行えるのか評価した．

**評価 1** 変数の対応付けが行えるか

**評価 2** 学習用に適したフィードバックメッセージを生成できるか

変数が少ない単純な問題や，類似した役割の変数を用いる問題である「平均の計算」「行列の積の計算」「配列の最大値，最小値の計算」「分散の計算」の 4 問について，学習者の編集途中のソースコードを想定し模範解答から文を削除したものと編集途中のソースコードに学習者がよくする間違いを混入させたものの計 118 のソースコードを作成した．混入させる間違いは「初期化忘れ」「文の場所の間違い」「式の間違い」などである．変数の対応付けには構文解析器 TEBA[6] の解析結果を用いた．

### 5.2 評価 1

記述意図を踏まえた変数の対応が行えることを編集途中のソースコード，間違いを含むソースコードに分けて評価した．実際には変数名は学習者と模範解答で異なるが，今回は同じ役割の変数は同じ名前として，同じ名前の変数が対応するか検証した．

表 1 に結果を示す．表の期待通りの対応は同名の変数がすべて対応し，違う名前の変数が対応しなかったソースコードの数でであり，WM は間違った対応があるソースコード，NM は対応すべき変数が対応しなかったソースコード，WM and NM はその両方が含まれるソースコードの数を表す．表より，およそ 7 割のソースコードで期待通りの変数の対応付けが行えた．

表 1 検証 1 対応結果

| ソースコードの状態 | 問題 | 期待通りの対応 | WM | NM | WM and NM | 合計 |
|---|---|---|---|---|---|---|
| 編集途中 | 平均の計算 | 3 | 0 | 1 | 0 | 4 |
| | 行列の積の計算 | 4 | 0 | 0 | 0 | 4 |
| | 配列の最大値，最小値の計算 | 5 | 1 | 2 | 0 | 8 |
| | 分散の計算 | 9 | 1 | 2 | 0 | 12 |
| | 小計 | 21 | 2 | 5 | 0 | 28 |
| 間違いを含む | 平均の計算 | 16 | 6 | 0 | 0 | 22 |
| | 行列の積の計算 | 16 | 0 | 1 | 0 | 17 |
| | 配列の最大値，最小値の計算 | 15 | 7 | 2 | 1 | 25 |
| | 分散の計算 | 19 | 7 | 0 | 0 | 26 |
| | 小計 | 66 | 20 | 3 | 1 | 90 |
| | 合計 | 87 | 22 | 8 | 1 | 118 |
| | 割合 | 0.737 | 0.186 | 0.068 | 0.008 | 1.000 |

### 5.3 評価 2

評価 1 で得られた変数の対応付けを基に変数名を置き換えたソースコードを用いて期待するフィードバックが行えるのか検証した．様々なフィードバックが行えることを確認するために，各問題から異なるフィードバックとなるようなソースコードを 3 つずつ，合計 12 のソースコードを選んだ．それぞれのソースコードに対して「次に何を修正すべきか」という観点から指摘する内容を決定し，編集スクリプトと自然言語による文の説明からその指摘を行えるのか検証した．自然言語による文の説明は，「sum=0;//@合計を求める変数の初期化」のように文にコメントとして付与した．

10 のソースコードでは期待するフィードバックが生成できることを確認した．例えば，平均の計算で for 文の中をまだ書けていない状態を想定した解答では「sum=sum+arr[i]; の挿入」を意味するフィードバックを行いたい．このソースコードと模範解答を用いて生成された編集スクリプトには該当する操作が含まれており，文の説明より「合計を計算する文がありません」というメッセージを生成できることが確認できた．期待するフィードバックが行えなかったソースコードは，配列の最大値，最小値の計算で if 文の不等号を逆にしたものと，分散の計算で分散の計算を行う for 文を，和を求める for 文の中に記述したソースコードであった．

## 6 考察

5 章での評価 1 より，多くのソースコードで期待する変数の対応付けが行えた．また，ここで間違った変数の対応をした場合でも，不要な文として削除されることもあり，十分な精度で対応付けられたと考えられる．今回の検証で最大値，最小値の max と min が逆に対応することがあり，変数名に意味のある単語が使われている場合はシソーラスなどを利用して変数名を考慮して対応付ける方法が考えられる．

評価 2 について，今回フィードバックが行えなかった 2 つソースコードは min と max が逆に対応したことによるもの，GumTree によるマッチングが行えなかったものである．後者については学習用のノードのマッチングということを踏まえ，関数定義のブロックは一致するなどの制約を増やす方法が考えられる．

今回は学習者の解答がある模範解答を目指しているという前提のもと検証を行ったが，実際には複数の解法が存在する．解答のバリエーションへの対応策として，それぞれの解法に対応する模範解答をあらかじめ用意し，学習者の解法と近いものを用いて編集スクリプトを生成し，近いものを採用する方法が考えられる．

## 7 おわりに

本研究では行き詰まり状態の学習者へフィードバックを行うために，模範解答と学習者のプログラムの宣言，文を比較する方法を提案した．

今後の課題は，模範解答とのマッチング方法の考案，実際の学習者の解答を利用した検証である．

## 参考文献


[1] Marin, J.V., Pereira,T., Sridharan, S. and Rivero, R.C.: “Automated Personalized Feedback in Introductory Java Programming MOOCs”, IEEE 33rd International Conference on Data Engineering, pp.1259–1270(2017).

[2] Lazar, T., Možina, M. and Bratko, I.: “Automatic Extraction of AST Patterns for Debugging Student Programs”, Artificial Intelligence in Education, Vol.10331, pp162–174(2017).

[3] Možina, M. and Lazar, T.: “Syntax-based analysis of programming concepts in Python”, Artificial Intelligence in Education,Vol.10948, pp.236–240(2018).

[4] Falleri, J., Morandat, F., Blanc, X., Martinez, M. and Monperrus, M.: “Finegrained and Accurate Source Code Differencing”, Proceedings of the 29th ACM/IEEE international conference on Automated software engineering, pp.313–324(2014).

[5] Chawathe, S.S., Rajaraman, A. , Garcia-molina, H. and Widom, J.: “Change Detection in Hierarchically Structured Information”, In Proceedings of the ACM SIGMOD International Conference on Management of Data, pp.493–504(1996).

[6] 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)．

[7] 柴田望洋：“新・明解 C 言語 入門編”，SB Creative(2014)．