一般社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report
SS2023-78(2024-03)

プログラミング演習における編集中のソースコードに対する自動フィードバック方法の考察

模範解答プログラムの編集遷移グラフを用いた誘導型フィードバック

澤田　侑希†　蜂巣　吉成††　吉田　敦††　桑原　寛明††

† 南山大学大学院理工学研究科　〒466–8673 愛知県名古屋市昭和区山里町 18
†† 南山大学理工学部　〒466–8673 愛知県名古屋市昭和区山里町 18

あらまし　プログラミング演習において，一部の学習者にとって自力でソースコードを完成させることは困難である．自動フィードバックの研究は多くされているが，完成に近いソースコードを対象としており，編集中のソースコードを対象としたものは少ない．本研究では，学習者が自身でコードを考えて完成させる能力を養うために，編集中のソースコードを対象とした自動フィードバック方法を提案する．模範解答プログラムから編集遷移グラフを作成し，学習者のソースコードをいずれかのノードに対応付け，誘導型フィードバックを提示する．自動フィードバックシステムを試作し，学習者 12 人のソースコードを適用することで，提案方法の有効性を評価した．
キーワード　プログラミング演習，自動フィードバック，編集中ソースコード

Considerations of an Automatic Feedback Method for a Learner's Editing Source Codes in Programming Exercises

Guided Feedback Using Edit Transition Graphs of a Model Answer Program

Yuki SAWADA†, Yoshinari HACHISU††, Atsushi YOSHIDA††, and Hiroaki KUWABARA††

† Graduate School of Science and Engineering, Nanzan University　18 Yamazato, Showa, Nagoya, 466–8673 Japan
†† Faculty of Science and Technology, Nanzan University　18 Yamazato, Showa, Nagoya, 466–8673 Japan

Abstract　In programming exercises, it is difficult for some learners to complete source codes by themselves. Many researches on automatic feedback methods has been proposed, but these studies primarily target source codes that is almost completion. There has been little research on editing source codes. We propose an automatic feedback method for a learner's editing source codes. The purpose of the feedback is to cultivate learners' ability to think through and complete the source codes. The learner's source code is mapped to one of the nodes in the edit transition graph, and guided feedback is provided to the learner. We implemented a prototype of the automatic feedback system to evaluate our method. We evaluated the effectiveness using the source codes of 12 learners.
Key words　programming exercise, automatic feedback, editing source code

1. はじめに

プログラミング演習において，学習者は与えられた演習問題に取り組むが，一部の学習者にとって自力でソースコードを完成させることは難しい．学習者が行き詰まった場合は教員や TA が支援するが，対応できる人数には限りがある．自動フィードバックシステムが実現できれば，対応できる人数の制限がなくなり，待ち時間の発生を防げる．編集中のソースコードに対するアプローチとして，大規模言語モデル（LLM）を用いた ChatGPT[1]などの生成 AI チャットサービスの利用が考えられるが [1]，解答をそのまま提示してしまうことや，ハルシネーションなどの問題がある．

本研究では，学習者が自身でコードを考えて完成させる能力を養うために，編集中のソースコードに対する自動フィードバック方法を提案する．編集中のソースコードを対象とするには，学習者のソースコードの編集状態を特定し，編集状態ごと

(注1)：https://openai.com/blog/chatgpt

に追加させるコードを決めておく必要がある．技術的課題は次の 2 つである．

（1） 学習者のソースコードの編集状態を特定する手法

（2） 編集状態ごとに追加するコードを決める手法

本研究における編集状態とは，ソースコードをセグメント [2]（4.1 節）と呼ぶコード片の組合せとして定義したものである．模範解答のプログラム（以下，模範解答プログラム）を構成するセグメントのすべての組合せを模範解答の状態として用意し，模範解答の状態にセグメントを 1 つずつ追加していく過程を編集遷移と捉え，模範解答の状態をノード，編集遷移をエッジとして編集遷移グラフを作成する．模範解答プログラムの編集遷移グラフを作成し，学習者のソースコードをいずれかのノードに対応付け，対応付いたノードに合わせるためのフィードバックと，遷移先のノードに応じたフィードバックを提示する．フィードバックを提示することで，学習者に誤りを確認させ，コードを追加させて次のステップへ誘導する．

2. 関 連 研 究

自動フィードバックに関する研究は多くされているが，主に誤りの特定に焦点を当てており，プログラムの修正や次のステップへの誘導に焦点を当てている提案は少ない [3]．対象は完成に近いソースコードであることが多く [4] [5]，編集中のソースコードに焦点を当てた手法 [6] は少ない．Kaleeswaran らは学習者の提出プログラムをクラスタリングし，誤りプログラムと正解プログラムを対応付けて自動フィードバックを実現している [4]．正解プログラムを用いてフィードバックを生成するので，編集中のソースコードに適用すると，複数のコードの追加を同時に指示することになる．模範解答の状態を正解プログラムとして使用できるとしても，編集中のソースコードには書き方が無数に存在するので，期待通りに分類できるとは限らない．Zimmerman らは学習者のソースコードと模範解答プログラムの pq-gram [7] を用いることで，挿入と削除のセットを生成して自動フィードバックを実現している [5]．pq-gram は木構造から高さ p，幅 q として各ノードのラベルを一列に並べたラベルタプルを抽出する手法であり，ラベルタプルの集まりを pq-gram プロファイルと呼ぶ．pq-gram 距離は 2 つの木のラベルタプルの共有度に基づいて，木が似ているかどうかを評価する指標である．模範解答の状態を用いて編集中のソースコードに適用すると，模範解答の状態に合わせるフィードバックを提示できるかもしれないが，コードを追加させるフィードバックを実現するには，学習者のソースコードに最小限のコードを追加した模範解答の状態を特定する必要がある．特定できたとしても，複数存在する場合，どの状態を用いると学習者にとって効果的か判断することは難しい．

誤り箇所特定や自動プログラム修正を利用し，解答を提示せずに修正方法を学習者にフィードバックできれば，学習者の支援に有効である．SBFL [8] はテストケースによる実行経路の情報を用いて誤り箇所を推定する手法であり，Araujo らは学習者のソースコードへの SBFL の適用を調査し，その利点と欠点について議論している [9]．Li らは学習者のコードの誤りを分類し，分類ごとの誤り頻度に基づいて SBFL で算出された疑惑値を重み付けることで，誤り箇所の推定方法を提案している [10]．我々はセグメントを用いることで，別解を考慮した誤り範囲の特定方法を提案した [2]．本研究では，フィードバックを参考にして学習者自身に誤りの有無を確認させる．

ソースコード 1　模範解答プログラム

Listing. 1　Model answer program

```
1  void strdel(char *s, char c)
2  {
3    char *t = s;
4    while (*s != '\0') {
5      if (*s != c) {
6        *t = *s;
7        t = t + 1;
8      }
9      s = s + 1;
10   }
11   *t = '\0';
12 }
```

ソースコード 2　学習者のソースコード

Listing. 2　Student's source code

```
1    *t = s;
2    while (*s == '\0') {
3    }
```

ソースコード 3　模範解答の状態 1

Listing. 3　Model state 1

```
1    *t = s;
2    while (*s != '\0') {
3    }
```

3. 問 題 分 析

ソースコード 1 は文字列から文字を削除する関数を作成させる演習問題の模範解答である．本研究では変数宣言を対象外とするので，3 行目は「`char *t;`」「`*t = s;`」に分割して扱う．ソースコード 2 は，while 文の条件式に誤りがあるような学習者のソースコードの例である．自動フィードバックは模範解答との比較で実現されることが多く [11]，ソースコード 2 をソースコード 1 との差分を用いてフィードバックすることを考えると，ソースコード 1 の 5，6，7，9，11 行目のコードを追加させることになる．一度に多くの編集を指示することになるので，学習者にとって効果的でない．ソースコード 1 とソースコード 2 の差分では，while 文の条件式以外の部分でも差分がある．そこで，模範解答プログラムを一定の大きさに分割し，すべてのコード片の組合せを模範解答の状態として用意する．模範解答を文単位で分割した場合の，初期化式と while 文のみ書かれた模範解答の状態をソースコード 3 に示す．ソースコード 2 とソースコード 3 との差分であれば，for 文の条件式に誤りがあることを特定できる．学習者のソースコードと最も近い模範解答の状態を対応付けられれば，編集状態を特定できる可能性がある．初期化式と while 文の間に余分な記述があるような学習者のソースコードでは，余分な部分のみ削除かコメントアウトするように指示したい．余分な記述がある場合，模範解答の状態との差分の大きさだけでソースコード 3 と対応付けるのは難し

ソースコード 4　模範解答の状態 2

Listing. 4　Model state 2

```
1  *t = s;
2  while (*s != '\0') {
3    if (*s != c) {
4    }
5  }
```

い．ソースコードの編集状態を特定するには，単にソースコードの差分を確認するだけでは不十分であり，制御構造の分析が必要がある．

学習者のソースコードがソースコード 3 と同じ状態の場合，if 文を追加するように指示したい．ソースコード 3 に if 文を追加した模範解答の状態をソースコード 4 に示す．自動で追加を指示するメッセージを提示する場合，編集状態を特定できたと仮定すると，書かれていないいずれかのコードを 1 つ選ぶことになる．教員が支援する場合，教育意図や演習問題の特性に合わせて指示を変えれば良いが，自動フィードバックを実現するには，編集状態ごとに追加させるコードを事前に決めておく必要がある．

これらを踏まえて，編集中のソースコードに自動でフィードバックするためには，(1) 学習者のソースコードの編集状態を特定し，(2) 編集状態ごとに追加するコードを決める必要がある．

4. 提 案 方 法

4.1 概　　略

本研究ではソースコードをセグメント単位で扱う [2]．セグメントの定義を定義 1 に示す．

定義 1　セグメント

（1） 制御文の本体（文）はセグメントである．ただし，複合文の波括弧は除く．

（2） 制御文の予約語と制御式を囲む丸括弧の部分はセグメントである．

（3） 1，2 以外の宣言または文の連続はセグメントである．

（4） 制御文の入れ子の場合は 1〜3 のセグメントも入れ子になるが，このうち最内のものをセグメントとする．

ソースコード 1 は変数宣言を除くと，「`*t = s;`」「`while (*s != '\0')`」「`if (*s != c)`」「`*t = *s; t = t + 1;`」「`s = s + 1;`」「`*t = '\0';`」のように 6 個のセグメントに分割する．

模範解答の理想的な編集過程を基に編集遷移グラフを作成する．模範解答の状態の定義を定義 2 に，編集遷移グラフの定義を定義 3 に示す．

定義 2　模範解答の状態

模範解答のセグメントのすべての組合せであり，それぞれの状態が 1 つのテキストとなる．

定義 3　編集遷移グラフ

模範解答の状態がノード，編集遷移がエッジであり，模範解答プログラムを終端ノードとする単方向の有向グラフである．終端ノード以外のすべてのノードから 1 つだけエッジが存在し，模範解答プログラムを根と捉えると，エッジが根に向かう木構造になる．

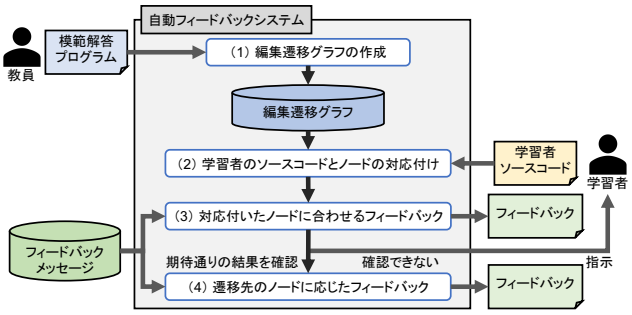

図 1　自動フィードバックシステムの処理の流れ

Fig. 1　Process flow of the automatic feedback system

編集遷移グラフを用いて学習者のソースコードに最も近い模範解答の状態を対応付けることで，編集状態を特定できる．対応付いたノードの遷移先を明らかにすることで，学習者に追加させるコードを選択できる．

技術的課題（1）を解決するために，学習者のソースコードと編集遷移グラフのすべてのノードから抽象構文木（AST）の pq-gram を生成し，すべてのノードとの pq-gram 距離を計算することで，距離が最も近いノードを選択する．pq-gram 距離は 2 つの木のラベルタプルの共有度に基づいて木が類似度を評価する指標であり，制御構造の情報を含めてソースコードの差分を比較できるので，編集状態の特定に有効である．技術的課題（2）を解決するために，すべての模範解答の状態にて，次の状態となる模範解答の状態を選択する．

解答をそのまま提示しても学習効果が乏しいので，フィードバックメッセージとして追加するコードのヒントや編集の確認方法を用意する．フィードバックの定義を定義 4，5 に示す．

定義 4　追加のためのフィードバック

コードを追加させるためのヒント（制御文の種類と役割，または式文の役割）を含んだフィードバックのこと．

定義 5　確認のためのフィードバック

制御文の処理や，式文の左辺値を確認させるための printf 関数と，その表示内容を含んだフィードバックのこと．

教員はテンプレートを用いてフィードバックメッセージを作成する．模範解答プログラムに一部をコメントとして記述し，それぞれセグメントごとに 1 つずつ用意する．

図 1 は自動フィードバックシステムの処理の流れであり，提案方法は 4 つのプロセスからなる．教員は演習問題を作成する際，模範解答プログラムとフィードバックメッセージを用意し，システムに入力する．システムは模範解答プログラムを基に編集遷移グラフを生成する．学習者は演習問題に行き詰まった場合，システムに編集中のソースコードを入力する．システムは学習者のソースコードと編集遷移グラフ内で最も近いノードを対応付け，コードの確認を促すフィードバックを提示する．学習者はフィードバックを参考にソースコードを修正し，期待通りの実行結果になったことを確認する．確認できた場合，システムは遷移先のノードに応じたコードの追加を促すフィードバックを提示する．確認できない場合，再度フィードバックを

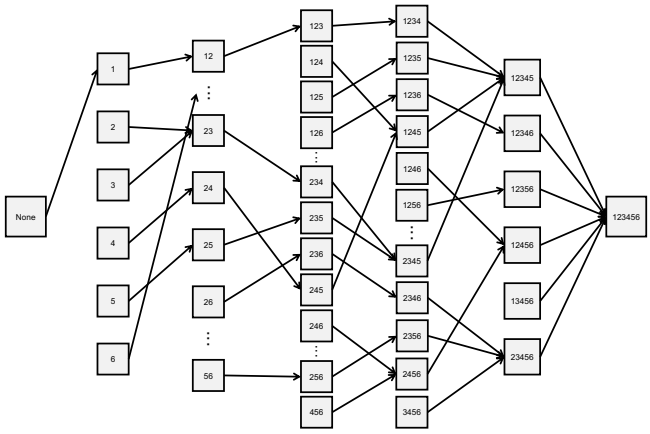

図 2　編集遷移グラフ

Fig. 2　Edit transition graph

要求するか，教員や TA に支援を求めるように指示する．

4.2　編集遷移グラフの作成

学習者のソースコードの編集状態を特定するために，編集遷移グラフのノードとなる模範解答の状態を生成する．ソースコード 3 は模範解答の状態であり，ソースコード 1 の 1，2 番目のセグメントの組合せである．模範解答が n 個のセグメントから成るとすると，模範解答の状態は何も書かれていない状態を含めて 2^n 個であり，ソースコード 1 では，ソースコード 1，3，4 を含む 64 個の模範解答の状態がある．else if，else はセグメント間で依存関係があるので，模範解答の状態に単体で出現する場合は例外として if に置き換える．模範解答の状態を AST に変換し，AST の pq-gram を生成する．

学習者に追加させるコードを決めるために，編集遷移グラフのエッジとなる編集遷移を選択する．編集遷移グラフは，模範解答プログラムを終端ノードとし，その他すべてのノードから終端ノードへの経路が一意に存在する．図 2 はソースコード 1 の編集遷移グラフである．グラフが大きいので，2 番目のセグメント以外は省略している．編集遷移とは，遷移元ノードにセグメントを 1 つ追加する過程であり，遷移元ノードが m 個のセグメントから成るとすると，編集遷移は $n - m$ 個存在する．遷移元ノードからのエッジは，編集遷移への重み付けにより 1 つだけ選択する．学習者がソースコードを編集する過程において，計算手順を自身で考えてコードを書かせたいが，学習者は試行錯誤しながら計算手順を考えるので，その順序は自明ではない．すべての模範解答の状態においてエッジを 1 つだけ選択しなければならないので，実行結果を確認できる状態へ導くことを優先するように，変数の定義と使用の関係，制御文の入れ子の関係を利用して編集遷移に重みを付け，重みが最大となる遷移をエッジとして採用する．初期化式や return 文，先に出現する制御文などを優先して追加させられるように，模範解答プログラムのセグメント間にポイントを設定し，遷移元に存在するセグメントと，遷移先候補で追加されるセグメントの情報を用いてポイントを割り当て，ポイントの和を重みとして遷移先の選択を実現する．

4.3　学習者のソースコードとノードの対応付け

学習者がフィードバックを要求したときに，学習者のソースコードと編集遷移グラフのノードの対応付けを開始する．学習者のソースコードの表記の揺れを削減するような正規化の処理をし，学習者のソースコードから AST の pq-gram を生成する．学習者のソースコードの pq-gram プロファイルと 4.2 節で用意したすべての模範解答の状態の pq-gram プロファイルで pq-gram 距離を計算し，最小距離のノードを選択する．距離が同じノードが複数存在する場合は模範解答プログラムにおけるセグメントの出現順に基づいてトップダウンに選択する．

ソースコード 2 は学習者のソースコードの例であり，ソースコード 3 が理想のマッチングである．AST から変数のノードを削除した場合のソースコード 2，3 の pq-gram 距離は 0.48 であり，ソースコード 1 から生成される編集遷移グラフのすべてのノードで最小距離となる．pq-gram の特性上，AST のノード数に距離が依存するので，理想のマッチングを得るのは難しい．AST を簡略化して精度を向上させることを目的として，AST の簡略化方法を **5.** で評価する．

4.4　対応付いたノードに合わせるフィードバック

学習者のソースコードに誤りがある場合は修正させたいので，対応付いたノードに合わせるために，ノードを構成するすべてのセグメントに応じた確認のためのフィードバックを提示し，期待通りの実行結果が確認できたか学習者に質問する．学習者はフィードバックを参考にソースコードを修正し，期待する実行結果が確認できたら，確認できたことをシステムに伝える．

ソースコード 2 に対応付いたノードはソースコード 3 であり，ソースコード 3 を構成するセグメントは「`*t = s;`」「`while (*s != '\0')`」である．学習者に提示するフィードバックを次に示す．

> ポインタ変数に初期値を代入する式文の直後に `printf("*t : %c\n", *t);` を記述し，Nanzan，z と入力して，期待通りの実行結果であるか確認してください．
>
> 文字列を走査する繰返しの制御文の処理に `printf("繰返しの制御文を通りました．\n");` を記述し，Nanzan，z と入力して，6 回表示されることを確認してください．
>
> 実行結果を確認できましたか？（y/n）：

4.5　遷移先のノードに応じたフィードバック

学習者のソースコードを次のステップに向かわせるため，遷移先のノードで追加されるセグメントに応じた追加のためのフィードバックと確認のためのフィードバックを提示する．一連のフィードバックによりソースコードが次のステップに進んだ後，学習者は自力で演習問題を解き進め，再度行き詰まったらフィードバックを要求する．フィードバックを参考にしても行き詰まりを解消できない学習者は，教員や TA が対応する．

ソースコード 3 の遷移先のノードはソースコード 4 であり，ソースコード 4 で追加されるセグメントは「`if (*s != c)`」である．学習者に提示するフィードバックを次に示す．

ソースコード 5　演習問題（2）の模範解答プログラム

Listing. 5　Model answer program of exercises (2)

```
1  void triangle(int n)
2  {
3    int i, j;
4
5    for (i = 1; i <= n; i = i+1) {
6      for (j = 1; j <= n-i; j = j+1) {
7        putchar(' ');
8      }
9      for (j = 1; j <= i; j = j+1) {
10       putchar('*');
11     }
12     putchar('\n');
13   }
14 }
```

実行結果を確認できましたか？（y/n）：y

削除しない文字か判定するための分岐の制御文を追加してください．

削除しない文字か判定する分岐の制御文の処理に `printf("分岐の制御文を通りました．\n");` を記述し，Nanzan，z と入力して，表示されることを確認してください．

5. 評　　価

AST を 3 つの方法で簡略化し，ソースコード対応付けの精度を評価する．精度が高い簡略化方法を使用して自動フィードバックシステムを試作し，学習者のソースコードを適用することで，提案方法の有効性を評価する．これらの評価を行うため，南山大学理工学部の学部 3 年生 12 人を対象として演習問題に取り組ませ，編集中のソースコードを一定の時間や行動に応じて保存したものを利用する．連続する 2 つで編集がないものを学習者が演習問題に行き詰まった状況として抽出し，評価の対象として扱う．本研究で使用する演習問題は，（1）文字列から文字を削除する関数（ソースコード 1），（2）＊を階段式に表示する関数（ソースコード 5），（3）＊をダイヤモンドの形に表示する関数（ソースコード 6）である．

pq-gram の特性上，共有しないラベルタプルが多いほど距離が大きくなるので，AST のノード数に距離が依存する．制御文に着目すると，一般にノード数が多い for 文では，別解や誤りによるラベルタプルの差異の影響を受けやすい．次の 3 つの方法で AST を簡略化することで，対応付けの精度が向上するか確認する．簡略化方法は次の 3 つである．

（1） 変数のノードを削除

（2） 変数，オペレータのノードを削除

（3） 変数，仮引数，リテラル，オペレータのノードを削除

学習者のソースコードと模範解答に同じ制御文が出現する場合，それらを容易に対応付けるため，予約語以外のノードを可能な限り削除したい．簡略化方法（1）はベースラインであり，変数の対応付けが困難なので，変数を削除する．簡略化方法（2）では，同じ制御文が模範解答に複数出現する場合，それらを判別可能にしたい．仮引数やリテラルは各セグメントで固

ソースコード 6　演習問題（3）の模範解答プログラム

Listing. 6　Model answer program of exercises (3)

```
1  void diamond(int n)
2  {
3    int i, j;
4
5    for (i = 1; i <= n; i = i+1) {
6      for (j = 1; j <= n-i; j = j+1) {
7        putchar(' ');
8      }
9      for (j = 1; j <= 2*i-1; j = j+1) {
10       putchar('*');
11     }
12     putchar('\n');
13   }
14   for (i = n-1; 0 < i; i = i-1) {
15     for (j = 1; j <= n-i; j = j+1) {
16       putchar(' ');
17     }
18     for (j = 1; j <= 2*i-1; j = j+1) {
19       putchar('*');
20     }
21     putchar('\n');
22   }
23 }
```

ソースコード 7　演習問題（2）の評価対象

Listing. 7　Source code for evaluation of exercises (2)

```
1    for (i = 0; i < n; i = i+1) {
2      for (j = 0; j < n; j = j+1) {
3        printf("*");
4      }
5      printf("\n");
6    }
```

ソースコード 8　ソースコード 7 の理想のマッチング

Listing. 8　Ideal matching of Listing.7

```
1    for (i = 1; i <= n; i = i+1) {
2      for (j = 1; j <= i; j = j+1) {
3        putchar('*');
4      }
5      putchar('\n');
6    }
```

有のものが書かれることが多いが，オペレータは別解や誤りの影響を受けやすいと仮定し，オペレータを削除する．簡略化方法（3）では，制御文と式文のノード数の偏りをなくすため，仮引数，リテラル，オペレータを削除する．

各演習問題において模範解答から大きく逸脱していないと判断した 5 つのソースコードを選び，評価対象とすべての模範解答の状態で距離を算出した．本研究では，理想のマッチングとの距離が近くなる簡略化方法が良いと仮定する．ただし，距離が近ければ良いと一概には言えない．ソースコード 7 は演習問題（2）の評価対象ソースコードであり，理想のマッチングであるソースコード 8 との距離はそれぞれ 0.74，0.47，0.72 である．簡略化方法（1），（3）はソースコード 8 と対応付くが，簡略化方法（2）はソースコード 9 と対応付く．簡略化方法（2）だと距離は最も近くなるが，理想のマッチングにはならない．距離の平均を表 1 に示す．表 1 より，模範解答プログラムに同じ制御文が複数出現する場合は簡略化方法（2），複数出現しない場合は簡略化方法（3）で距離が近くなる傾向がある．有効性の評価では，演習問題ごとに傾向に合わせて簡略化方法を採用する．

有効性を評価するために，自動フィードバックシステムを

ソースコード 9　簡略化方法（2）で対応付く状態

Listing. 9　Model state that corresponds to simplification method (2)

```
for (i = 1; i <= n; i = i+1) {
  for (j = 1; j <= n-i; j = j+1) {
    putchar(' ');
  }
  putchar('\n');
}
```

表 1　簡略化方法の実験結果

Table 1　Results of the AST simplification methods

対象	簡略化方法（1）	簡略化方法（2）	簡略化方法（3）
演習問題（1）	0.81	0.62	0.58
演習問題（2）	0.71	0.42	0.76
演習問題（3）	0.82	0.51	0.81

表 2　有効性の実験結果

Table 2　Results of the effectiveness of our method

対象	フィードバック[4]	フィードバック[5]
演習問題（1）	50.0%	68.2%
演習問題（2）	69.7%	70.8%
演習問題（3）	38.7%	37.9%

試作した．AST の生成は TEBA[2]を利用し，pq-gram の生成は Augsten ら [7] の pqgrams[3]を利用した．3 つの演習問題のソースコードを適用し，出力されるフィードバックメッセージが期待通りになるか確認する．対応付いたノードに合わせるフィードバックで期待するのは，編集遷移グラフのノードに適切にマッチングした場合のフィードバックである．遷移先のノードに応じたフィードバックで期待するのは，ソースコードを見て教員がするであろうフィードバックである．評価対象となるソースコードは，演習問題（1）で 22 個，演習問題（2）で 33 個，演習問題（3）で 31 個である．実験結果を表 2 に示す．

6. 考　　察

有効性の評価より，演習問題（1），（2）では半数以上のソースコードに対して自動フィードバックを実現できた．pq-gram 距離が大きく離れている場合，期待通りのマッチングになりにくいことが明らかになった．距離に閾値を設け，閾値以上の場合はフィードバックを提示せずに教員が対応すれば，誤ったフィードバックで学習者を混乱させることを防げる．フィードバックの精度は閾値を 0.60 とすると，演習問題（1）では 60.0%，60.0%，演習問題（2）では 77.8%，73.7%，演習問題（3）では 52.2%，47.6% となる．理想のマッチングにならなかったソースコードの特徴は余分な記述がある，誤りや書きかけのコードがある，構造が異なる，簡略化の影響を受ける，別解があるが挙げられる．

学習者のソースコードと編集遷移グラフのノードを対応付けるために pq-gram を利用したが，一部の演習問題ではマッチングの精度が期待に満たない結果となった．期待通りでないマッチングの多くは，類似した制御構造が複数出現する場合に，それらの判別が困難なことが原因である．

遷移先のノードに応じたフィードバックは，マッチング精度に依存することが明らかになった．遷移先の選択に編集遷移の重みを用いたが，より効果的にするには，多数の演習問題の特徴を分析し，最適な指針を定める必要がある．マッチング精度を改善し，妥当な重みを付ければ，高い割合で期待するフィードバックを出力できる可能性がある．

7. おわりに

本研究では，編集中のソースコードに対する自動フィードバック方法を提案した．自動フィードバックシステムを試作し，学習者のソースコードを用いて評価したことで，提案方法が学習者支援に有効である可能性を示した．今後の課題は，提案方法の改善による精度の向上，フィードバックメッセージの自動生成，ユーザビリティテストによる学習効果の評価である．

謝辞　本研究の一部は JSPS 科研費 23K11359，2023 年度南山大学パッヘ奨励金 I-A-2 の助成を受けた．

（注2）：http://tebasaki.jp/src/

（注3）：https://github.com/thdiaman/pqgrams

（注4）：対応付いたノードに合わせるフィードバック

（注5）：遷移先のノードに応じたフィードバック

文　　献

[1] N. Kiesler, D. Lohr, H. Keuning : Exploring the Potential of Large Language Models to Generate Formative Programming Feedback, 2023 IEEE Frontiers in Education Conference, pp.1–5, 2023.

[2] 澤田侑希，梅田祐一郎，蜂巣吉成，吉田敦，桑原寛明：プログラミング演習におけるセグメントを用いたソースコードの誤り範囲特定方法の提案，コンピュータソフトウェア，Vol.40，No.4，pp.4_29–4_36，2023.

[3] H. Keuning, J. Jeuring, B. Heeren : A Systematic Literature Review of Automated Feedback Generation for Programming Exercises, ACM Transactions on Computing Education, vol.19, No.1, Article 3, pp.1–43, 2018.

[4] S. Kaleeswaran, A. Santhiar, A. Kanade : Semi-supervised Verified Feedback Generation, 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp.739–750, 2016.

[5] K. Zimmerman, C. R. Rupakheti : An Automated Framework for Recommending Program Elements to Novices, 2015 30th IEEE/ACM International Conference on Automated Software Engineering, pp.283–288, 2015.

[6] 蜂巣吉成，石元慎太郎，吉田敦，桑原寛明：プログラミング学習者の編集途中のソースコードと模範解答における変数の対応づけ方法の提案，ソフトウェア工学の基礎 XXVII（FOSE 2020），pp.109–114，2020.

[7] N. Augsten, M. Bohlen, J. Gamper : Approximate Matching of Hierarchical Data Using pq-Grams, in VLDB, pp.301–312, 2005.

[8] W. E. Wong, R. Gao, Y. Li, R. Abreu, F. Wotawa : A Survey on Software Fault Localization, IEEE Transactions on Software Engineering, Vol.42, No.8, pp.707–740, 2016.

[9] E. Araujo, M. Gaudencio, D. Serey, J. Figueiredo : Applying Spectrum-based Fault Localization on Novice's Programs, 2016 IEEE Frontiers in Education Conference, pp.1–8, 2016.

[10] Z. Li, X. Zhou, Y. Wu, Y. Liu, X. Chen : Applying Faulty Statement Category Frequency to Localize Faults for Student Programs, 2021 16th International Conference on Computer Science & Education, pp.969–974, 2021.

[11] A. P. Cavalcanti, A. Barbosa, R. Carvalho, F. Freitas, Y.-S. Tsai, D. Gašević, R. F. Mello : Automatic Feedback in Online Learning Environments: A Systematic Literature Review, Computers and Education: Artificial Intelligence, Vol.2, No.100027, 2021.