プログラミング学習者向け ソースコード提示システムTAMBA

中川 尊雄[1,a)]　藤原 新[1,†1,b)]　畑 秀明[1,c)]　松本 健一[1,d)]

概要：プログラミング学習者が優れたプログラマになるまでの学習過程には数多くの難関が存在する．多くの学習者が利用するオンラインジャッジシステム (OJS) では，様々の課題に対する他利用者の解答を不具合原因の特定や，品質改善などの参考にできる．ただし，解答の実装やレベルは多岐に渡り，学習者の能力に合わない（理解できない／非効率な）解答を参照しても，役に立たない．本研究では，OJS 利用者の解答提出系列を分析することで，学習者の実力に沿った解答ソースコードを提示し，効率的に学習を支援する手法を提案し，TAMBA として実装した．また，手法の有効性検証のため，「誤答ソースコードの不具合原因を特定する」というシナリオに基づく被験者実験を実施した．実験の結果，TAMBA による提示結果は単なる過去の解答一覧と比べて，被験者から有意に「役立つ」と評価された．

TAMBA: Source Code Recommendation System for Programming Learners

1. はじめに

プログラミング学習者にとって乗り越えがたい障壁として，プログラムが予測と異なる振る舞いを見せた際の原因特定や，品質の改善が挙げられる [1]．これらが難関とされる理由の一つに，何が正解形なのかが学習者にとって不明瞭である，ということが考えられる．

こうした時，AOJ[2] や Topcoder[3] に代表されるオンラインジャッジシステム (OJS) では，他の学習者が執筆したプログラムを自由に閲覧し，参考にできる．OJS は，(1) 学習用のプログラミング課題集と，(2) 提出された解答を自動的にコンパイルおよびテストし，即座に実行結果と正誤を通知するジャッジシステムからなる [4]．

多くの OJS では，過去に提出された全ての解答ソースコードが問題別に保存されており，学習者は他利用者の解答とその正誤，実行時情報などを閲覧できる．このため，OJS を利用する学習者は，他利用者の解答を参考に，自身のプログラムの誤りや実装上の問題点（計算量や可読性など）を解決できると考えられる．ただし一般に，ある問題に対して解となるソースコードは，利用するアルゴリズムやライブラリなどにおいて多彩である．加えて，OJS 利用者の学習段階も多岐に渡り，全くの初心者から本職のプログラマまで様々である．そのため，学習者は頻繁に自身の能力に合わない（理解できない／非効率な）解答を参照することになり，学習効率の低下が懸念される．

そこで我々は，効率的な学習の実現をめざし，他の学習者の解答提出履歴を用いて，学習者を徐々に優れた解答へ誘導する学習支援手法を提案する．提案手法の概要は次の通りである：(1) 予め全利用者の過去の解答提出履歴を分析しておき，問題ごとの「解答の進化パターン」を抽出しておく．(2) 学習者が現在のプログラムを入力すると，解答の進化パターンに従って，学習者のプログラムより少し優れた（あるいは，不具合のない）ソースコードを提示する．

我々は，手法の評価および利用者に対する具体的な支援を目指し，プログラミング学習者向けソースコード提示システム TAMBA を開発した．

本論文では以降，提案手法とシステム（TAMBA），システムの有効性評価実験，結果と考察，関連研究の順に説明する．

1 奈良先端科学技術大学院大学
NAIST, Ikoma, Nara 630-0192, Japan
†1 現在，株式会社ワークスアプリケーションズ
a) takao-n@is.naist.jp
b) fujiwara.shin.fe5@is.naist.jp
c) hata@is.naist.jp
d) matumoto@is.naist.jp

©2016 Information Processing Society of Japan

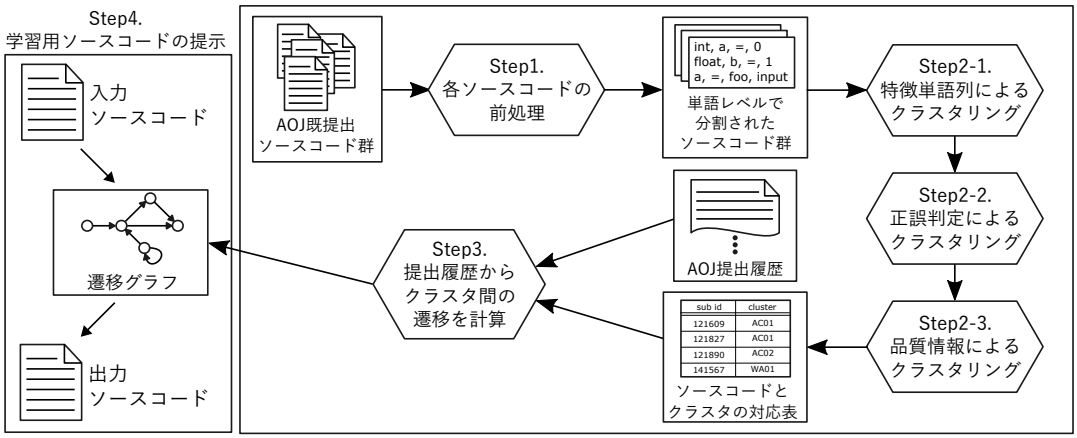

図 1 提案手法の全体像 (右は主にシステムの前処理，左は利用時の動作)

2. 提案手法とシステム (TAMBA)

2.1 概要

多くの OJS において，学習者は課題に対する解答を何度でも提出できる．学習者が複数回にわたって提出を行う動機として，ジャッジシステムによって誤答と判定された場合の不具合修正や，実行時間や可読性など品質の改善が考えられる．このことから本手法では，後に提出された解答ほど，解答の質や学習者の習熟度が向上しているものと仮定する．

我々の提案する手法は次のようなものである．まず，問題ごとの提出ソースコード群を実装方法，正誤，品質によって分類し，提出履歴を用いて分類群間の遷移を表すグラフを生成しておく．先の仮定が正しければ，このグラフの遷移を辿ってゆくことで，誤答は正答に，実装方法，品質は高度なものへと進化すると考えられる．

学習者自身が現在の解答を入力すると，入力と最も類似度の高い分類群の遷移先から，正答であるようなソースコードを提示する．そのため，学習者は自らの学習段階や知識から離れすぎた解答を閲覧することなく，スムースに学習を進めることが可能となる．

我々は提案手法をシステムとして実装し，プログラミング学習者向けソースコード提示システム TAMBA と名付けた．TAMBA が解答提出履歴から遷移グラフを作成し (事前処理)，ソースコードを提示する (利用時の動作) までの概要を図 1 に示す．図中の各ステップにおける処理は次の通りである．

Step1: 解答ソースコードの前処理
Step2: ソースコードの分類
Step3: 遷移グラフの作成
Step4: 遷移グラフに基づくソースコードの提示

2.2 解答ソースコードの前処理

ソースコードを実装方法別に分類する際に用いる N-gram IDF [5] では，単語別に分割されたソースコードが必要となる．TAMBA では，ソースコードを記号と空白部分で分割することで単語列を取り出す．

2.3 ソースコードの分類

TAMBA は，次の手順で解答ソースコードを分類する．

Step2-1: 特徴単語列による実装方法別分類
Step2-2: AOJ の正誤判定による正誤別分類
Step2-3: 品質情報による品質別分類

Step2-1 では，ソースコード群を実装方法別に分類する．ここでは，ソースコードに含まれるすべての単語列から，特徴的な単語列を N-gram IDF を用いて抽出し，その出現頻度の組を特徴量とした階層クラスタリングによる分類を試みる．

N-gram IDF は，Shirakawa らによる単語列への大域的重みづけ手法であり，既存手法 IDF を N 個からなる単語の列 (N-gram) に拡張したものである [5]．N-gram IDF では，単一の単語ではなく，順序を考慮した単語列に対して重みづけが行われるため，識別子名（ライブラリや変数）と構造の特徴を一挙に扱える点が優れている．他の特徴量抽出手法として知られるトピックモデル [6] や PDG [7] は，単体では構造か識別子名の片方しか抽出できない．

階層クラスタリングでは，単語列の出現頻度の組がなす特徴ベクトルを用いて解答間の類似度を算出し，自動的に分類を行う．後の例や実験では，最終クラスタ数を著者らの主観により 8 としている．クラスタ同士の距離は最長距離法によって計算し，特徴ベクトル間の類似度計算にはコサイン類似度を用いた．

Step2-2 では，正答ソースコードと誤答ソースコードの間に明確な品質面の優劣があると考え，Step2 の分類群を正誤情報によって更に二分する．以降，正誤によって分類された各クラスタをそれぞれ正答クラスタ，誤答クラスタと呼ぶ．

Step2-3 では，N-gram と正誤による分類の後，正答ソー

©2016 Information Processing Society of Japan

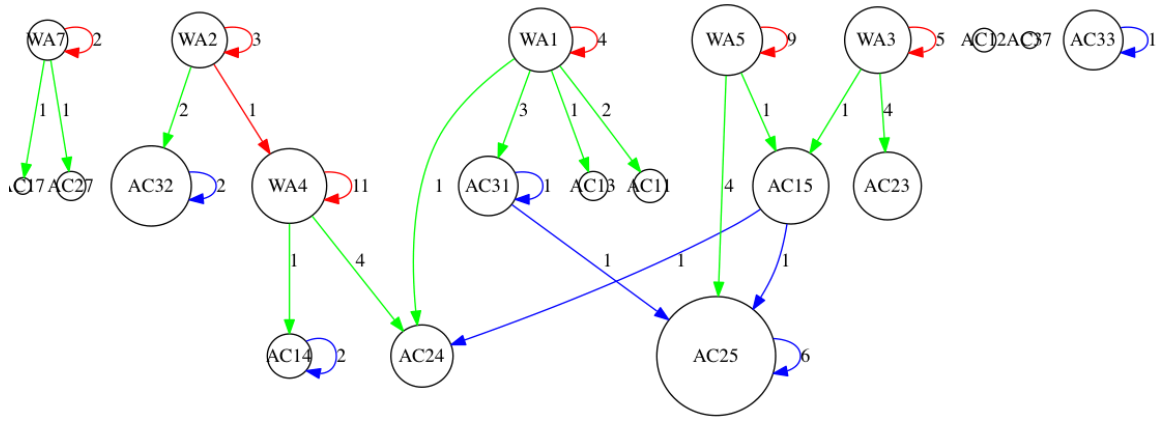

図 2　TAMBA の出力（遷移グラフ）例

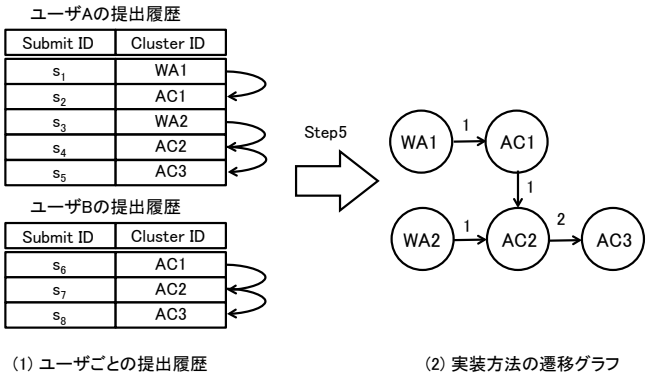

図 3　遷移グラフの作成

スコード群をさらに品質面の違いから階層クラスタリングで分類する．以降の例や実験では，最終クラスタ数を 3 としている．品質の評価尺度は様々あるが，本システムでは，計算機上のリソース消費量という側面から品質を捉え，コードの規模，メモリ使用量，CPU 時間の三つについて，最小値 0，最大値 1 に正規化したものを特徴とした．

2.4　遷移グラフの生成

TAMBA では，OJS の利用者が解答を改善してゆく過程を重み付きの遷移グラフで表現する．グラフのノードは各クラスタを，エッジは改善時の遷移を意味する．遷移グラフは問題ごとに，その問題に対する各利用者の提出履歴（順序情報）を用いて生成される．

遷移グラフの生成方法を図 3 に示す．(1) の表は全利用者 (A,B) の提出履歴，(2) は，生成される遷移グラフを表す．利用者 A はこの問題に対する解答を，s_1 から s_5 の順に，B は，s_6 から s_8 の順に提出しており，各提出には Step2 の分類 ID が付与されている．クラスタ ID 内の WA (Wrong Answer) は誤答クラスタを， AC (ACcepted) は正答クラスタを表す．

TAMBA は，ある問題に対して同一利用者が行った，連続する 2 つの提出について，古い提出のクラスタから新しい提出のクラスタへ遷移するエッジを形成する．たとえば，利用者 A でははじめ，s_1 から s_2，の順で提出しているので，WA1 から AC1 の間にエッジが形成される．

エッジの重みは，同一の遷移をした利用者数である（単一の利用者の重複する遷移は対象としない）．たとえば，s_4 から s_5 ならびに $s7$ から s_8 のように，両利用者が同じ遷移をした場合，エッジの重みは 2 となる．ただし TAMBA では，s_2，s_3 間のような正答ソースコードから誤答ソースコードへの遷移は明らかな間違いとみなし，遷移グラフに反映しない．

AOJ の提出履歴から生成した遷移グラフの例を図 2 に示す．

2.5　遷移グラフに基づくソースコードの提示

Step 5 までの遷移グラフ生成処理は，TAMBA の事前処理であった．Step 6 は，学習者（利用者）の利用時における動作であり，利用者の現在のソースコードと予め生成した遷移グラフから，参考にすべき解答ソースコードを提示する．

TAMBA は，利用者から入力を受け付けると，利用者の入力したソースコードと各クラスタとの類似度を計算する．そして，最も類似度が高いクラスタに利用者のソースコードが含まれると仮定し，その遷移先に存在するソースコード群を適切な例として提示する．

図 2 を例に挙げると，入力が WA5 のノードに含まれる場合，TAMBA の提示するソースコードは AC15 ならびに AC25 に含まれる解答ソースコードとなる．同様に，入力が AC15 に含まれるならば， AC24 ならびに AC25 のソースコードが提示される．

この方式では，提示対象ソースコードが時に混乱を招く量になる可能性があるため，TAMBA では，提示候補からもっとも類似度が高い 5 件のソースコードを提示する．

©2016 Information Processing Society of Japan

表 1 提出数の多い 10 件の問題

ID	問題名	概要	正答提出数	誤答提出数	全提出数
1	List of Top 3 Hills	入力された 10 個の数を大きい順に 3 つ出力	327	139	466
2	Digit Number	入力された 2 つの整数値の和の桁数を求める	238	106	344
3	GCD and LCM	入力された整数値の最大公約数と最小公倍数を出力	172	110	282
4	Reverse Sequence	入力された文字列を反転して出力	191	8	199
5	Parallelism	入力された 2 直線が平行かの判定	67	238	305
6	Is it a Right Triangle?	特定の 3 辺をもつ三角形が直角三角形か判定	244	102	346
7	Hello world	Hello World を出力	403	55	458
8	Simultaneous Equation	与えられた連立方程式の解を求める	161	358	519
9	QQ	九九の一覧を計算して結果を出力	466	135	601
10	Prime Number	ある整数値とその 2 倍値の間の素数の数を出力	185	127	312

3. 実験

被験者実験によって，TAMBA の出力と，従来から存在する AOJ の解答一覧ページを比較し，提案手法の有効性を検証する．実験参加者は奈良先端科学技術大学院大学の大学院生 8 名，いずれも 23 歳から 26 歳である．アンケート調査の結果，実験参加者のプログラミング経験は 3 年から 12 年であり，うち 3 名はなんらかの OJS を利用した経験があると答えた．以下，対象データに対する事前分析，実験設計の順に説明を行う．

3.1 対象データと事前分析

実験では，実際の OJS である AOJ 上の問題データ，ならびに提出履歴を用いる．そこで，本節ではまず対象データの構成について簡単に紹介する．その後，更に事前分析として，本研究の前提（一人の学習者が複数回の提出を行う，解答には様々な実装方法が存在する）が成立することを確かめる．

3.1.1 対象データ

AOJ は，各種のプログラミングコンテスト問題集や，独自問題集を公開しており，多くの利用者に利用されている [4]．AOJ の問題には，プログラムの仕様，入出力の形式，入力の制約，サンプル入力，サンプル出力が記述されており，学習者は仕様に沿うようプログラムを作成し，提出することを求められる．

本研究で用いる AOJ のデータは，全問題の提出履歴と解答ソースコードを含むものである．提出履歴データは各行がひとつの提出と対応しており，その属性は，提出 ID，提出者 ID，問題 ID，正誤判定結果，メモリ利用量，CPU 時間，使用言語，コードサイズからなる．

AOJ において，解答ソースコードは様々な言語で記述できるが，本研究では利用者の多さ，ならびに分析の容易さの点から Java のみを実験および事前分析の対象とする．

3.1.2 事前分析

対象データの特徴について調査するため，特に提出数の多い 10 件の問題に対する 500 件の解答を著者のうち一人がマニュアルリーディングを行う．はじめに，提出数の多い 10 件の問題の概要，解答の全提出数，正答数，誤答数を表 1 に示す．

マニュアルリーディングでは，表に挙げた 10 個の問題に対する全解答から，各問題 50 件の解答を無作為抽出し，その実装方式を著者の主観によって分類した．結果，10 件中，5 件の問題 (ID: 1, 2, 3, 4, 10) では提出によって異なる複数の実装方法が採用され，残りの 5 件 (ID: 5, 6, 7, 8, 9) では単一の実装方法が採用されていることがわかった．以降，ID1,2,3 の問題について，実装方法の種類と内訳を例として挙げる．

List of Top 3 Hills (ID: 1) では，標準ライブラリの sort によるもの (19 件)，入力を順に参照し大きな値を記憶するもの (17 件)，バブルソート (9 件)，クイックソート (5 件) が見られた．Digit Number (ID: 2) では，2 値の和の log10 の計算 (10 件)，2 値の和を 10 で割った回数の数え上げ (16 件)，2 値の和を文字列に変換し文字数を数える (24 件) が見られた．GCD and LCM (ID: 3) では，ユークリッドの互除法（再帰）(22 件)，ユークリッドの互除法（ループ）(27 件)，標準ライブラリの gcd によるもの (1 件) が見られた．

一方，単一の実装方法のみが採用されている問題では，例えば Hello World (ID:7) のように初歩的な実装でも計算量上の問題が起きないと思われるものが多かった．

上記の結果から，問題に対する解答実装方法は複数ありうること，より高度な問題ほど実装の違いが見られることがわかった．実装方法によっては，処理効率（例: List of Top 3 Hills におけるバブルソートとクイックソートの違い）が異なったり，学習者に特定のドメイン知識を要求する (例: Digit Number において桁数は log10 で求められる) ことがわかった．

また，各問ごとの 1 人当たり平均提出数を調査したところ，1 人当たり 1.2 件から 3.9 件程度提出していることがわかった．上位十件全体での平均は 1.8 件となり，複数回の解答提出を行う利用者の存在が確かめられた．

以上より，本研究の前提である「学習者が複数回解答を

©2016 Information Processing Society of Japan

提出する」ならびに「解答には様々な実装方法が存在する」が確かめられた．

3.2 実験設計

実験では，提示された AOJ のプログラミング問題と，問題に対応する誤答ソースコードを読み，誤答の原因を特定する実験課題を繰り返し実施してもらう．各課題において参加者は，(a) 誤答を入力とした TAMBA の出力，あるいは (b) 提出された解答の一覧，のどちらかを補助資料として提示される．実験全体では，3 問の問題について各 2 件の誤答ソースコードが提示され，実験参加者は合計で 6 件の実験課題を課される．

各課題には 3 分の制限時間があり，制限時間内に原因特定ができた場合は，その原因を実験監督者に口頭で説明してもらう．説明された誤答内容が正しかった場合，次の課題へ進み，誤っていた場合，引き続き同じ問題に取り組んでもらう．

各課題が終わるたび，被験者は（補助資料が）“Q1:誤答原因の特定に役立ちましたか？”，“Q2:プログラミングの学習に役立つと思いますか？”，という質問項目について，“役立つ” から “役立たない” まで 5 段階のリッカート尺度を用いたアンケートに答える．更に，各課題終了後のアンケートに加えて，全課題終了後にもプログラミング経験や年齢を解答してもらうアンケートを実施する．このアンケートには自由記述欄を設け，実験についての感想や，TAMBA を利用した感想について記述してもらう．

順序効果や学習効果を防ぐため，問題の提示順と，(a), (b) どちらの補助資料が先に示されるかを被験者ごとにランダム化し，カウンターバランスを取る．実験課題に使用した 3 問の問題は，著者の主観によって，実験参加者のレベルに合うと考えたもの (ID1,2,10) を選んだ．また，各問 2 件の誤答ソースコードについては，2 つの誤答原因や実装方法が異なるように選んだ．

実験の手順を次に示す．2-1 から 2-4 まは実験課題であり，実際には 6 回反復される．

Step 1: 実験内容の説明
Step 2-1: 実験課題：問題文の内容理解
Step 2-2: 実験課題：誤答ソースコード 1 の原因特定
Step 2-3: 実験課題：誤答ソースコード 2 の原因特定
Step 2-4: 実験課題：アンケート記入
Step 3: 全体についてのアンケート記入

なお，時系列的な矛盾を避けるため，課題となった誤答ソースコードの執筆者のソースコードは，補助資料から削除した．

4. 結果と考察

4.1 TAMBA の提示結果

はじめに，TAMBA が実際に誤答ソースコードに対して，誤答原因に役立ちそうな正答ソースコードを提示できたかどうか，例を用いて示す．前章の表 1 より，問題 Digit Number に対する誤答ソースコード二件と，それを入力した場合の TAMBA の出力（5 件のうち 1 件）を図 4 から図 7 に示す．各図では，省スペース化のため Import 文，Main クラスの宣言部ならびに解答に影響しない例外処理部を意図的に省略している．

この問題は入力された 2 値の合計値の桁数を求めるものである．問題の入力は複数行からなり，学習者は全ての行の入力について桁数を求め，出力する必要がある．

誤答ソースコード 1 (図 4) では，2 値の合計の桁数を出力する必要があるにも関わらず，2 値の桁数の合計を出力している．例えば，入力が “1000 999” であるとき，誤った結果 “7” を出力する．TAMBA の提示した解答 6 では 2 値の合計に対して log10 を計算しており，同入力に対する正しい結果 “4” を出力する．このペアは，桁数の計算に

```
public static void main(String args){
    BufferedReader br = new BufferedReader(
        new InputStreamReader(System.in));
    String line = null;
    try{
        while((line = br.readLine() ) != null){
            String sl = line.split(" ");
            int a = (int) Math.log10(
                      Integer.parseInt(sl0));
            int b = (int) Math.log10(
                      Integer.parseInt(sl1));
            System.out.println(a+b);
        }
    ～例外処理部分省略～
}
```

図 4 入力した誤答ソースコード 1(下線部が誤り)

```
public static void main(String args){

    ~~~~~~~
    Scanner sc = new Scanner(System.in);
    int a = sc.nextInt();
    int b = sc.nextInt();
    int c = a + b;
    int k = 0;
    while(c!=0){
        c /= 10;
        k++;
    }
    System.out.println(k);

    ~~~~~~~
}
```

図 5 入力した誤答ソースコード 2(波線部が不足)

©2016 Information Processing Society of Japan

log10 関数を用いている点，入力に BufferedReader クラスを用いる点が類似している．

一方，誤答ソースコード 2 (図 5) では，全入力に対する結果ではなく，1 行目の結果しか出力しない．TAMBA が提示した解答 7 は，全体の処理を while ループで囲うことにより，複数行の入力に対応している．このペアは，10 で割り進める解法である点，入力に Scanner クラスを用いる点が類似している．

以上より，TAMBA が学習者の入力と類似する正答ソースコードを提示することが，定性的にではあるが示された．また，二例のどちらでも，TAMBA は識別子と構造の両観点から類似した解答を抽出できている．これは，先に述べた N-gram IDF の利点である，識別子と構造を双方考慮できる，という部分がよく働いた結果であると考えられる．

```
public static void main (String args) {
    try {
        BufferedReader br = new BufferedReader(
            new InputStreamReader(System.in)
        );
        String line = "";
        while ((line = br.readLine()) != null){
            String str = line.split(" ");
            int a = Integer.parseInt(str0);
            int b = Integer.parseInt(str1);
            double sum = (double)(a+b);
            double digit = Math.floor(Math.abs(
                                Math.log10(sum)))+1;
            System.out.println((int)digit);
        }
    ～例外処理部分省略～
}
```

図 6 誤答 1 に対する出力 (下線部が関係する修正)

```
public static void main(String args) {
    Scanner sc=new Scanner(System.in);
    while(sc.hasNext()){
        int a=sc.nextInt();
        int b=sc.nextInt();
        int c=a+b;
        int count=0;
        while(0<c){
            c/=10;
            count++;
        }
        System.out.println(count);
    }
}
```

図 7 誤答 2 に対する出力 (下線部が関係する修正)

4.2 被験者実験の結果

実験参加者に対するアンケート結果を図 8，図 9 に示す．図はアンケート結果に占める各評価値の割合を示しており，1 に近いほど “役立つ”，5 に近いほど “役立たない” ことを示している．

上の帯グラフは，TAMBA で出力した解答ソースコードを提示した場合に得られたアンケート結果を示しており，下の帯グラフは，AOJ の提出一覧を提示した場合に得られたアンケート結果を示している．

図 8 は，Q1 に対する結果である．TAMBA は役立つを示す 1 と 2 が 40%程度を占めているのに対し，AOJ での提出一覧では，20%程度であった．アンケート結果に対するウィルコクソンの順位和検定の結果，$p < 0.01$ で TAMBA と AOJ の提出一覧に有意な差が見られた．

図 9 は，Q2 に対する結果である．TAMBA は役立つを示す 1 と 2 が 45%程度を占めているのに対し，AOJ での提出一覧では，25%程度であった．アンケート結果に対するウィルコクソンの順位和検定の結果，TAMBA と AOJ の提出一覧に有意な差は確認できなかった．

また，実験中に被験者が正しい誤答原因を特定できた件数は，TAMBA での全検証 24 件中 12 件，AOJ の提出一覧での全検証 24 件中 7 件であった．フィッシャーの正確確率検定を実施した結果，TAMBA と AOJ の間に有意な差は確認できなかった．

4.3 考察

被験者の評価からは，TAMBA は AOJ の提出一覧よりも，誤答原因の特定に役立つことがわかった．学習に役立つかどうかについては，評価は高かったが有意差は見られず，必ずしも学習に役立つとはいえないことがわかった．同様に，誤答原因の特定に成功した件数においても，TAMBA 利用時の方が高いものの，有意差は見られなかった．

アンケートの自由記述欄においては，次のように TAMBA の有効性を裏付けるコメントが見られた．「誤答ソースコードと TAMBA によって提示された解答ソースコードは非

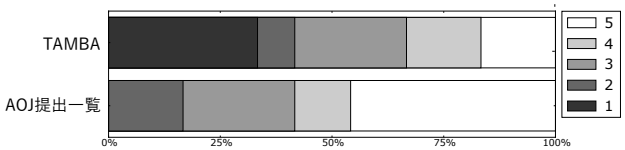

図 8 Q1“誤答原因の特定に役立ちましたか？” に対する結果 (1:役立つ，5:役立たない)

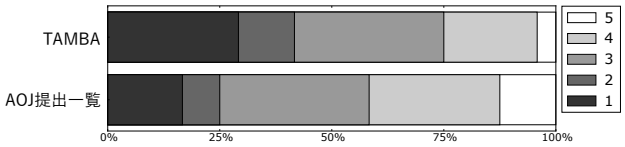

図 9 Q2“プログラミングの学習に役立つと思いますか？” に対する結果 (1:役立つ，5:役立たない)

©2016 Information Processing Society of Japan

常に類似しており，誤答原因の特定に役立った.」,「誤答ソースコードで用いているアルゴリズムの正答ソースコードが出てきたため特定しやすかった.」.

各コメントは，TAMBA の提示する解答ソースコードは利用者から見ても誤答ソースコードと類似しており，原因特定のため有効であることを示唆している.

一方，TAMBA の改善点として，次のコメントが見られた.「TAMBA によって 5 件のソースコードが提示されたが，上位の解答ソースコードしか参考にしなかった.」,「誤答ソースコードとの相違点を提示することで，より原因の特定に役立つと思った.」.

各コメントは，解答ソースコードの提示方法を改善することで，支援が更に効率化される可能性を示唆している．例として，TAMBA への入力と提示ソースコードの類似度を表示することで，学習者により多くの情報を与えることができる.

5. 妥当性への脅威

本研究の外的妥当性の脅威として，AOJ 上での提出数が多い 10 件の問題を実験に用いたことが挙げられる．そのため，今回対象としなかった OJS や，提出数の少ない問題での有効性については断言できない．ただし，AOJ では様々なプログラミングコンテストの過去問を集めて公開しており，問題の一般性は比較的高い．また，提出数の多さは，すなわち学習者からの需要が高いという意味でもあり，一定の妥当性は確保されたと考えている.

内的妥当性への脅威として，実験設計の問題点が挙げられる．本研究では，学習者が誤答した場合の原因特定を想定して実験を行ったが，実験参加者が同様の誤答で行き詰るとは限らない．アンケートでは実験参加者が誤答ソースコードのみで原因を特定できた例もあり，意図した条件にならなかったケースが確認されている．ただし，主観評価では「誤答原因の特定に役立った」という結果が有意に示されているため，問題は少ないと考えられる．今後の評価においては，実験者が制限時間内に誤答原因を特定できなかった場合に限定した分析など，より現実的な環境を構築する必要性が認められる.

また，構成概念妥当性への脅威として，ソースコード分類に用いた特徴量の設計の問題点が挙げられる．たとえば，品質情報として CPU 時間やメモリ量，コード規模を挙げているが，これらはコンピュータのリソース消費量の観点から品質を見たもので，一側面的である．今後は，人間にとっての理解しやすさなど，他の観点から見た品質についても考慮する必要があると考えられる.

6. 関連研究

我々はこれまでに，プログラミング学習者の支援を目的として，OJS の解答ソースコードを解法により分類した [8]．本稿では，分類結果と提出履歴を用いて参考ソースコードを提示する手法を提案・実装している点で，過去の研究より進展している.

これまで，ソースコードの検索技術が広く研究開発されてきた．Inoue らは部品の重要度を表す Component Rank を用いたソースコード検索ツール SPARS-J を開発した [9]．Sadowski らは Google における検索ツールの使途を調査した [10]．Bajracharya らはキーワード，構造を索引としたオープンソースコード検索ツール Sourcerer を開発した [11].

また同様に，ソースコードの推薦技術も提案されてきた [12]．山本らは実装中のソースコードから類似するソースコードをシームレスに推薦するシステムを Eclipse に実装した [13]．柿元らは協調フィルタリングにより，ソースコード再利用のために実装中のクラスと類似するクラスを推薦するシステムを提案した [14].

ここまでに挙げた研究はいずれも実際の開発現場におけるソースコードの再利用を目的としたものである．本研究の目的はプログラミング学習の支援であり，小規模プログラムについて，実装方法の違いなども考慮したソースコード推薦を実現している.

Zimmerman らはプログラミング初学者のために，部分的なソースコードから所望のソースコードを推薦するフレームワークを提案した [15]．Zimmerman らの目的は，学習者の支援であり，本研究と目的が類似している．本研究の手法は，OJS を利用する他の学習者の経歴に合わせ，やや優れたソースコードを提示するものであり，手法および想定するシナリオが異なる.

ソースコードの再利用や知識の共有を目的に，ソフトウェアを分類する研究がされている．Kawaguchi らはソースコード中の識別子に着目し，LSA を用いて分類するツール MUDAblue を開発した [16]．Tian らはプログラム中の言語情報から LDA を用いてドメイン別の分類を実現した [6]．これらの研究は言語情報からソースコードを分類する点で本研究と類似しているが，目的の点で大きく異なる.

ソフトウェアの保守性を高めることを目的として，同一あるいは類似しているコード片であるコードクローンに関する研究も多くされている．Kamiya らは高速にコードクローンを検出するツール，CCFinder を開発した [17]．Roy らはコードクローンをタイプごとに分類し，比較実験を行った [18]．肥後らはコードクローンの種類と検出技術の研究成果を体系的にまとめた [19]．本研究は，同一あるいは類似するだけでなく，より優れたソースコードを提示する点でコードクローンの研究とは異なる.

7. おわりに

本研究では，オンラインジャッジシステムを利用するプログラミング学習者の支援を目指し，他の学習者の提出履

©2016 Information Processing Society of Japan

歴から抽出した解法の進化パターンに沿って参考コードを提示する手法を提案・実装し，実験によりその有効性を検証した．

我々は手法の有効性検証および将来的な実用化のために，プログラミング学習者向けソースコード提示システム TAMBA を実装した．TAMBA は，実際のオンラインジャッジシステムである AOJ (Aizu Online Judge) の問題を対象に，提出履歴から遷移グラフを生成し，その情報をもとに利用者にソースコードを推薦する．

被験者実験の結果，TAMBA の出力は AOJ の提出一覧に比べ，有意に「誤答原因の特定に役立つ」と評価された．著者らによる出力の検証でも，誤答ソースコードと類似し，かつ，誤答原因が適切に修正されたソースコードを候補として提示できることが確認された．

本研究の今後の展望として，実際のオンラインジャッジシステム利用者を対象とした大規模な実験や，環境を整えてのシステムの公開などが挙げられる．また，今回は正答ソースコードの品質向上を実現するという点については検証実験を行わなかったため，その実施も今後の目標となる．より発展的には，学習履歴データと利用者の入力をもとに学習段階を測定したり，学習者へフィードバックを行うような応用も興味深い．

謝辞　AOJ のデータをご提供いただいた会津大学の渡部有隆上級准教授に感謝します．本研究は JSPS 科研費 16H05857 と頭脳循環を加速する戦略的国際研究ネットワーク推進プログラム：ソフトウェアエコシステムの理論構築と実践を加速する分野横断国際ネットワークの構築（G2603）の助成を受けた．

参考文献

[1] Ko, A. J., Myers, B. A. and Aung, H. H.: Six Learning Barriers in End-User Programming Systems, *Proceedings of the 2004 IEEE Symposium on Visual Languages - Human Centric Computing*, VLHCC '04, Washington, DC, USA, IEEE Computer Society, pp. 199–206 (online), DOI: 10.1109/VLHCC.2004.47 (2004).

[2] ICPC Programming Club: AIZU ONLINE JUDGE: Programming Challenge, University of Aizu (online), available from ⟨http://judge.u-aizu.ac.jp/onlinejudge/⟩ (accessed 2015-11-12).

[3] Topcoder: Topcoder, Appirio (online), available from ⟨https://www.topcoder.com/⟩ (accessed 2016-1-24).

[4] 有隆渡部：オンラインジャッジの開発と運用 -Aizu Online Judge-，情報処理，Vol. 56, No. 10, pp. 998–1005 (2015).

[5] Shirakawa, M., Hara, T. and Nishio, S.: N-gram IDF: A Global Term Weighting Scheme Based on Information Distance, *Proceedings of the 24th International Conference on World Wide Web*, International World Wide Web Conferences Steering Committee, pp. 960–970 (2015).

[6] Tian, K., Revelle, M. and Poshyvanyk, D.: Using Latent Dirichlet Allocation for Automatic Categorization of Software, *Proceedings of the 2009 6th IEEE International Working Conference on Mining Software Repositories (MSR '09)*, Washington, DC, USA, IEEE Computer Society, pp. 163–166 (online), DOI: 10.1109/MSR.2009.5069496 (2009).

[7] Ferrante, J., Ottenstein, K. J. and Warren, J. D.: The Program Dependence Graph and Its Use in Optimization, *ACM Transactions on Programming Language and Systemns (TOPLAS)*, Vol. 9, No. 3, pp. 319–349 (online), DOI: 10.1145/24039.24041 (1987).

[8] 藤原新，　畑秀明，門田暁人，松本健一：プログラミング初学者のためのソースコード再利用支援，研究報告ソフトウェア工学（SE），Vol. 2015-SE-190, No. 16, pp. 1–7 (2015).

[9] Inoue, K., Yokomori, R., Yamamoto, T., Matsushita, M. and Kusumoto, S.: Ranking Significance of Software Components Based on Use Relations, *IEEE Transactions on Software Engineering*, Vol. 31, No. 3, pp. 213–225 (online), DOI: 10.1109/TSE.2005.38 (2005).

[10] Sadowski, C., Stolee, K. T. and Elbaum, S.: How Developers Search for Code: A Case Study, *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2015)*, New York, NY, USA, ACM, pp. 191–201 (online), DOI: 10.1145/2786805.2786855 (2015).

[11] Bajracharya, S., Ossher, J. and Lopes, C.: Sourcerer: An Infrastructure for Large-scale Collection and Analysis of Open-source Code, *Science of Computer Programming*, Vol. 79, pp. 241–259 (online), DOI: 10.1016/j.scico.2012.04.008 (2014).

[12] Robillard, M. P., Maalej, W., Walker, R. J. and Zimmermann, T.: *Recommendation systems in software engineering*, Springer (2014).

[13] 山本哲男，吉田則裕，肥後芳樹：ソースコードコーパスを利用したシームレスなソースコード再利用手法，情報処理学会論文誌，Vol. 53, No. 2, pp. 644–652 (2012).

[14] 柿元健，角田雅照，大杉直樹，門田暁人，松本健一：協調フィルタリングに基づく Java クラスファイル推薦システム，電子情報通信学会技術研究報告. DC, ディペンダブルコンピューティング，Vol. 104, No. 346, pp. 29–34 (2004).

[15] Zimmerman, K. and Rupakheti, C. R.: An Automated Framework for Recommending Program Elements to Novices, *Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering*, ASE 2015, ACM, pp. 283–288 (2015).

[16] Kawaguchi, S., Garg, P. K., Matsushita, M. and Inoue, K.: Mudablue: An automatic categorization system for open source repositories, *Proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC '04)*, IEEE Computer Society, pp. 184–193 (2004).

[17] Kamiya, T., Kusumoto, S. and Inoue, K.: CCFinder: A Multilinguistic Token-based Code Clone Detection System for Large Scale Source Code, *IEEE Transactions on Software Engineering*, Vol. 28, No. 7, pp. 654–670 (online), DOI: 10.1109/TSE.2002.1019480 (2002).

[18] Roy, C. K., Cordy, J. R. and Koschke, R.: Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach, *Science of Computer Programming*, Vol. 74, No. 7, pp. 470–495 (online), DOI: 10.1016/j.scico.2009.02.007 (2009).

[19] 肥後芳樹，楠本真二，井上克郎：コードクローン検出とその関連技術，電子情報通信学会論文誌 D，Vol. 91, No. 6, pp. 1465–1481 (2008).

©2016 Information Processing Society of Japan