

プログラミング演習中の学習者の行動分析に基づく 課題依存型行き詰まり検出器の試作

A prototype of lesson-dependent impasse detector
based on learners' behavior analysis on programming class

小暮 悟^{*1}
Satoru Kogure

木下 恭輔^{*1}
Kyosuke Kinoshita

山下 浩一^{*2}
Koichi Yamashita

野口 靖浩^{*1}
Yasuhiro Noguchi

小西 達裕^{*1}
Tatsuhiko Konishi

伊東 幸宏^{*3}
Yukihiro Itoh

^{*1}静岡大学 情報学部
Faculty of Informatics, Shizuoka University^{*1}

^{*2}常葉大学 経営学部
Faculty of Business Administration, Tokoha University^{*2}

^{*3}静岡大学
Shizuoka University^{*3}

In a typical programming exercise, learners create their programming code for exercises. Teachers should evaluate their programming code in detail and give feedbacks to the learners. However, it is difficult for teachers to evaluate and give a feedback to a learners who are in impasse in real time. We had also developed a system that detects learners' lesson-independent impasse by automatically collecting learners' behavior during programming exercise. In this paper, we propose three method to detect lesson-dependent impasse based on the previous system. We developed a prototype system that implements a part of the proposed methods.

1. はじめに

従前より行われている一般的なプログラミング演習では、まず講義でプログラミングに関する一般知識や、学習対象のアルゴリズムの提示が行われる。そして演習では、実際の課題を解くために、サンプルコードなどを参考にしながらコーディングしていく。これらのプログラミング演習を支援する目的で、様々なプログラミング学習支援システムが開発されている。例えば、体験型のアルゴリズム理解支援環境 [野口 10][大山 11][伊藤 15] や、プログラムの挙動を視覚化する支援システム [Moreno 04][今泉 11][Yamashita 16] が挙げられる。

プログラミング演習を支援するシステムとして、演習中に教員や TA をサポートするシステムも存在する。我々も、演習中の進捗をリアルタイムで判定し教員がその結果を利用して全体・個別指導に役立てられる演習支援システム [Kogure 13]、演習中の指導履歴を再利用可能な個別指導支援システム [Noguchi 16] を開発してきた。さらに、演習中の学生の行き詰まりを検出できる学習支援システム [Yamashita 17] を開発してきた。この支援システム（以降、先行支援システム）は、「コンパイルを無意味に繰り返す」「エディタでのプログラム編集が止まっている」といった、演習全般の一般的行き詰まりの検出を目指したシステムである。情報系の大学で開講されているプログラミング演習（100 名規模）の授業で実際にプログラミング演習の学習者の演習に関する行動を収集した。

これらの先行研究を通じて、実際の現場での先行支援システムの活用方法について、プログラム演習を担当している複数の教員への聞き取り調査を行ったところ、汎用的行き詰まりに加えて、課題に依存した行き詰まり（例えば、設定した課題に対する典型的誤り）も検出できるとよりよい学生指導が行えるのではという調査結果を得た。そこで、本研究では、先行支援システムをベースに、課題依存型の行き詰まり検出手法について提案する。実際の授業で収集したプログラム編集、コンパ

イル、実行ログの一部をサンプルデータとして、試作した課題依存型行き詰まり検出器による検出性能を調べたところ、精度 0.89、再現率 0.39 を得た。

2. 先行・関連システム

2.1 学習者プログラムの静的解析による正誤判定システム

鈴木らは、教師が用意した正解アルゴリズムと学習者が提出したプログラムをマッチングすることで、学習者のプログラムのどこが合っていてどこが間違っているかを判定できる静的解析器 [鈴木 07] を開発している。対象プログラミング言語は C 言語である。教師は、Problem Analysis Diagram (PAD) で記載された正解アルゴリズムを用意する。この正解アルゴリズムは、予め用意した正解プログラムから PAD を生成後、PAD エディタを用いて微調整して用意する。提出された学習者のプログラムも PAD に自動変換することで、PAD 同士の構造を考慮した再帰的マッチングを行うことができる。プログラミング演習の状況を勘案し、別解への対応や、記述順序や記述内容の多様性を考慮するため、記述順序が任意でもマッチング可能な任意順序構造と、記述されているもののいずれかが書いてあればマッチング可能な任意手段構造を取り扱える拡張 PAD を定義した。正解アルゴリズムは拡張 PAD で表現できるようにシステムが構築されている。システムは比較結果として、正解アルゴリズムのステートメント数をベースとした一致率 A と、学習者プログラムのステートメント数をベースとした一致率 B の 2 つを計算する。

本システムは主に、プログラミング演習の提出物の採点にかかる教員のコストを削減する目的で作成された。このシステムを用いると、一致率 A および一致率 B がともに高いプログラムのレポートは必須課題をきっちり終わらせておりプログラムをじっくり見る必要がないと教員が判断できる。また、一致率 A および一致率 B がある程度高い学習者については、専用のビューアを用いて「誤っていると判定された部分」のみプログラムを見直すことができる。また、一致率 A が 100%で、一致率 B が 100%でない場合は、学習者が追加課題を実施している可能性を検出できるため、「学習者のプログラムのみに

連絡先: 小暮 悟, 静岡大学情報学部
〒432-8011 静岡県浜松市中区城北 3-5-1
E-mail: kogure@inf.shizuoka.ac.jp

含まれる記述」だけをチェックすることが可能である。このように教員が全学生のすべてのプログラムを詳細に確認する必要がなくなるため、採点の時間的コストを削減できる。

本研究においては、この静的解析器を、提出されたプログラムが典型的誤りを含むプログラムかどうかを判定するために利用する。詳細は、3.1 節で述べる。

2.2 学習者プログラムの動的解析による正誤判定システム

王らの学生プログラム動的解析器 [王 13] は、2.1 節で示した静的解析のあとに実行される。動的解析は、静的解析では扱えない正しいプログラムを正しいと判定するために、プログラムの構造ではなく、学習者プログラムの実行履歴を観測して、正解プログラムの実行履歴と比較する。教師側は、複数のテストケースを事前に用意する。動的解析器は、gdb のデバッグ用コマンドを自動生成し、そのコマンドを gdb で実行することで、観測ポイントでの各変数の値や、繰り返しの回数等を取得する。その値と、教師が予め設定しておく正解と比較することで、学生のプログラムが仕様を満たしているかどうか判定する。

本研究では、行き詰まり検出に動的解析がどのように利用できるかの検討結果を示す。

2.3 先行支援システム

先行支援システムの行き詰まり判定 [Yamashita 17] では、学習者の行き詰まり (以降, impasse) を以下の 4 タイプに分類している。

- Type 1: どう実装したらいいかわからない
- Type 2: コンパイルエラーを解決できない
- Type 3: 実行時エラーを解決できない
- Type 4: 論理エラーを解決できない

impasse がどのような状況によって検出されうるかを調査するため、のべ 15 人の情報学系大学院生に対し、様々な難易度の課題を出し、その際のコーディング状況を計 30 時間収集している [Yamashita 17]。収集された情報は以下のとおりである。以降、以下の 4 つの情報をコーディング情報と呼ぶ。

- ソースコード (一定時間毎)
- ソースコード (コンパイル毎)
- コンパイル結果・エラー (コンパイル毎)
- プログラム実行・エラーログ (実行毎)

これらのコーディング情報を分析し、上記 4 つのどの impasse に陥っていることをどのように判断できるか検討がされている。その検討により、各 impasse の検出に用いることが可能な特徴的振る舞い (以降, 迷い兆候) が検討された。以下に 10 個の迷い兆候と実際にシステムで利用された閾値を示す。

- Type 1-1: コードが 15 分以上編集されていない
- Type 2-1: あるコンパイルエラーが 4 回以上のコンパイルで解決できていない
- Type 2-2: 一度解決したコンパイルエラーが再び観測された
- Type 3-1: 幾つかの実行時エラーが 7 回以上観測された

- Type 4-1: コードの同一箇所を 4 回以上連続で修正している
- Type 4-2: printf() のような観測文を挿入して実行後、削除している
- Type 4-3: コンパイルを 2 分間の間に 5 回以上している
- Type 4-4: ソースコードの編集が以前の編集内容に戻っている
- Type 4-5: 一度の編集でソースコードの 25% 以上が編集された
- Type 4-6: ある変数を含む文が、5 箇所以上連続で編集された

これらの迷い兆候を検出できる検出器を実装し、その結果を時系列で確認可能なビューアが構築されている。ビューアは以下の 3 通りの利用方法が想定されている。

1. 学生が自身の学習を振り返る
自身の時系列 impasse 検出結果のヒートグラフを俯瞰し、自分がつまずいた場所でどのようなソースコードで、それがどう修正されて正しい動作になったかを見ることで、自身の不得意な点を振り返ることができる。
2. 教員・TA が学習者全体の impasse の状況を把握する
検出された学生の impasse について、ソースコードの変異からその impasse に該当するソースコードの部分特定することができる。更に、正解プログラムについて、プログラムの行番号と、その単元での学習目標の対応を予め指定しておくことで、クラス全体の学習目標に関する impasse の重要度 (impasse の解決にかかった時間など) を測定することができる。このデータは、次週の授業内容や翌年度の授業内容の改善に利用できる。
3. 各個人の impasse の詳細な状況を確認する
演習中に全体の状況を把握した後、ある impasse に陥っている学生の一覧が出力され、学生の ID をクリックするとその学生の時系列 impasse 検出結果を詳細に閲覧できる。これらは、全体説明が必要かどうか、あるいは、個人対応が必要かどうかの判断材料に使える。

本研究ではこの先行支援システムで行った、課題非依存の一般的な迷い兆候からは直接推定不可能な、課題依存型の impasse を検出するための枠組みを提案する。

2.4 関連研究

井垣らの C3PV [井垣 13] は、学生がオンラインエディタで作業をする。システムはコーディング過程、エディタ領域ログ、コントローラ領域ログを記録する。記録したログから「総行数」「課題毎のコーディング時間」「単位時間あたりのエディタ操作数」「課題毎のエラー継続時間」の 4 つの指標を計測する。計測した指標を可視化し教師に提示することで机間巡視を支援する。C3PV は本研究と同様、学習者のログを収集し机間巡視を支援するシステムであるが、課題特有の impasse に着目したものではない。

松澤らは、ブロック言語と Java 言語をシームレスに変更可能なプログラミング環境 [松澤 14] を提案している。このシステムを用いて学習者のプログラム履歴を収集し、ブロックエディタの使用率を調べ、学習者自身のプログラミングに対する苦手意識とブロックエディタの使用率の間に相関が見られたことを報告している。演習中の行動を自動収集し、学生の状況把

握に用いる点は同じであるが、課題依存型の *impasse* の検出には着目していない。

3. 課題依存型 *impasse* 検出手法の提案

「プログラムの構造比較による検出」「プログラムの実行結果比較による検出」「プログラム自体の比較による検出」の3つの比較に基づく *impasse* 検出手法を提案する。

3.1 プログラムの構造比較による検出

ここでは、2.1 節で示した静的解析機を利用する。この手法は特に「雛形を配布されるタイプのプログラミング演習」に向いている。教師は「学生に配布する雛形のプログラム」*prog 0* を準備し、課題ページで学生に配布する。また、教師は予め「正しく実装ができたプログラム (*prog C*)」「これまでの経験上把握している典型的誤り (W_i)」を含むプログラム (*prog W_i*)」を用意しておく。ここで、「学習者のプログラム (*prog S*)」が与えられたとする。 $matchRate(S, X)$ を「*prog X* を正解 (標準アルゴリズム) と設定したときの *prog S* の一致率」とする。この時、 $matchRate(S, C) < matchRate(S, W_i)$ である典型的誤り W_i が存在したとき、学習者プログラムは、典型的誤り W_i を含むと推測できる。5 章で詳細を述べる。

3.2 プログラムの実行結果比較による検出

ここでは、2.2 節で示した動的解析機を利用する。この手法は特に「入出力仕様は固まっているが実装方法は自由なタイプのプログラミング演習」に向いている。ある典型的誤り W_i を犯しているときに正しく処理できないテストケース (*case C_i*) と、正しく動作してしまうテストケース (*case F A_i*) の両者を用意する。*case C_i* において正しく動作せず、*case F A_i* において正しく動作してしまうプログラムが存在したとき、学習者プログラムは典型的誤り W_i を含むと想定できる。本手法については検討でとどまっております試作・実装は行っていない。

3.3 プログラム自体の比較による検出

静的解析は、静的解析が理解可能な構文以外が使われている学習者プログラムの解析ができない。また、動的解析は、テストケースを入力して実行し実行履歴を比較する関係上、コンパイルおよび実行できるプログラムである必要がある。そこで、静的解析で識別不可能なプログラムや、コンパイルエラー、実行時エラーが含まれるプログラムでも支援ができるように、プログラム自体を *diff* 等で比較する手法を検討した。

diff による比較では、「雛形を渡して実装するタイプ」への対応を考慮する。ここまでと同様、正解プログラム *prog C*、雛形プログラム *prog P*、学生プログラム *prog S*、典型的誤り W_i を含むプログラム *prog W_i* としたときに、以下のような手順で、誤りを含むかどうか判定できると考えている。本手法については検討でとどまっております試作・実装は行っていない。

1. *prog C* と *prog P* の差分を *diff* で取り、*prog C* において、正しく実装すべきプログラムの範囲を求める。
2. 求めたプログラム範囲に対応する、*prog S* のプログラム断片と、*prog W_i* のプログラム断片を *diff* に入力し、差分をもとめる。
3. 差分行数がしきい値を下回った場合、学生プログラムには典型的誤り W_i が含まれると判断できる。

表 1: 課題内容と想定した典型的誤り

課題名	内容・雛形・演習	典型的誤り
課題 A	内容: 自由落下する物体の飛距離の計算	W_{A1} : <code>BufferedReader</code> の使い方がわからず実装できない
	雛形: 固定パラメータでの飛距離計算 演習: 各パラメータのキーボードからの入力	W_{A2} : <code>float</code> 型変数の 0 の初期化を 0.0f ではなく 0.0 としてしまう
課題 B	内容: お釣りの硬貨の枚数カウント	W_{B1} : カウント変数の初期化を忘れてしまい、枚数が正しくカウントできない。
	雛形: 500 円玉のカウントまで 演習: 100 円玉以下のカウント	
課題 C	内容: ピラミッド型に * を表示	W_{C1} : 表示するメソッドに渡す引数がわからず、とりあえずでループ変数 i を渡してしまう。
	雛形: 空白と * を引数分だけ表示するメソッド 演習: 渡されたメソッドを使ってピラミッド型を表示	W_{C2} : ピラミッドを描画する際に改行の出力を忘れてしまう。

4. 検出システムの試作

本研究で提案する課題依存型の *impasse* 検出の流れを以下に示す。なお、本研究では Java の講義での活用を考えているが、利用する静的解析器 [鈴木 07] は C 言語用である。そのため、Java プログラムを静的解析機が理解できる C 言語の表記に変換する枠組みを導入した。

1. コーディング情報を収集する
2. 収集した Java プログラムを C 言語に変換する
3. 先行支援システム [Yamashita 17] による汎用 *impasse* 検出を行う
4. *impasse* が検出された後、 $matchRate(S, C)$ および、 $matchRate(S, W_i)$ が求められる。
5. $matchRate(S, C) < matchRate(S, W_i)$ である典型的誤り W_i が存在するとき、システムは典型的誤り W_i が学習者プログラムに含まれていると判断する。

5. 検出性能の評価

先行支援システムの評価の際に収集された情報系学部 1 年生 (100 名クラス) に開講されたプログラミングにおけるコーディング情報から、無作為に表 1 に示す 3 種類の課題 (課題 A, 課題 B, 課題 C) を選んだ。選んだ課題について、典型的誤りを W_{A1} , W_{A2} , W_{B1} , W_{C1} , W_{C2} の 5 種類用意した。100 名から 10 名を無作為に抽出し、該当課題に対応する述べ 43 の典型的誤りを含むプログラムを収集した。次に、それぞれの課題に対する正解プログラム *prog C_A*, *prog C_B*, *prog C_C*, および、典型的誤りに該当するプログラム *prog W_{A1}*, *prog W_{A2}*, *prog W_{B1}*, *prog W_{C1}*, *prog W_{C2}* を作成し、それらに該当する PAD を生成した。

課題 A の雛形を Program.1 に示す。このプログラムでは、まず整数型で、x 座標、y 座標、x 方向の速度、y 方向の速度

表 2: 典型的誤りを含む行き詰まりの検出性能

	検出された	検出されなかった
典型的誤り	16	25
典型的誤りではない	2	0

の初期値が設定された後、次のステップの位置を推定するプログラムとなっている。

```

1  import java.io.*;
2
3  class FallInt {
4      public static void main(String args[]) {
5          int x, y, x_speed, y_speed;
6          int g;
7          g = -10;
8
9          x = 0;
10         y = 100000;
11         x_speed = 800;
12         y_speed = 0;
13         while( y >= 0 ) {
14             y_speed = y_speed + g;
15             x = x + x_speed;
16             y = y + y_speed;
17         }
18         System.out.print("estimated distance = "
19                             + x + "\n");
20         return;
21     }
22 }

```

Program 1: 課題 A の雛形プログラム

収集したプログラム 43 個に対し、該当する典型的誤り W について、一致率について $matchRate(S, C) < matchRate(S, W)$ が成り立つかどうか判定を行った。判定結果を表 2 に示す。これにより、精度と再現率を計算してみると、精度は $16/18 = 0.89$ 、再現率は $16/41 = 0.39$ となった。精度は 90% 弱であることから、教師が演習の支援に使用しようとした際には一定の実用性があると考えられる。一方で再現率はそれほど高くない。この原因は、構文から外れたプログラムが静的解析できないこと、そもそも準備した典型的誤り以外の誤りが存在したことなどが考えられる。

6. まとめ

教師が課題に依存した行き詰まりに対して、典型的誤りに関する情報を事前に登録しておくことで、課題内容に依存した学習者の行き詰まりを検出できる枠組みを提案した。静的解析による比較、動的解析による比較、プログラムコードによる比較の 3 つを提案した。そのうち、静的解析による比較を用いた課題依存行き詰まりの検出器を施策し、その性能を調査した。

今後は、残り 2 つの検出方法の導入とともに、再現率を向上させるための枠組みを検討する。

参考文献

[野口 10] 野口 靖浩, 小暮 悟, 小西 達裕, 伊東 幸宏: “プログラマ視点を考慮したアルゴリズム学習支援システムの検討”, 人工知能学会 第 60 回 先端的学习科学と工学研究会 予稿集, pp.1-6 (2010)

[大山 11] 大山裕: “アルゴリズム体験ゲーム「アルゴロジック」”, 情報処理, Vol.53, No.3, pp.316-320 (2011)

[伊藤 15] 伊藤栄一郎: “ビジュアルプログラミング環境を用いたアルゴリズム学習支援”, 山梨学院大学経営情報学論集, Vol.21, pp.1-9 (2015)

[Moreno 04] Moreno, A., Myller, N., Sutinen, E.: “Visualizing programs with Jeliot 3”, AVI 04: Proceedings of the working conference on Advanced visual interfaces, pp.373-376 (2004)

[今泉 11] 今泉俊幸, 橋浦弘明, 松浦佐江子, 古宮誠一: “プログラミング学習支援環境 AZUR: ブロック構造と関数の可視化”, 電子情報通信学会技術研究報告, KBSE, 知能ソフトウェア工学, 110(386), pp61-66 (2011)

[Yamashita 16] Yamashita, K., Fujioka, R., Kogure, S., Noguchi, Y., Konishi, T., Itoh, Y.: “Learning Support System for Visualizing Memory Image and Target Domain World, and Classroom Practice for Understanding Pointers”, Proceedings of ICCE2016, pp.521-530 (2016)

[Kogure 13] Kogure, S., Nakamura, R., Makino, K., Yamashita, K., Konishi, T., Itoh, Y.: “Monitoring System for the Semi-Automatic Evaluation of Programs Written During Classroom Lectures,” Proceedings of ICCE2013, pp.339-347 (2013)

[Noguchi 16] Noguchi, Y., Osawa, R., Yamashita, K., Kogure, S., Konishi, T., Itoh, Y.: “Networked Tutoring Support System for a Programming Class based on Reusable Tutoring Content and Semi-automatic Program Assessment,” Proceedings of ICCE2016, pp.252-257 (2016)

[Yamashita 17] Yamashita, K., Sugiyama, T., Kogure, S., Noguchi, Y., Konishi, T., Itoh, Y.: “An Educational Support System Based on Automatic Impasse Detection in Programming Exercises,” Proceedings of ICCE2017, pp.288-295 (2017)

[鈴木 07] 鈴木浩之, 小西達裕, 伊東幸宏: “抽象的データ構造を含むアルゴリズム表現に基づくプログラム評価支援システムの構築”, 教育システム情報学会誌, Vol.24, No.3 (2007)

[王 13] 王子貞, 小暮悟, 小西達裕, 伊東幸宏: “学習者プログラム半自動評価システムへの動的解析の導入”, 教育システム情報学会, 第 38 回全国大会講演論文集, pp.283-284 (2013)

[井垣 13] 井垣宏, 斎藤俊, 井上亮太, 楠本真二: “プログラミング演習における進捗状況把握の為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, No.1, pp.330-339 (2013)

[松澤 14] 松澤芳昭, 保井元, 杉浦学, 酒井三四郎: “ビジュアル-Java 相互変換によるシームレスな言語移行を指向したプログラミング学習環境の提案と評価”, 情報処理学会論文誌, Vol.55, No.1, pp.57-71 (2014)