プログラミング演習における探索的プログラミング行動の自動検出手法の提案

槇原 絵里奈　井垣 宏　吉田 則裕　藤原 賢二　飯田 元

プログラミング演習において，リアルタイムに教員が各学生の進捗を把握し，適切なアドバイスを与えることは困難である．本研究では，探索的プログラミングと呼ばれる，実装が不明確な箇所に対して，修正・コンパイル・実行を繰り返すプログラミング行動に着目する．学生がプログラムのどの箇所に対して探索を行っているかを検出し，課題に対する取り組みや難所をリアルタイムに特定することを目指す．提案手法を実際のプログラミング演習に適用したところ，同一課題における学生間のアプローチの違いや，エラーが生じた原因の特定が容易になることが確認できた．

In programming exercise, it is difficult for educators to grasp each student's progress in real-time and provide them with accurate advice. In this research, we focus on exploratory programming. Exploratory programming is the repetition circle of edit-compile-run when a programmer needs to deal with an unfamiliar portion of source code. Our proposed method identifies how each student solves or struggles each assignment by detecting exploratory programming behaviors. As for the result of applying our proposed method on real programming exercise, we found the difference between approaches performed by each student to the same assignment. Moreover, our proposed method allows educators to identify the factors of compile/runtime errors by students more smoothly.

1　はじめに

近年のソフトウェアの急速な需要拡大を背景に，情報系学部・学科を有するほとんどの教育機関においてプログラミング演習と呼ばれる授業が開講されている [2]．プログラミング演習において，教員はその日に扱うプログラミングの機能 (条件分岐やループ文，配列など) や課題の解説のほか，課題につまずいている学生に対しアドバイスを行う必要がある．しかし，プログラミング演習は全受講生に対し，少数名の教員・ティーチングアシスタント (以下 TA と呼ぶ) で構成されていることが多いため，教員や TA が各学生の課題の進捗や扱っているプログラミング言語に対する理解度を測ることは困難であると考えられる．

以上の問題点を解決するため，本研究では学生が実際にプログラミングを行う上での振る舞い (以下プログラミング行動と呼ぶ) を分析することで，プログラミング演習中に課題につまずいている学生を教員側からリアルタイムに検出する手法を提案する．本研究ではとくに探索的プログラミングと呼ばれるプログラミング行動のサイクルに着目した．探索的プログラミングとは，プログラムの一部の実装方法が不明確である際に，その箇所に対して編集，コンパイル，実行，結果の確認を繰り返し行うことを指す．すなわち，探索的プログラミングを行っている学生はプログラムの特定の機能の実装につまずいている可能性がある．

そこで，本研究では学生の探索的プログラミング行動を自動検出することで，教員や TA に負担を強いることなくリアルタイムに課題やプログラミングにつ

An Automatic Detection Method for Exploratory Programming Behavior in Programming Exercise.
Erina Makihara, Hajimu Iida, 奈良先端科学技術大学院大学, Nara Institute of Science and Technology.
Hiroshi Igaki, 大阪工業大学, Osaka Institute of Technology.
Norihiro Yoshida, 名古屋大学, Nagoya University.
Kenji Fujiwara, 豊田工業高等専門学校, National Institute of Technology, Toyota College.
コンピュータソフトウェア, Vol.35, No.1 (2018), pp.110–116.
[研究論文 (レター)] 2017 年 3 月 10 日受付.

まずいている学生を検出する手法を提案する．まず，我々は探索的プログラミングが行われているプログラムの箇所を抽出するために，既存研究 [5] [4] で述べられている探索的プログラミングの定義を初学者にも適用できるように拡張した．次に，学生がつまずいているプログラミングの機能や箇所をより正確に特定するため，修正箇所の深度とブロックを定義し，それらを自動検出する手法を提案する．

提案手法を実際の初学者向けプログラミング演習に適用したところ，文法の理解不足や，アルゴリズムの理解不足により課題の実装につまずいている学生を検出することができた．

2 探索的プログラミング

2.1 既存研究における探索的プログラミング

探索的プログラミングとは，ソフトウェア開発において開発者が不慣れな言語や API を用いるときや，新しいアルゴリズムを用いる場合に，ソースコードの修正，コンパイル，実行，デバッグのサイクルを繰り返すこと，すなわち複数種類の実装をそれぞれ試行・評価しながら進めていくプログラミングのサイクルを指す [5] [4]．開発者が探索的にプログラミングを進めることにより，開発者は開発対象に対する理解を深め，ソースコードの品質を改善することができると述べられている [4]．そのため，プログラミング初学者でも，新しい知識や技術について学ぶ際に探索的プログラミングを行うことが望まれる [1]．

2.2 初学者向け探索的プログラミングの定義

既存研究において，探索的プログラミングは，実装するプログラムの要求や要求の実現方法が曖昧であるとき，編集，コンパイル，実行，デバッグのサイクルを繰り返すことと述べられている [5] [4]．既存研究で述べられている定義を基に，我々は初学者が行う探索的プログラミングの実態調査を行った [3]．この際，プログラミング演習の形態に合わせ，同一課題の同一箇所を連続して修正している場合に探索的にプログラミングを行っているものとして，探索的プログラミングを検出した．しかし，調査の結果，初学者は同一の課題でも様々な規模で探索を行うことが判明した．

例えば特定の課題において，ある学生は if 文の追加・削除を繰り返し，その都度結果がどのように変化するか確認，すなわち探索的プログラミングを行っていた．一方，他の学生は探索的プログラミングを if 文の条件式に対し行い，また別の学生は if 文の条件式の中でも比較演算子のみに対し行っていた．したがって，初学者の探索的プログラミングを支援する上では，修正がどのブロックをまたいであるいはブロック内で行われているか，また修正された文字数の大小など，修正の規模に応じる必要がある．このように修正が行われる規模を本研究では粒度と呼ぶ．

以上の結果より，我々は初学者による探索的プログラミングを「同一のブロック，行，あるいは行を構成する要素に対して修正およびコンパイル・実行結果の確認が連続で行われていること」と拡張した．本稿で述べる修正とは，文や文字の挿入，削除，入れ替えを指す．ブロックとは通常中括弧 “{}” で囲まれた部分を表すが，ここではそのブロックを特徴づける前後の記述も含めるものとする．また，ブロックが入れ子になった場合を考慮するため，ブロックの深度を定義する．ブロックの深度とは，ブロックが入れ子になった場合，外側から数えて何番目のブロックであるかを表す．例えば，図 1 のソースコードの場合，各ブロックの深度は以下のようになる．

深度 0 main(1 行目)

深度 1 for(4 行目)

深度 2 if(5 行目)，if(8 行目)

本論文では初学者向けプログラミング演習で対象となることが多い C 言語や Java を前提としている．

```
 1:int main(){
 2:int i;
 3:
 4:for(i=0;i<50;i++){
 5:  if(i%3==0){
 6:   printf(“bar”);
 7:  }
 8:  if(i%5==0){
 9:   printf(“foo”);
10:  }
11: }
12:}
```

図 1 複数のブロックの深度を含むソースコード

これらの言語ではブロックを記述する際，中括弧を用いるため，提案する定義によって探索的プログラミングの判定が可能である．一方，Python や Ruby のような中括弧を用いない言語においてもブロックの概念は存在する．したがって，我々が提案した探索的プログラミングの定義をさらに拡張することで，ブロックの記述に中括弧を用いないプログラミング言語の探索的プログラミングも判定可能であると考える．

3 探索的プログラミング行動自動検出の手順

我々は差分情報に基づいた探索的プログラミングが行われている間のプログラミング行動 (以下探索的プログラミング行動と呼ぶ) を自動検出する手法を提案する．提案手法では，まず修正が行われた箇所を内包する，ブロックを特徴づける名前 (main・for・if など) およびそれらの深度の情報を全て検出する．次に，各深度において同一のブロックが修正されている場合，その深度において探索的プログラミングが行われているものと判断する．

図 2 に修正が行われた箇所を内包する全ブロックの名前および深度の検出の手順について示す．以下，各手順について詳述する．

手順 1 連続するソースコードの編集履歴から `Rev.n` と `Rev.n+1` を選択する

手順 2 手順 1 で選択されたソースコードの修正箇所 (差分) を求める．差分の検出には Google-diff-match-patch[†1] を用いる

手順 3 手順 2 で求められた修正箇所を含むソースコードをトークンに分解し，空白を消す

手順 4 手順 2 で求めた修正箇所，すなわち探索が行われた箇所の粒度を手順 3 の結果を基に求める

- **粒度大** ブロック
- **粒度中** 行
- **粒度小** 行を構成する要素

手順 5 手順 2 で求められた修正箇所を内包する全てのブロックの深度，行番号，ブロックを特徴づける名前 (以下，ブロック情報と呼ぶ) を，手順 3 の結果を基に特定する．

手順 6 手順 1 から手順 5 を次に連続するソースコードの編集履歴に対して行う

以上の手順 1 から手順 6 の検出を全ての `Rev.n` と `Rev.n+1` に対し繰り返し行うことによって，一連のソースコードの編集履歴における，修正箇所を内包する全てのブロック情報を得ることができる．

また，複数箇所が修正されていた場合，まず全ての修正箇所に対して手順 1 から手順 6 をそれぞれ行う．次に，検出された修正箇所を内包する全てのブロック情報を深度ごとに比べ，全てに共通するものをその修正におけるブロック情報として検出する．例えば，図 3 の 6 行目と 9 行目が同時修正されていた場合，これらの修正箇所に共通するブロック情報として，深度 0 の main ブロックおよび深度 1 の for ブロックが検出される．

次に，上記の手順 1 から手順 6 によって得られた情報を基に，探索的プログラミングを検出する流れについて説明する．手順 1 から手順 6 を複数回繰り返した例を図 3 に示す．2.2 章に述べたとおり，本研究では探索的プログラミングを「同一のブロック，行，あるいは行を構成する要素に対して修正およびコンパイル・実行結果の確認が連続で行われていること」と定義している．したがって，図 2 の手順で検出された一連の差分情報において，ある区間に同深度で同じ名前のブロックが検出されていた場合探索的プログラミングが行われていたと判断できる．図 3 の場合，深度 0(main 関数内) では全修正において探索的プログラミングが行われている．深度 1 の場合，`Diff.1` から `Diff.5`(`Rev.1` から `Rev.6` にかけての修正) で 4 行目から開始する for 文のブロック内で探索的プログラミングが行われている．深度 2 の場合，`Diff.1` から `Diff.2`(`Rev.1` から `Rev.3`) では 5 行目から始まる if 文内で，`Diff.4` から `Diff.5`(`Rev.4` から `Rev.6`) では 8 行目から始まる if 文内で探索的プログラミングが行われており，この場合は if 文の条件式の書き方や，if 文による条件分岐でつまずいていることが考えられる．

†1 https://code.google.com/p/google-diff-match-patch/

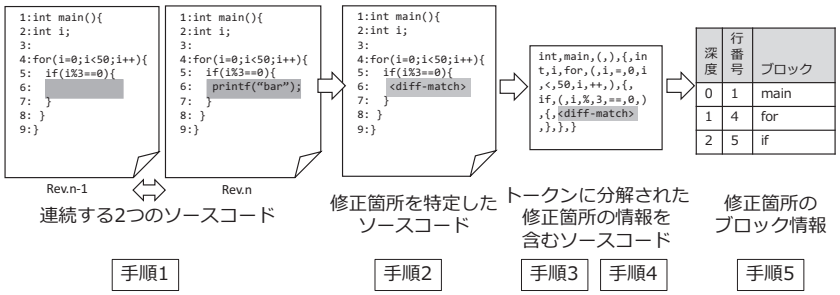


図 2 差分箇所の深度とブロック情報の検出

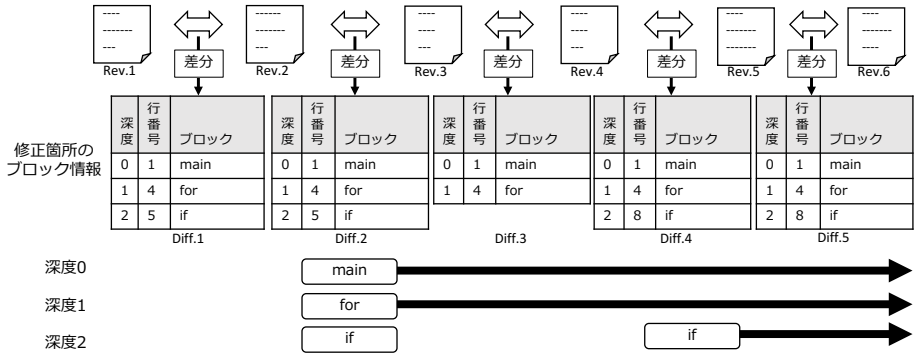


図 3 探索的プログラミング行動の検出

4 実験

4.1 実験概要

実際のプログラミング演習で得られたデータを用いて，3 章で述べた手順にしたがい探索的プログラミング行動の自動検出に関する実験を行う．本実験の目的は，探索的プログラミング行動の自動検出によって得られるデータから，学生のプログラミングに対するつまずきが確認できるか調査を行うためである．以下，本実験で用いたデータについて述べる．

実験対象データは我々の先行研究にあたる探索的プログラミング支援ツールの評価実験を行った際に得たものである [3]．実験参加者は大阪府立大学工業高等専門学校本科 2 年生 41 名であり，C 言語の習得を目的としたプログラミング演習において実験を行った．実験では学生らに 45 分で可能な限り，提案ツールを用いて if 文，for 文，配列など，それまでに講義済みのプログラミングの機能のみで解くことができる課題 4 問を解いてもらった．提案ツールはウェブ上で動作するプログラミング環境であり，学生らはブラウザを介して提案ツール上でソースコードの実装，保存，コンパイル，実行および結果の確認を行うことが可能である．提案ツールは，学生らがソースコードを保存，コンパイル，実行したとき，あるいは 1 分おきに実装中のソースコードをデータベースへ自動保存し，併せてソースコードが保存されたときの学生 ID，時間，ソースコードのスナップショット，操作内容 (保存・コンパイル・実行)，操作結果，出力内容等もログとして保存している．今回の実験では，2.2 章に示した初学者向け探索的プログラミングの定義に当てはまった全てのコンパイル，実行時のログを対象に探索的プログラミング行動を調査した．調査対象にあたるコンパイル時のログは 359 件，実行時のログは 287 件である．

表 1　探索が行われた箇所の粒度と深度の検出結果

(○ はコンパイル・実行時にエラーが生じなかったもの，× はエラーが生じたものを示す)

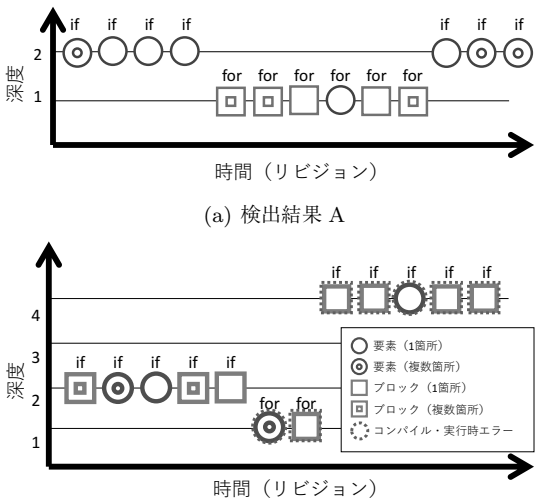
探索の粒度	深度ごとの検出数														
	深度 0		深度 1		深度 2		深度 3		深度 4		深度 5		深度 6		
エラーの有無	○	×	○	×	○	×	○	×	○	×	○	×	○	×	合計
ブロック (1 箇所)	12	18	8	5	2	3	2	1	0	6	0	0	0	0	57
ブロック (複数箇所)	17	20	10	15	4	7	2	1	0	2	0	0	0	0	78
行 (1 箇所)	6	3	3	3	1	3	1	1	2	0	0	0	0	0	23
行 (複数箇所)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
要素 (1 箇所)	24	20	30	22	11	14	6	9	3	4	0	0	0	0	143
要素 (複数箇所)	17	16	26	13	17	5	3	4	2	1	0	0	0	1	105
合計	76	77	77	58	35	32	14	16	7	13	0	0	0	1	406

4.2　検出結果

表 1 に，図 2 の手順 4 にあたる，探索が行われた粒度と深度について検出結果をまとめる．検出にあたって，図 2 で得られた修正箇所のブロック情報のうち，最も深度が深いものを検出対象とした．例えば，図 3 の `Diff.2` では深度 2，4 行目の if 文，`Diff.3` では深度 1 の for 文，`Diff.4, Diff.5` では深度 2，8 行目の if 文が該当する．最も深度が深いものを検出対象とすることで，修正箇所に最も近いブロック，すなわち修正箇所に最も関係があると思われるブロック情報に着目できる．

表 1 より，今回の実験ではほとんどの学生が深度 4 までのブロックで探索的プログラミングを行っていることがわかる．すなわち，深度 6 で探索を行っている学生は，ブロックを過度に入れ子にするといったアルゴリズム上の問題もしくは，文法上の誤りで入れ子が深くなっていることが考えられる．実際に深度 6 で探索している学生のソースコードを確認したところ，if 文の文法を間違えているためエラーが続いており，エラーを消すために試行錯誤する過程で if 文が入れ子になっていた．また，探索が行われる粒度として要素単位の修正が 1 箇所，複数箇所共に多かった．要素単位での探索が行われたソースコードの修正内容を確認したところ，if 文や for 文の条件式部分の修正や比較演算子の修正などが存在した．

次に，探索的プログラミングが行われた粒度と深度およびブロックの名前を学生の課題ごとに可視化を行った例を図 4 に示す．図中の丸は要素単位の探索，四角はブロック単位の探索，丸と四角の上の文字は探索的プログラミングが行われたブロック名を指す．点線で囲まれた丸や四角はコンパイル・実行時にエラーが生じたものを示す．



(a) 検出結果 A

(b) 検出結果 B

図 4　探索的プログラミング行動の可視化

この課題は，与えられた配列の数値がうるう年であるか否かを判定する問題である．うるう年の判定には 3 つの条件があり，学生は for 文で配列の要素を走査し，if 文を組み合わせることで配列の要素がうるう年であるかを判定する．図 4(a) の学生の場合，主に深度 2 の if 文の要素を編集しながら動作を確認しているため，if 文を入れ子にせずに論理演算子を修正しながら動作確認をしていると予測される．また，エラーも発生していない．この学生のソースコードを確認したところ，実際に論理演算子を使用して 3 つ

の条件式を組み合わせており，コンパイルエラーは生じないものの求めている答えと違う出力が行われていたため，条件式や if 文の中で修正を繰り返していた．一方，図 4(b) の学生は途中までエラーが生じていなかったものの，深度 1 の for 文の要素を修正した際にエラーが生じ，そのまま深度 4 の if 文の修正でつまずいていることがわかる．さらに，この図では示していないが，図 4(b) の学生のソースコードには深度 3 の if 文も存在していた．以上より，この学生が if 文を過度に入れ子にしようとして行き詰まっていることが予測される．この学生のソースコードを確認したところ，論理演算子で組み合わせていた条件を 3 つの if 文に分解した際に文法エラーが生じ，その状態で修正を続けていた．

5 考察

図 4 より，探索的プログラミングが行われている深度とコンパイル・実行時エラーがいつ生じたかの情報を組み合わせ，教員に提示することで，教員はその学生がプログラミングのどのような要素でつまずいているか把握が容易になる．例えば，if 文の特定の要素の探索が続いている場合，条件式の文法や条件そのものについての記述について試行錯誤していると考えられる．さらに複数箇所の要素の探索が続いている学生はデバッグを，複数箇所のブロックの探索が続いている学生はアルゴリズムそのものに試行錯誤していることが考えられる．以上のように，プログラムのどの特定の箇所の探索を行っているか，あるいは複数箇所の探索を行っているか，エラーが生じているかといった情報がブロック単位で得られることで，学生がどのような探索的プログラミングを行っているかや，行き詰まっているかを判定することが容易になる．

また，提案手法では構文解析を行わずにソースコードの差分を利用して修正箇所を特定している．差分を用いることで初学者に多いセミコロンや括弧の閉じ忘れなど，文法エラーが含まれたコンパイルが成功しないソースコードでも，リアルタイムに修正箇所の検出やブロックや深度などの情報の検出が可能である．

一方，図 4 で対象とした課題において，コンパイル・実行は行っているものの，main 関数内の探索的プログラミングばかりを行っている学生も見受けられた．これらの学生のソースコードを確認したところ，全てのブロックにおいて使用しているカウンタ変数や printf によるデバッグ文を一斉に修正していた．学生がプログラミングのどのような要素に対しつまずき，探索的プログラミングを行っているのかを教員に提示するためには，探索的プログラミングの粒度や深度の情報に加え，学生がつまずいた箇所の原因や状況までわかりやすくフィードバックするような UI を構築する必要がある．

6 おわりに

本論文では，プログラミング演習における学生の探索的プログラミング行動を検出することで，学生が課題にどのように取り組んでおり，どのような箇所で行き詰まっているかをリアルタイムに特定する手法を提案した．提案手法によって得られた探索的プログラミング行動のログより，同一の課題に対する各学生の異なるアプローチが検出でき，教員は学生のアプローチに則ったアドバイスが可能であると考えられる．

今後の課題としては，教員へ可視化する情報の選定や，UI デザインの改善といった進捗状況を教員がより把握しやすくするための可視化手法と，初学者による良い意味での探索的プログラミングの促進の 2 つが挙げられる．後者について，本研究における初学者向け探索的プログラミングの定義では，初学者の進捗状況の把握や行き詰まり箇所の特定といった目的のために，エラーが残ったまま闇雲にソースコードを変更・コンパイルを繰り返すといった行動も含めて探索的プログラミングとして特定した．今後は，従来研究 [3] における探索的プログラミングのような，複数の実装の評価，すなわちコンパイル・実行が成功するソースコードを維持しつつ，修正を加えていくことでプログラムの完成度を高めるといった良い意味での探索的プログラミングを初学者が行えるよう支援する枠組みについても検討したい．

参 考 文 献

[1] Carnegie Mellon University：Variations to Support Exploratory Programming. http://www.exploratoryprogramming.org/
[2] 独立行政法人情報処理推進機構：IT 人材白書 2014, 独立行政法人情報処理推進機構, 2014.
[3] 槇原絵里奈, 藤原賢二, 井垣宏, 吉田則裕, 飯田元:初学者向けプログラミング演習のための探索的プログラミング支援環境 Pockets の提案, 情報処理学会論文誌, Vol. 57, No. 1 (2016), pp. 236–247.
[4] Sandberg, W. D. :Smalltalk and exploratory programming, *ACM SIGPLAN Notices*, Vol. 23 (1988), pp. 85–92.
[5] Sheil, B.:Environments for exploratory programming, *Datamation*, Vol. 29, No. 7 (1983), pp. 31–144.

槇原絵里奈

2013 年大阪工業大学情報科学部情報システム学科卒業．2015 年奈良先端科学技術大学院大学情報科学研究科博士前期課程修了．現在，同大学院博士後期課程に在学．修士 (工学)．ソフトウェア工学教育，とくにプログラミング教育支援に興味を持つ．

井垣　宏

2000 年神戸大学工学部電気電子工学科卒業．2005 年奈良先端科学技術大学院大学情報科学研究科博士後期課程修了．同年同大学院特任助手．2015 年大阪工業大学情報科学部准教授．博士 (工学)．ソフトウェア工学教育，サービス指向アーキテクチャ，ソフトウェアプロセス等の研究に従事．

吉田則裕

2004 年九州工業大学情報工学部知能情報工学科卒業．2009 年大阪大学大学院情報科学研究科博士後期課程修了．同年日本学術振興会特別研究員 (PD)．2010 年奈良先端科学技術大学院大学情報科学研究科助教．2014 年名古屋大学大学院情報科学研究科附属組込みシステム研究センター准教授．2017 年より同大学大学院情報学研究科附属組込みシステム研究センター准教授 (改組による)．博士 (情報科学)．コードクローン分析手法やリファクタリング支援手法に関する研究に従事．

藤原賢二

2010 年大阪府立工業高等専門学校総合工学システム専攻修了．2015 年奈良先端科学技術大学院大学情報科学研究科博士後期課程修了．同年同大学院博士研究員．2016 年豊田工業高等専門学校情報工学科助教．博士 (工学)．リファクタリングの適用履歴分析，プログラミング教育支援に関する研究に従事．

飯田　元

1988 年大阪大学基礎工学部情報工学科卒業．1991 年同大学大学院博士課程中退．同年同大学基礎工学部情報工学科助手．1996 年奈良先端科学技術大学院大学情報科学センター助教授．2005 年同大学情報科学研究科教授．博士 (工学)．