Vol.2013-CE-121 No.8
2013/10/12

プログラム差分検出ツール「De-Gapper」による授業分析

保福　やよい[1,2,a)]　長 慎也[3]　西田 知博[4]　兼宗 進[2]

概要：プログラミング学習で使用される例題や課題がスモールステップで作成されていることを確認するために、プログラム間の飛躍（ギャップ）を検出するツール「De-Gapper」の開発を行っている。本稿では、大学で行われた Java 言語の授業に対してツールを用いた分析を行った結果を報告する。分析の結果、授業が基本的にはスモールステップで展開されていることを確認するとともに、学生に示される例題と課題においていくつかのギャップの存在を検出することができた。

A class analysis by "De-Gapper" : the tool to detect gaps between programs

Abstract: We are developing "De-Gapper" which is a tool to detect gaps between examples and exercises in materials of programming classes to verify those are small. In this paper we will show the result of analysis of the teaching material for the course of Java programming in a University. As a result, we can verify most programming steps are small. And we also find some samples have big gaps between examples and exercises.

1. はじめに

大学でのプログラミング教育は導入時に Scratch やドリトルなど教育用言語が扱われる場合もあるが、中心は C や Java などの汎用言語である。しかし、汎用言語の学習は教育用言語に比べて学習する内容が多く、以前のことを理解しないまま次の学習に進むこともあり、学習者がドロップアウトすることも多い。[1]

我々は学習のなかで一度にたくさんの学習項目が出現している場所（ギャップ）があり、それがつまづきの原因の一つになっているのではないかと考えた。

新しいことを少しずつ教えていくスモールステップの考え方は初等中等教育を中心に学校教育などで取り入れられているが、我々はこの考え方がプログラミング教育にも有効であると考えた。[2][3][4]

プログラミング言語の学習は、データ型などの文法を理解し、コードを読み、シミュレートをし、コードについての説明ができ、コードが書けるようになるという段階を踏む [5]。

汎用言語の学習に於いては、文法などを正確に理解するためにテキストや教科書は重要で、それに添って学習が進められることが多い。そこで我々は、テキストや教科書における学習内容がスモールステップを実現しているかを自動的に判定するツールを開発した。

このツールを用いることにより、機械的に学習内容のギャップを知り、説明の順番を入れ替える、新しい例題を加えるなどをして、授業を改善することができる。[3]

しかし、実際の授業にツールを適用し、効果を調べることはまだ行われていなかった。

本稿ではツールを過去に行われた Java 言語の授業のテキストに適用し、新出項目の量と学生の理解度の関係を分析した結果を報告する。

2. 関連研究

初学者のプログラム理解のために開発されたツールは多い。ツールの内容は、直接学生を支援するもの、学習履歴を蓄積し、教員に示すものなどに分かれる。

学生を支援する研究としては、プログラミング初学者に対する制御構造と関数の流れを視覚的に表現するツール [6]、型、変数、オブジェクトとデータ参照の理解を促進

1 神奈川県立相模向陽館高等学校
Sagami Kouyoukan High School
2 大阪電気通信大学
Osaka Electro-Communication University
3 明星大学
Meisei University
4 大阪学院大学
Osaka Gakuin University
a) hohuku@amy.hi-ho.ne.jp

© 2013 Information Processing Society of Japan

する Anchor Garden[7]、ソースコードに出現する単語を区別する能力を養う概念学習システム [8] などがある。

学生と教員双方を支援する研究としては、学習者の書いたプログラムや学習者が出したエラーなどの詳細な学習履歴を収集し、学習者の出来を判断したり、学習者に自動的にアドバイスを与えるシステム [9] などがある。

教員を支援する研究としては、学習者の提出履歴などの課題取組状況を収集・分析し、適切な方法で教授者へ通知するシステム [10]、演習中の学生の進捗状況を把握するモニタリングシステム [8] などがある。

これらはいずれも授業中に教員や学生を支援するものであり、授業前にテキスト等の教材についての分析は対象としていない。

3. 構文による差分検出

3.1 スモールステップによる学習

心理学者 Skinner は、教育目標の到達に向かって既習事項を復習しながら、学習内容を段階的に配置して提示するプログラム学習理論を提唱した [11]。この理論は、学習者に合わせて少しずつ学ぶことができるコンピュータ支援教材（CAI）や e-ラーニングなどに影響を与えた。

この理論を応用した学習方式として、高校数学を学習するために作られた公文式学習法 [12] が知られている。公文式の教材は、問題のステップを小刻みに構成し、やさしい問題から、だんだん高度な問題へと直線的につながってる。たとえば、分数の加減については 120 枚ものプリントで構成される。

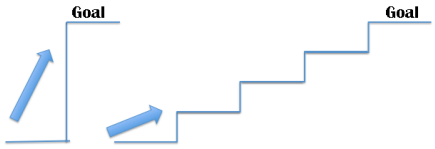

図 1 スモールステップによる学習

3.2 構文によるギャップの検出

スモールステップを実現させるためには，飛躍のある部分に他のプログラムを補足して，飛躍を埋めることが必要である．しかし，飛躍を人間による目視で見つけるのには時間がかかったり，見落としたりする可能性がある．この問題を解決するために，教科書の中のプログラムに現れた新しい学習項目を機械的に発見するツールを開発することが必要であると考えた．

我々は，教材に収録されているプログラムを先頭から順に構文解析し，「新しい構文要素」が多く出現するプログラムを「飛躍のあるプログラム」と考え，機械的に飛躍の検出を行うツール “De-gapper” を開発した [3]．

3.3 De-gapper の実装

De-gapper は，教材に収録されている各プログラムのソースコードを構文解析し，新しい構文要素（それ以前には出てこなかった構文要素）を検出して出力する．

例えば，`printf("Hello");` というプログラムを**図 2** に示すような構文定義をもとに解析を行ったとする．その際に生成された構文木 (**図 3**) をルートから辿ったときに用いられた構文定義の名前を順番に繋げたものを，**図 4** のように「パス名」として列挙する．その中で，以前のソースファイルでは列挙されなかったパス名があれば，それを新しい構文要素として検出する．

De-gapper は，以前に出てきたプログラムと同じ構文定義の要素を使っている場合でも，要素を含むパス名が異なっていれば，新しい構文要素とみなす．例えば，代入文が以前のプログラムに出てきていて，新しく if 文の処理中に代入文があるプログラムを解析する際には「if 文の中に現れる代入文」として新しく検出する．

```
vardecl := type name EQUAL expr SEMICOLON
expr-stmt := expr semicolon
expr := func-call | arithmetic
      | IDENT | NUMNER | LITERAL
arithmetic := expr OP expr
func-call := name arguments
name := IDENT
arguments :=  lparen arg-list rparen
arg-list := expr ( ',' expr )*
```

図 2 構文定義の例 (C 言語，一部)

Fig. 2 Example of grammar definition(some part of C)

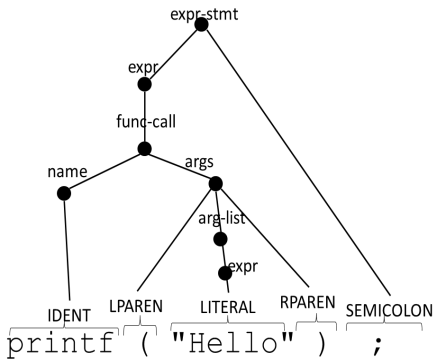

図 3 `printf("Hello");` から生成される構文木

3.4 構文要素の種類

検出された構文要素は，「major な構文要素」と「minor な構文要素」に分類される．minor な構文要素とは，そのような構文が書けるということを，学習者が他の major な構文要素から「容易に類推することができる」[3] 構文要素

```
a-1: expr-stmt
a-2: expr-stmt/expr
a-3: expr-stmt/expr/func-call
a-4: expr-stmt/expr/func-call/name
a-5: expr-stmt/expr/func-call/name/IDENT
a-6: expr-stmt/expr/func-call/arguments
a-7: expr-stmt/expr/func-call/arguments/LPAREN
a-8: expr-stmt/expr/func-call/arguments/arg-list
a-9: expr-stmt/expr/func-call/arguments/arg-list/expr
a-10: expr-stmt/expr/func-call/arguments/arg-list/expr/LITERAL
a-11: expr-stmt/expr/func-call/arguments/RPAREN
a-12: expr-stmt/SEMICOLON
```

図 4 `printf("Hello");` から検出されるパス名

である．

例えば，**図 5** で示した２つのプログラムからは，それぞれ**図 6** と**図 7** で示したパス名が検出される．プログラム 1 では，関数呼び出しの引数に加法式が書ける (1-1)，加法式の中に識別子，演算子及び数値が書ける (1-2,1-3,1-4) ということが検出され，プログラム２では，変数宣言の中に加法式が書ける (2-1)，変数宣言の中の加法式に数値及び演算子が書ける (2-2,2-3)，ということが検出される．しかし，2-2 及び 2-3 が示す内容の本質は「加法式の中に数値及び演算子が書ける」こと，つまり 1-3 及び 1-4 で検出されたものと同様であるので，プログラム 1 を理解していれば「容易に類推することができる」と考え，2-2 と 2-3 は minor な構文要素に分類される．

```
//プログラム 1
int n=10;
printf("%d", n+100);
```

```
//プログラム 2
int n=10+20;
printf("%d", n+100);
```

図 5 minor な新しい構文要素をもつプログラム

```
1-1: func-call/arg-list/expr/arithmetic/
1-2: func-call/arg-list/expr/arithmetic/expr/IDENT
1-3: func-call/arg-list/expr/arithmetic/OP
1-4: func-call/arg-list/expr/arithmetic/expr/NUMBER
```

図 6 プログラム 1 から検出されるパス名（一部）

```
2-1: vardecl/expr/arithmetic
2-2: vardecl/expr/arithmetic/expr/NUMBER
2-3: vardecl/expr/arithmetic/OP
```

図 7 プログラム 2 から検出されるパス名（一部）
Fig. 7 A part of syntax element detected from program 2

3.5 構文要素の種類の解釈

構文の差分は、前節のように major と minor に分けて扱うことで詳細な検討が可能になる。

major は今までのプログラムに出現していない新しい構文を表すため、新たな学習概念の出現を検出していると考えられる。

minor は過去に出現した構文が違った場面で使われる状況であり、プログラムの複雑な度合いを検出していると考えられる。

これらをまとめると次のようになる。major または minor が多い場合は学習のステップにおける段差が大きい状態に、少ない場合は学習におけるステップがない状態に相当し、どちらもスモールステップになっていないと考えられる。

major が多い 複数の新しい概念が出現している。いちどに多くの新しい内容を扱っていないかを確認する必要がある。

minor が多い 新しい概念はないが、プログラムが必要以上に複雑になって居ないかを確認する必要がある。

major と minor が少ない それまでの課題と同じ内容であり、その課題が必要かどうかを確認する必要がある。

4. 授業での評価

開発したツール De-gapper を使い、大学で行われた授業を検証する。最初に、授業結果から授業の難易度が高い箇所を判定し、続いてその箇所が事前の情報（説明用の例題と課題）から De-gapper を用いて検出できることを確認する。

4.1 授業環境

今回は静岡大学情報学部社会学科で 2012 年度後期に文化系の学生を対象に行われた「プログラミング」[13] 科目の授業の授業テキストと授業データを対象に、分析・評価を行った。受講者は前期に Squeak[14] を用いた 90 分間のプログラミングを 4 回体験している 1 年生 108 名、2 年生

2 名である。授業は 2 コマ連続で 15 週間行われた。教員は 2 名、アシスタントは 6 名である。

この授業は Java 言語の習得を目的としたものであり、CE119 で提案されたビジュアル記述型言語とテキスト記述型言語の併用開発環境「BlockEditor」[13] を用いて行われた。学習者はビジュアル言語とテキスト言語の例題を任意に選択して学習できるが、課題についてはどちらかの使用が指定されているものと任意に選択できるものが存在した。本研究で使用した「作業時間」のデータは、学習者がビジュアル言語とテキスト言語を任意に選択できる課題のデータである。

授業の内容を**表 1** に示す。実験授業は全 15 回行われたが、本稿では授業データの得られた 13 回までを対象としている。

分析の対象とした例題と課題は、Web 上で公開されている授業テキスト [15] の内容を使用した。94 個の例題はテキストのプログラムを使用した。71 個の課題はテキストの課題を筆者のうち 2 人が分担して模範解答を作成した。

演習問題 71 題のうち、学習者がビジュアル言語とテキスト言語を任意に選択できる 32 題については、平均作業時間、平均行数などの授業データを得ることができた。これらの 32 題は、ほぼすべての章にわたって存在した。

表 1　テキストと授業の詳細

授業回	章	内容	例題	演習
1	1	環境設定	1	
2	2	タートルグラフィックス（基本命令、逐次処理）	2	4
3	3 4	変数 条件分岐	10	5
4	5 6	繰り返し 繰り返しの入れ子	8	12
5	7 8	オブジェクトの生成 オブジェクトの利用	13	7
6	9 10	画像とアニメーション キー入力とマウス入力	22	4
7	11	CUI プログラム（例題による理解）	3	3
8	11	CUI プログラム（課題による演習）	0	8
9	12	メソッド定義（引数の扱い）	8	5
10	13	メソッド定義（戻り値あり）	10	8
11	23 14	2 次元座標（セルオートマトン） 再帰とフラクタル図形	3 7	5 7
12 13	15 16	集合データ構造 並替えアルゴリズム	3 4	1 2
合計			94	71

それぞれのギャップを調べるため、**表 2** の 4 種類のプログラム群にツールを適用し、major,minor の値を調べた。また、それぞれの累積数を計算し、授業データと比較を

表 2　4 種類のプログラム群

種類	問題数
例題	94
演習問題	71
授業課題	32
例題、演習問題	165

した。

5.　授業全体の分析

授業の難易度については、学生の解答時間に着目した。授業データにおける作業時間と行数を比較した**図 8** を見ると、課題で作成するプログラムの行数はほぼ一定であり、出題する教員は一定の難易度で課題を作成しようとしていることがわかる。一方、課題に要する時間は授業が進むに連れて増加し、大きく 2 箇所で値が増加する。これは学生にとっての難易度を表している。この「教員からは見えにくいが学生には存在する 2 つのギャップ」を検出できることを検証する。

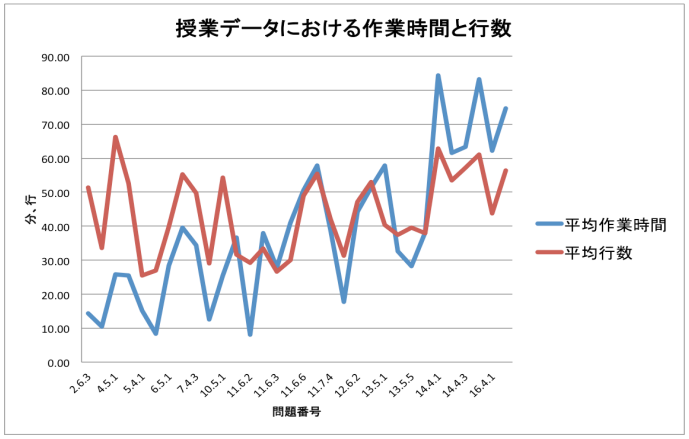

図 8　授業データにおける作業時間と行数

演習問題にツールを実行した**図 9** を見ると、新出概念を示す major はほぼ直線であり学習項目はスモールステップで現れていることがわかる。

一方、課題の複雑さを示す minor は小さなギャップと大きなギャップを示しており、課題のギャップを検出できていることがわかる。これは課題だけでなく例題を含めた**図 10** でも同様である。

これらの結果から説明用の例題や課題を提案ツールで調べることにより、「学生にとって難易度が高くなってしまう箇所を、授業を行う前にチェックすることが可能になる」ことを確認できた。

6.　個々の例題と課題の分析

6.1　授業課題の難易度の検出

個々の課題の作業時間は、その課題に含まれる新しい概念と関係があると考えられる。そこで、課題ごとに累積で

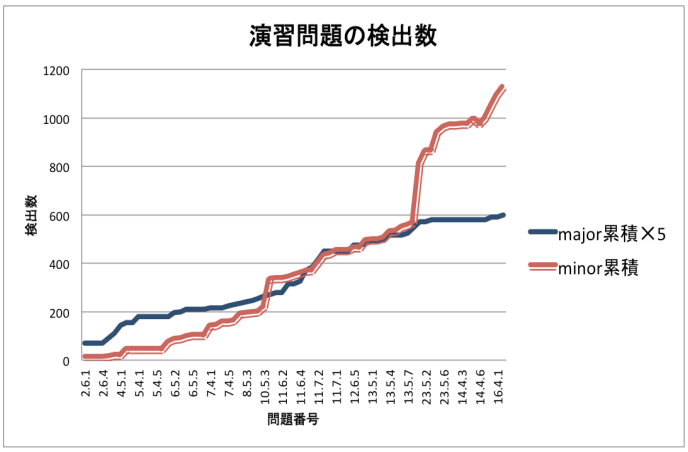

図 9 De-Gapper による演習課題の分析

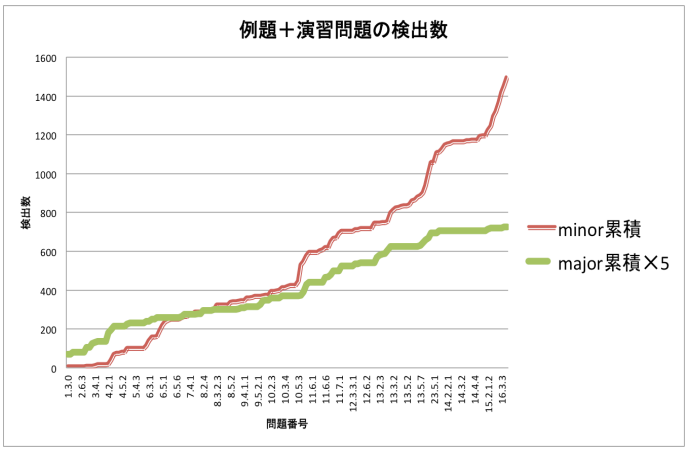

図 10 De-Gapper による例題と演習課題の分析

ない major と作業時間の関係を求め**図 11** に示した。学生の作業時間が長い課題（山のピーク）を見ると、major の値も高いことがわかる。このことから、授業の例題や課題を元に、授業を行う前に例題や課題の難易度を major によって判定できることを確認できた。

なお、グラフの右端の部分では、作業時間が高くなっているにも関わらず、major の変化はない。この部分に相当する授業では、「再帰」「ソートアルゴリズム」などを扱っており、構文としての新しい要素は出現していない。今回の提案手法と De-gapper は「構文による差分検出」を行っていることから、「言語に関する新しい概念の学習」が続いている範囲において有効であることを示している。

6.2 授業課題の複雑さの検出

個々の課題の複雑さはプログラムの量に関係があると考えられる。そこで、課題ごとの minor とプログラムの行数の関係を求め**図 12** に示した。プログラムの行数が長い課題（山のピーク）を見ると、minor の値も高いことがわかる。このことから、課題の複雑さを minor で判定できることを確認できた。

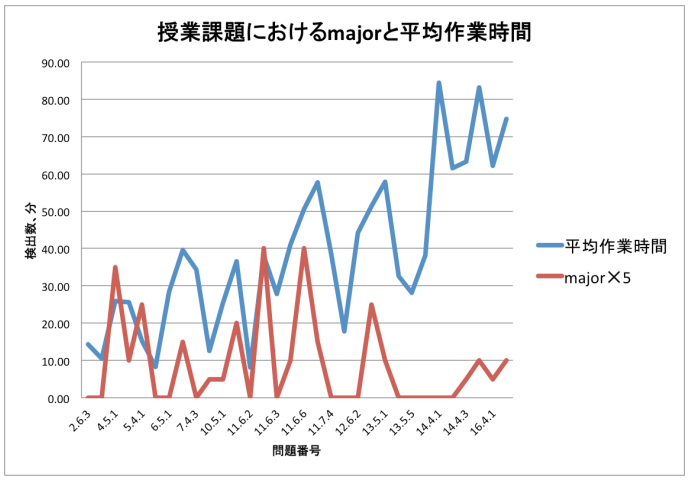

図 11 授業課題ごとの major と作業時間

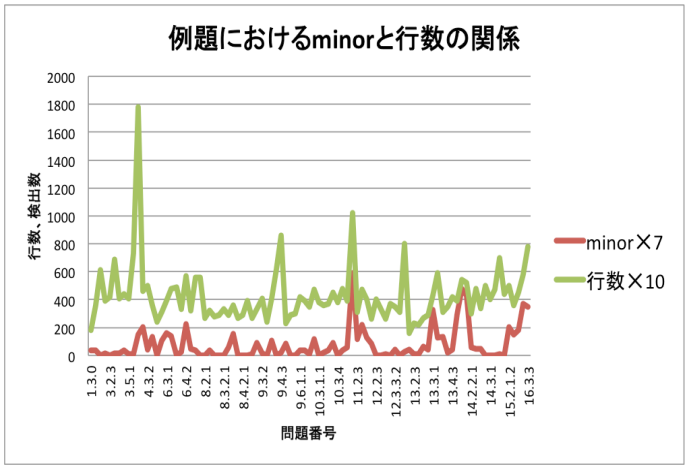

図 12 minor と行数の関係

7. まとめ

プログラミング学習で使用される例題や課題の飛躍（ギャップ）を検出するツール De-gapper を実際の授業データで評価した。その結果、授業の流れにおいて学生にとって難易度の高い授業箇所を事前に検出することができ、個々の例題と課題についても複数の概念が出現する難易度の高い箇所を検出できることを確認した。

今後は、今回の分析で検出された例題や課題について、具体的な原因と改善策を整理するとともに、今回の分析で明らかになった提案手法と提案ツール De-gapper の適用可能範囲について拡張の可能性を検討したいと考えている。

参考文献

[1] Michael McCracken, Vicki Almstrum, Danny Diaz, et al. A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. ACM, SIGCSE Bulletin, Vol.33, No.4, pp.125–180, 2001.

[2] 保福やよい, 西田知博, 長慎也, 兼宗進. なぜプログラミングは難しいのか？ 繰り返しの理解構造と C の教科書分析からのアプローチ. 情報教育シンポジウム (SSS2012),

2012.
[3] 保福やよい, 長慎也, 西田知博, 兼宗進. プログラミング初学者の段階的な理解モデルの検討とツールによる支援. 情報処理学会 コンピュータと教育研究会, CE(118), 2012.
[4] Yayoi Hofuku, Shinya Cho, Tomohiro Nishida, and Susumu Kanemune: Why is programming difficult? Proposal for learning programming in ”small steps” and a prototype tool for detecting ”gaps”. ISSEP2013, Oldenburg, Germany, 2013.
[5] 山本三雄, 関谷貴之, 山口和紀. プログラミングのスキル階層に関する研究, 情報処理学会, コンピュータと教育研究会, CE104, 2010.
[6] 今泉 俊幸, 橋浦 弘明, 松浦 佐江子 , 古宮 誠一. ブロック構造の可視化によるプログラミング学習支援環境 azur〜関数の動作の可視化〜. 情報処理学会研究報告. ソフトウェア工学研究会報告 2010-SE-167(11), 1-7, 2010.
[7] 三浦元喜, 杉原太郎, 國藤進. オブジェクト指向言語における変数とデータの関係を理解するためのワークベンチ. 情報処理学会論文誌. Vol.50, No.10, pp.2396–2408, 2009.
[8] 平澤智明, 高野辰之, 宮川治, 北澤由貴, 古澤資栄, 小澤諒. プログラミングの入門教育を対象とした概念学習システムの開発. 情報処理学会研究報告. コンピュータと教育研究会報告, 2010-CE-107(8), pp.1–8, 2010.
[9] 長慎也, 筧捷彦. proGrep 〜 プログラミング学習履歴検索システム. 情報処理学会研究報告, コンピュータと教育研究会報告, 2005-CE-78(5), 2005.
[10] 長谷川伸. プログラミングの入門教育を対象としたリアルタイム授業支援システム. 情報処理学会論文誌, Vol.58, No.12, pp.3135–3149, 2011.
[11] Burrhus Frederic Skinner. Teaching Machines. Science, Vol.128, pp.969–977, 1958.
[12] 公文式.http://www.kumon.ne.jp/
[13] 松澤芳昭, 酒井三四郎. ビジュアル型言語とテキスト記述型言語の併用によるプログラミング入門教育の試みと成果. 情報処理学会 コンピュータと教育研究会, CE(119), 2013.
[14] Squeak. http://www.squeak.org/
[15] Yoshiaki Matsuzawa. Turtle Cafe2. http://dl.dropboxusercontent.com/u/11409191/turtlecafe2/html4/index.html

© 2013 Information Processing Society of Japan