Vol.2020-HCI-187 No.8
2020/3/16

プログラミング教育における間違い探し問題の自動生成

窪田 開[1,a)]　福里 司[1,b)]　五十嵐 健夫[1,c)]

概要：小学校では 2020 年，中学校では 2021 年からプログラミング教育が必修化される．必修化の主な目的は「プログラミングを体験しながら自分の意図した処理を行わせるための必要な論理的思考力を子ども達に身につけさせること」であるため，プログラミングを教える立場である教員は「どのような課題を解かせるべきか」を常に考える必要がある．そこで本論文では，課題の一つ「ソースコード内の間違いを探す問題」を対象とし、そのような問題を自動生成する手法を提案する．ただし，対象とする言語は C 言語とする．具体的には，ユーザ（教育者）は事前に用意したソースコードと身につけさせたい内容（用意されたリストから選ぶ）を入力する．次に提案するシステムは，ソースコード中の「文字」や「数値」の重要度を計算し，重要度が最大となる文字列を探索する．最後に，得られた箇所に対して，事前に用意した変換ルールを元に変換を適用する．本論文では，教育者によるユーザテストの結果を通して，本手法の有用性を評価する．

1. はじめに

文部科学省では，「プログラミング的思考（自分が意図する一連の活動を実現するために何が必要なのかを論理的に考える力）」の育成を目的とし，小中学校段階の「プログラミング教育」を 2020 年度から計画的に実施することが発表されている [1]．このような背景の下，プログラミング教育を担当する教員は，プログラミング教育の目的に沿った適切な教材や課題を常に考える必要がある．これまでの課題としては，プログラムの一部を空欄にしたソースコードと目的となる処理を提示し，その空欄を埋めさせる「空欄補充問題」が課題として主に用いられている（例：経済産業省が実施する基本情報技術者試験 [2]）．この問題形式は「空欄部にどのような処理を組み合わせれば意図した動作を実現できるのか」を考える力を育成することができる一方，「意図通りに動作しないプログラムをどのように改善したら，意図した活動に近づくのか」を考える力（エラーの原因と改善方法）を学ぶ方法としては適切ではない．以上の点から，育成すべき資質と能力に合わせて異なる問題形式を用意しなければならない [3]．

そこで本研究では，ソースコードの一部に「エラーコード」を挿入することで，その理由と改善方法を考えさせる「プログラミング教育向けの間違い探し問題」について検討する．但し，間違い探し問題を作成するにあたり，ランダムな位置にエラーコードを挿入する場合（例：一番最初の行）や，不適切なエラーコードを挿入した場合（例：存在しない変数を挿入），プログラミング教育の目的である論理的な思考を育成できない可能性がある．つまり，出題意図に合わせて適切な位置とエラーの内容を変更しなければならず，課題を作成する教員にとって大きな手間となってしまう．そこで，本研究では教員の負担軽減を目指し，ソースコード内を解析し出題意図に沿った間違い探し問題を自動生成するシステムを提案する（図 1）．但し，本システムで生成する問題は，既存の空欄補充問題と同様に，回答者は (1) *if* 文や *for* 文などの基本的な文法，型，変数，配列，入出力等の概念は知っている，(2) 基本的なアルゴリズムを学習する段階の方を対象としている．更に我々は「間違い探し問題の形式がプログラミング的思考を育成するのに適切かどうか」「問題の自動生成アルゴリズムの有用性」を検証するために、ユーザテストを実施した。ユーザテストでは、プログラミングの経験が豊富な大学院学生に、教育者側の立場としてシステムの出力を評価してもらった。

2. 関連研究

2.1 プログラミング教育における演習課題

プログラミング教育向けの演習課題としては，主に (1) ゼロからコードを作成する記述問題，(2) プログラム中に存在する空欄を埋める空欄補充問題（図 2），(3) 選択肢の中からプログラム中に存在する空欄を埋める選択式の空欄補充問題の三種類が挙げられる [4]．特に，空欄補充問題

[1] 東京大学
The University of Tokyo
[a)] heihachi@is.s.u-tokyo.ac.jp
[b)] tsukasafukusato@is.s.u-tokyo.ac.jp
[c)] takeo@acm.org

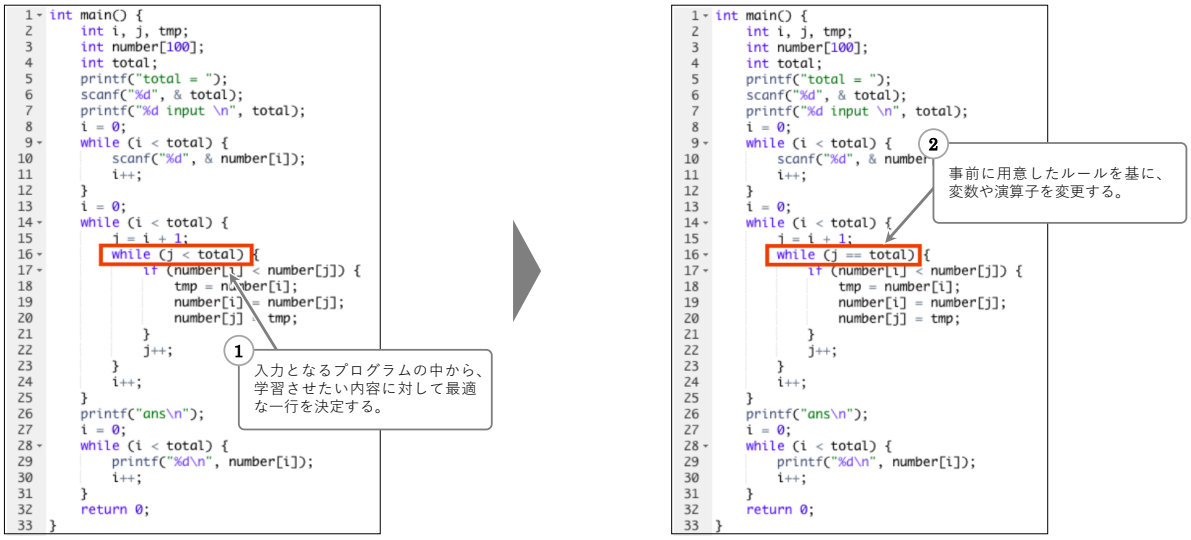

図 1 本研究の概要．教育者が身に付けさせたい学習内容（例：条件分岐）を基に入力ソースコード (a) に対し，エラーを自動挿入することで，間違い探し問題 (b) を生成する．

を中心とした問題は「プログラムを意図どおりに動かすためには何が不足しているのか」を育成する方法として効果的である．しかし，この課題は「意図通りに動作しないプログラムをどのように改善したら，意図した動作に近づくのか」を考える能力をはじめ，プログラミング教育を通じて育成すべき資質や能力を全てを扱うことが難しい．そこで，我々は育成すべき能力に合わせて演習課題を用意する必要であると仮定した．

2.2 プログラミング教育を支援するシステム

プログラミング的思考 [5] を育成することを目的とし，さまざまなプログラミング教育支援システムが提案されている [6][7][8]．田口らは，学習内容や難易度ごとに幅を持つプログラミング演習課題のデータベースを用意することで，各学習者に適切な課題を選出及び出題するシステムを提案した [9]．更に中島ら [10] は，学習者の習得度を基にプログラムコードを作成する記述式の問題の難易度を調整する QA サイクルを提案した（例：習得度が高い学習者にはゼロから，習得度が低い場合はプログラムの一部のみを記述する）．これらの手法は，学習プロセス自体の効率化を実現している反面，どのような課題がプログラミング的思考を育成するために適切かなどといった問題形式自体の検討がなされていない．

その一方，エラーが埋め込まれたソースコードに対し，エラーを自動検出するプログラムを作成するソフトウェア開発者向けのゲームが考案されている [11]．このシステムではエラーの原因と改善方法を考えることで「意図した結果に近づける能力」を育成できることから，プログラミング以外の教育方法として用いられている（例：英語から日本語への自動翻訳結果の「間違い」を利用することで英文法の学習を補助する学習システム [12]）．そこで，我々はプログラミング教育において，プログラムのエラー（間違い）を直す力がプログラミング的思考の育成になると考えた．

```
1  int main(){
2      int k, i, num, small;
3      scanf (“%d”, &k);
4      scanf (“%d”, &small);
5      i = 1;
6      while (i < j){
7          scanf (“%d”, &num);
8          if (                )
9              small = num;
10         i++;
11     }
12     printf (“%d¥n”, small);
13     return 0;
14 }
```

図 2 空欄補充問題の一例 (赤枠部)．

2.3 プログラミング教育用の問題を生成する手法

プログラミング教育向けの問題を作成する方法はこれまでに数多く提案されている [13][14]．内田ら [15] は，プログラミング言語によって規定されている書式やキーワードを覚えることを目的とし，Java で記述されたソースコードの中からランダムに抽出したキーワードと識別子を空欄にする空欄補充問題の生成システムを提案した．しかしこの手法では，(1) 生成された問題が簡単になり過ぎてしまい，教育には不適切になる可能性がある点と，(2) 教育者が意図していない問題となってしまう可能性がある点が課題として挙げられる．そこで，有安ら [16] は，抽象構文木を基にソースコード中の「変数」「変数の値」「条件式」を解析し，教員の意図に沿った空欄補充問題を作成する手法を提案した．また，柏原ら [17] は，C 言語プログラムをプログラム依存グラフ (PDG) を用いて，プログラムコードの

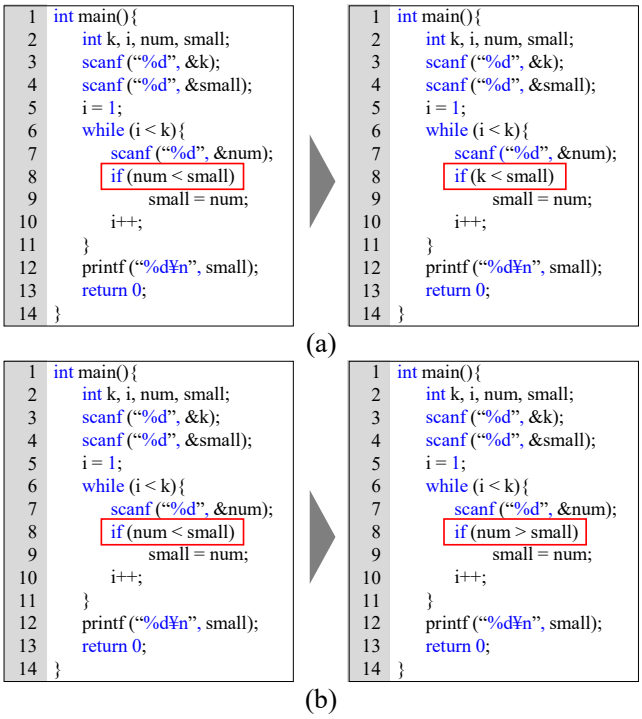

図 3　エラー挿入 (赤枠部) の一例. (a) 変数の変更, (b) 演算子の変更.

全ての行の中から「どの部分を空欄にするべきか」を決定する手法を提案した．更に PDG を用いて作成した空欄位置が適切であるかどうかを調査するために，教育経験のある被験者を対象にユーザテストを行った [18]．そこで我々は，柏原らの手法を参考に，教員が意図する学習内容に対して最適なエラー挿入箇所を決定し，間違い探し問題の自動作成の手法を目指す．

3. 提案システム

本研究で対象とする「間違い探し問題」とは，ソースコードの中からエラーを含む一文を発見し，修正するものである（図 3）．このような問題を作成するために，本研究ではエラーのないプログラムコードと，教員が生徒（学習者）に学習させたい内容を入力情報とする．学習させたい内容としては，プログラミング初心者が間違いやすい内容である (1) 入出力関数（*printf* 関数と *scanf* 関数），(2) *while* 文の条件式，(3) *if* 文の条件式，(4) 制御文の初期値，(5) 制御文の引数，(6) プログラム中の重要文の変数・演算子の六種類の中から一つ選択するものとする．但し，エラーの挿入箇所を複数にした場合，学習させたい内容とは異なる解決策を導き出してしまう可能性（例：条件式の修正の代わりに，周辺の変数等を変更する等）が存在するため，エラーの挿入箇所は一か所のみに限定した．

3.1 エラー挿入位置について

ソースコード内に学習内容に対応する文（例：*if* 文）が

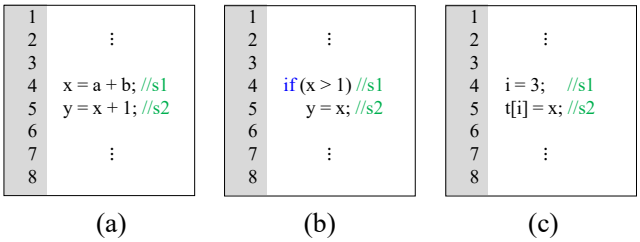

図 4　プログラム中の依存関係. (a) データ依存, (b) 制御依存, (c) アドレス依存.

複数存在する場合，複数の候補の中からエラーを挿入すべき位置を決定する必要がある．そこで我々のシステムでは，既存手法 [17][18] を参考に，PDG を用いたプログラム内の重要文の探索処理を行う（本研究では PDG を frama-c [19] を用いて実装した）．PDG はプログラムの一文ごとにノードを作成し，(1) データ依存関係（例：$s1$ において変数 x が定義され，$s2$ において変数 x が参照されている状態），(2) 制御依存関係（例：$s1$ が条件文またはループ文であり，$s2$ が実行されるかどうかは $s1$ の実行結果に依存する状態），(3) アドレス依存関係（例：$s1$ において変数 i が定義され，変数 i によって参照するアドレスが変更される状態）を基にノード間を接続するものである（図 4）．

次に，候補となるプログラムの行に対する重要度を以下の評価関数 E_1 を用いて計算し，値が最大となる一行を決定する．

$$E_1 = E^i_{data} + E^i_{address} \tag{1}$$

E^i_{data} は，i 行目のデータ依存関係の辺の本数，$E^i_{address}$ は i 行目のアドレス依存関係の辺の本数である．データ依存関係とアドレス依存関係を用いた主な理由として，プログラミング教育において変数や配列などといったデータの変化について問うことが重要であることが挙げられる．式 (1) で得られた値が最大となる行が複数存在する場合，エラーを挿入する一行を決定することは困難である．そこで，本研究では式 (1) で決定できない場合に，以下の評価関数 E_2 を用いて順位付けを行う．

$$E_2 = E^i_{control} \tag{2}$$

$E^i_{control}$ は，i 行目の制御依存関係の本数である．

但し，変数宣言の文や *return* 文にエラーを挿入した場合，変数の意味や出力結果が大きく変化してしまうため，エラー挿入箇所の候補から外している．

3.2 エラー挿入に対するルールについて

プログラムコード内にランダムなエラー（例：ランダムな文字列）を挿入する場合，プログラミング教育の本来の目的である「意図通りに動作しないプログラムをどのよう

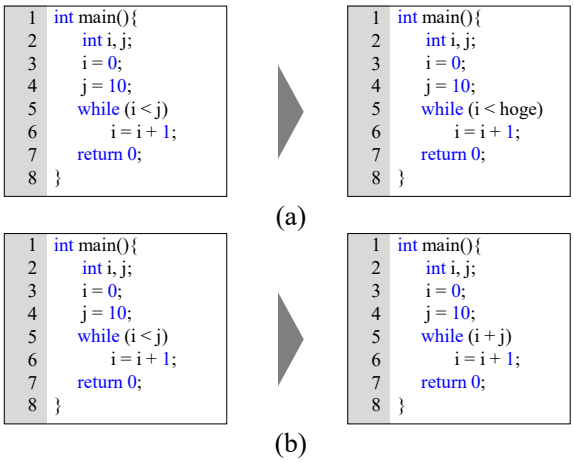

図 5 不適切なエラーの一例. (a) 存在しない変数 “*hoge*” に変更, (b) 条件式として成立しない演算子 “+” に変更.

に改善すればよいのか」を考える能力を育成することは難しい．そこで本研究では，上記で説明した六種類の学習内容に対するルールを設計し，このルールに基づいてエラーコードを作成する．

- 入出力関数： 入出力関数中の変数の書き換え，または「&」のつけ外し，フォーマット指定子の書き換えのいずれかを行う．
- *while* 文の条件式：*while* 文の変数や演算子の書き換え，または = のつけ外しのいずれかを行う．
- *if* 文の条件式：*if* 文の変数や演算子のいずれかを書き換える．
- 制御文の引数：制御文のインクリメントや制御内で更新される変数を書き換える（例：二重ループで同じイテレータ変数に変更する）．
- 制御文の初期値：制御文にある代入文の変数，演算子，数値のいずれかを書き換える．
- プログラム中の重要文の変数・演算子：全ての行に対して重要度（式 (1)(2) を参照）を計算し，値が最大となる一文を検索する（柏原ら [17] の手法と同様）．そして得られた一文の変数または演算子を書き換える．

但し，変数や演算子等をランダムに変更した場合も同様に，エラーの原因を発見することが容易になってしまい，課題教材として適切ではない可能性がある（図 5(a)(b)）．そこで我々は，不適切なエラーの挿入を防ぐための制約条件を追加する．具体的な方法としては，有安らの手法 [16] を参考に，抽象構文木を基にプログラム中に使用されている変数と演算子を抽出する（本研究では，抽象構文木の実装に pycparser [20] を用いた）．次に，変数を書き換える際はソースコード内に存在する変数名を利用する，演算子を書き換える際は，同種類のもの（例：比較演算子を変更する際は，異なる比較演算子に変更する）を用いる．

4. ユーザテスト

提案システムの有用性を確認するために，六名の被験者 ($P1, P2, \cdots, P6$) を対象とする二種類のユーザテストを実施した．一つ目は空欄補充問題と間違い探し問題の比較，二つ目はエラーを挿入する位置の妥当性の検証である．但し，参加した被験者はいずれもプログラミング歴三年以上，C 言語を用いたプログラミング経験を有する方であるため，教育者側の立場として回答してもらっている．

4.1 実験手順

実験方法としてはいずれも，事前に用意した「学習させたい内容」と「演習課題（プログラムコード）」を各被験者に与え，それぞれに対してアンケート項目に回答してもらう形とする．演習課題として用意したプログラムは，(1) n 個の整数の最小値を出力するプログラム，(2) n 以下の素数を出力するプログラム，(3) バブルソートを用いて n 個の整数を降順に出力するプログラム，(4) 選択ソートを用いて n 個の整数を昇順に出力するプログラムの四種類，学習内容は (1) *while* 文の条件式，(2) *if* 文の条件式，(3) プログラム中の重要文の変数や演算子の三種類とし，これらを組み合わせることで演習課題を作成した．但し，学習者（生徒）はいずれも「変数などの基本的な概念，*if* 文や *while* 文などの基本的な文法を理解した上で，それらを組み合わせたプログラムを学び始めた」段階とする．

4.2 空欄補充問題 vs 間違い探し問題

一つのプログラムから (1) 提案手法で作成した間違い探し問題（E）と，(2) 本システムで決定したプログラムの一行を空欄にすることで作成した空欄補充問題（B）の二種類のペアを用意し，それぞれに対する以下の質問項目を回答してもらった．

- 五段階リッカートスケール（1: とても簡単，5: とても難しい) を用いた問題形式の難易度評価とその理由
- どちらの問題形式がアルゴリズムの理解に役立つと思うかとその理由
- どちらの問題形式がプログラミング教育に役立つと思うかとその理由
- プログラミング教育を実際に行う場合，どちらの問題形式を採用すべきか（両方採用，不採用も含む）とその理由

但し，問題形式自体の慣れによるバイアス（後半に提示した問題形式が簡単に見えてしまう可能性）を軽減するために，上記のペアを二つ用意し，一度目は空欄補充問題，間違い探し問題の順番，二度目は間違い探し問題，空欄補充問題の順番で回答してもらった．

4.3 エラー挿入位置について

一つのプログラムコードから，(1) 学習内容に対し，最も重要であると判定された一行に対してエラーを挿入したもの（H）と (2) 学習内容に対し，最も重要でないと判定された一行にエラーを挿入したもの（L）の二種類のペアを用意した．次に被験者にこのペアを与え，「どちらの問題がアルゴリズムの理解に役立つかとその理由」「どちらの問題がプログラミング教育に役立つと思うかとその理由」「どちらの問題が学習内容に即していると思うかとその理由」を回答してもらった．但し，被験者にはどちらの問題が重要度の高い文であるかは提示をしないものとする．

4.4 実験結果

4.4.1 空欄補充問題 vs 間違い探し問題

実験結果は表 1 と表 2 に示す．問題形式の難易度については，空欄補充問題は平均 2.33（分散 1.22），間違い探し問題は平均 3.33（分散 0.89）となり，間違い探し問題のほうが難易度が高い結果が得られた．その主な理由として，「空欄補充問題では，コードのどこにエラーが含まれているかを探す必要がない一方，間違い探し問題はエラー箇所を探す工程と，修正する工程の二段階に分かれているため難しい」ことが挙げられる．一方，空欄補充問題の方が難しいと答えた被験者は，「学習内容が既知であるため，間違い探し問題でもプログラムの挙動を想像しやすかった」との回答している．

プログラム自体の挙動を理解する際，被験者四名が間違い探し問題のほうが適切であると回答した．その主な理由として「間違い探し問題の方が回答の自由度が高い」「空欄補充問題では空欄付近のみを見れば回答できるものの，間違い探し問題はコード全体を理解する必要がある」との意見が得られた．一方，空欄補充問題を選んだ二名は「間違い探し問題に必要なのはデバッグ能力である」「空欄補充問題は一部ではあるもののアルゴリズムを理解してゼロから書く必要がある」との意見が得られた．

学習面においては，被験者三名人の被験者が間違い探し問題を選んだ．理由としては，「間違い探し問題は実際のプログラミングに必要な能力かつ，空欄補充問題では鍛えづらいデバッグ能力を鍛えられる」ことが意見として得られた．その一方，空欄補充問題を選んだ三名からは「空欄補充問題はプログラムの重要部分に集中できるため，特定の学習には適切である」などの意見も得られた．

今後のプログラミングの教育上どちらの問題を採用すべきかについてが，一名は間違い探し問題，一名が空欄補充問題，四名は両方必要であると回答した．両方必要であると回答した四名は，「それぞれの問題形式の難易度が異なる」「空欄補充問題はプログラムを作る，間違い探し問題はプログラムを直すといった鍛える能力が異なる」ことを回答した．空欄補充問題を選んだ一名については，「間違い探し問題はプログラムの非本質部分に時間を割かれるため，演習課題としては適切と言えない」と回答した．

表 1　問題形式に対する難易度の違いについて．

#		P1	P2	P3	P4	P5	P6
1	空欄補充問題 (B)	2	1	3	4	3	1
2	間違い探し問題 (E)	4	2	4	2	4	4

4.4.2 エラー挿入位置について

実験結果を表 3 に示す（以下，重要度の高い文を書き換えた問題を「一問目」，重要度の低い文を書き換えた問題を「二問目」とする）．

プログラム自体の理解の面では，被験者四名が一問目が適切であると回答した．その主な理由としては，「一問目はよりアルゴリズムの本質部分に近かったため」「二問目では，特に *printf* 文に関するエラーの際，アルゴリズムの理解なしに答えられたため」などの意見が得られた．二問目を選んだ二名は，「一問目は特に配列に関する代入文の場合，一文だけでエラーの原因を簡単に判断できたため」「一問目は変数の値が定義されていない変数に書き換えられていたので，すぐに違和感を感じたため」と回答した．

学習の有用性については，三名の被験者が一問目が適切であると回答した．主な理由として，「一問目はプログラムの流れを理解する必要があるため」などの意見が得られた．二問目を選んだ三名のうち一名に関しては，「エラー挿入位置ではなく，二問目のエラー内容が自然であったため」と回答した．

学習内容に即しているかについては，四名の被験者が一問目が適切であると回答した．その理由としては「一問目は変数の動きやどのような計算がなされているかを理解する必要がある」「二問目のエラーの位置はアルゴリズムの全体の流れを考える必要がないため」などと回答した．一方で二問目を選んだ二名の内一名は「一問目の結果は制御構造に関係しないため」と回答した．

5. 議論

ユーザテスト結果を基に，間違い探し問題自体（問題形式）とその自動生成システムについての議論を行う．問題形式については「今後のプログラミング教育上どちらの問題を採用すべきか」の結果から，間違い探し問題と空欄補充問題は役割に応じて使い分ける必要があると考えられる．つまり，それぞれの問題形式を使い分けるために，各問題形式の特徴を確認する必要がある．間違い探し問題については，エラーが挿入されている箇所を探す工程と改善方法を考える工程の二段階で構成される．その一方，空欄補充問題は解くべき部分を探す必要はない．つまり，間違い探し問題は回答方法の自由度が高くなり，難易度も高くなる傾向にある．更に，どこが間違っているかを見つけるスキルが試されるため，プログラム全体へのより深い理解が求

表 2　問題形式に対する学習支援について．空欄補充問題 (B) vs 間違い探し問題 (E)．

#		P1	P2	P3	P4	P5	P6
1	プログラム自体の理解にはどちらの問題形式が適切か	B	E	E	B	E	E
2	学習にどちらの問題形式が役立つか	B	E	B	E	E	E
3	今後のプログラミング教育においてどちらの問題形式を採用すべきか	Both	Both	Both	Both	E	B

表 3　エラー挿入位置に対する学習支援について．重要度高 (H) vs 重要度低 (L)．

#		P1	P2	P3	P4	P5	P6
1	プログラム自体の理解にはどちらの挿入位置が適切か	H	H	L	H	H	L
2	学習にどちらの位置が役立つか	H	H	L	L	H	L
3	プログラミング教育の導入目的にどちらが合っているか	H	H	L	H	H	L

められる．空欄補充問題については，局所的な理解となる可能性はあるものの，プログラムの重要部分のみに集中することができる．つまり，それぞれの問題形式を通して育成可能な資質や能力としては，間違い探し問題はコードを読んでデバッグする能力，空欄補充問題は頭でイメージしたアルゴリズムを（局所的に）ゼロから書く能力が挙げられる．まとめると「難易度が高い」「全体的な理解」「直す能力」を目的とした際は間違い探し問題，「簡単」「局所的な部分への集中」「一から書く能力」を目的とした場合は空欄補充問題を採用するべきであると言える．

次に，間違い探し問題を自動生成する上でエラーを挿入する位置について考察する．ユーザテストの結果から，提案システムで重要であると判定された行は学習内容に即し，アルゴリズムの本質部分に近いと判断できる．但し，被験者二名から「適切ではない」と判断されてしまった原因として，(1) 配列に関する簡単な代入文が選ばれた場合，(2) 変数の値が定義されていない変数に書き換えられてしまったことが挙げられる．(1) については，要素を変数で指定した配列の場合（例：*for* 文）は複数の変数から得られる重要度を加算してしまうため，重要であると判定されやすい事が原因として考えられる．今後は，単純な加算ではなく重み係数等を用いて，上記の問題を軽減する予定である．また (2) については，重要文の計算方法ではなく間違い生成の方法に問題があることが考えられるため，変数や演算子をより適切なものに書き換えるための制約を追加することで改善が可能であると考えられる．

6. 今後の課題

本システムは，現在エラーを挿入する箇所を一行に限定している．しかし，実際のプログラムでは変数の入れ替えをはじめ複数行で一つの動作を指定する状況が多々存在する．そこで今後の課題として，複数行に対してエラーを挿入する機能を実装する必要がある．また，プログラムコードの中から重要な行を決定する方法として，現状はデータ依存とアドレス依存の二つを主に用いているが，依存関係自体の優先度（重み係数）を変更することで，より詳細な学習内容への対応や問題の難易度を調整する機能（学習者の習熟度の考慮）も検討する必要がある．更に，上記の機能を有するインターフェースを作成し，今後は教員にとってより使いやすいシステムを目指す予定である．

最後に「間違い探し問題」自体を他分野（例：ゲーム分野）に応用することを検討している．例えば，将棋の電王戦や囲碁での AlphaGo において動作の最適化 [21][22] が注目されている．そこで，我々は動作決定の段階で意図的に「悪い手」を提示することで，学習者に「なぜこの動きが良くないのかを考える力」を育成できると考えている．

7. むすび

我々は，プログラミング的思考を育成する方法として「間違い探し問題」を提案し，更に間違い探し問題を自動生成するシステムを提案した．具体的にはプログラム依存グラフ（PDG）に基づいてエラーを挿入する位置を計算，及び構文木を基に変数及び演算子の書き換えを行った．また，空欄補充問題との比較実験を実施することで，間違い探し問題と空欄補充問題それぞれの利点と，用途に応じて使い分ける必要があることを示した．更に，エラーを挿入する位置の計算についても，ユーザテストを通して適切であることを示した．現在の小中学校の教員たちは，プログラミング教材を大量に用意する必要があることからも，本研究によって効率的かつ大きな改善が期待できる．

参考文献

[1] 文部科学省：小学校プログラミング教育の手引 (第二版)，https://www.mext.go.jp/component/a_menu/education/micro_detail/__icsFiles/afieldfile/2018/11/06/1403162_02_1.pdf (2018).
[2] 高田美樹：基本情報技術者 らくらく突破 C 言語 (2019).
[3] Anderson, J. R., Boyle, C. F., Corbett, A. T. and Lewis, M. W.: *Cognitive Modeling and Intelligent Tu- toring*, pp. 7–49, The MIT Press (1990).
[4] 竹内亮太郎，大久保弘崇，粕谷英人，山本晋一郎：空欄補充問題の自動生成による Haskell プログラミング学習支援環境，第 171 回ソフトウェア工学研究会，pp. 15:1–15:8 (2011).
[5] 山本利一，本郷　健，本村猛能，永井克昇：初等中等教育におけるプログラミング教育の教育的意義の考察.
[6] Brown, M. H.: Zeus: A System for Algorithm Animation and Multi-View Editing, *Proceedings of IEEE Workshop*

on Visual Languages, pp. 1–22 (1991).

[7] Kasahara, R., Sakamoto, K., Washizaki, H. and Fukuzasa, Y.: Applying Gamification to Motivate Students to Write High-uality Code in Programming Assignments, *Proceedings of the ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '19)*, ACM, pp. 92–98 (online), DOI: 10.1145/3304221.3319792 (2019).

[8] Ishizue, R., Sakamoto, K., Washizaki, H. and Fukazawa, Y.: PVC: Visualizing C Programs on Web Browsers for Novices, *Proceedings of the 49th ACM Technical Symposium on Computer Science Education (SIGCSE '18)*, ACM, pp. 245–250 (online), DOI: 10.1145/3159450.3159566 (2018).

[9] 田口　浩，糸賀裕弥，毛利公一，山本哲男，島川博光：個々の学習者の理解状況と学習意欲に合わせたプログラミング教育支援，情報処理学会論文誌，Vol. 48, pp. 958–968 (2007).

[10] 中島秀樹，高橋直久，細川宜秀：プログラミング学習のための QA サイクル: 受講者の習得度に応じた問題自動提示メカニズム，電子情報通信学会論文誌，Vol. J88-D1, No. 2, pp. 439–450 (2005).

[11] Clegg, B. S., Rojas, J. M. and Fraser, G.: Teaching Software Testing Concepts Using a Mutation Testing Game, *Proceedings of IEEE/ACM 39th International Conference on Software Engineering: Software Engineering Education and Training Track (ICSE-SEET)*, pp. 1–4 (online), DOI: 10.1109/ICSE-SEET.2017.1 (2017).

[12] Nagai, K.: A Machine Translation System for Learning English, *Bachelor Thesis*, The University of Tokyo (2015).

[13] 山本　樹，國宗永佳：初級プログラミング学習者に対するプログラム概念理解のための支援方法の実施報告，第 39 回 教育システム情報学会 全国大会，Vol. F5-2, pp. 451–452 (2014).

[14] 長谷川靖成，大竹　諒，田島侑典：プログラミング学習における意図を考慮した空欄補充問題の自動生成，修士論文，南山大学 (2012).

[15] 内田保雄：初級プログラミング学習のための自動作問システム，技術報告 123(2007-CE-092)，宇部工業高等専門学校 経営情報学科 (2007).

[16] 有安浩平，池田絵里，岡本辰夫，國島丈生，横田一正：学習者に合わせたＣ言語演習穴埋め問題の自動生成，第 1 回データ工学と情報マネジメントに関するフォーラム (DEIM フォーラム)， Vol. D9-5:1–D9-5:5 (2009).

[17] Kashihara, A., Terai, A., Jun' ichiToyoda : Making Fill-in-Blank Program Problems for Learning Algorithm, *Proc. (Vo.1) of International Conference on Computers in Education 99*, pp. 776–783.

[18] 柏原昭博，久米井邦貴，梅野浩司，豊田順一：プログラム空欄補充問題の作成とその評価，人工知能学会論文誌，Vol. 16, No. 4, pp. 384–391（オンライン），DOI: 10.1527/tjsai.16.384 (2001).

[19] CEA-List and Inria: Frama-C, `https://frama-c.com/` (online-accessed: 2020.01.20).

[20] Bendersky, E.: pycparser v2.19, `https://github.com/eliben/pycparser` (online-accessed: 2020.01.20).

[21] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T. P., Simonyan, K. and Hassabis, D.: Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm, *CoRR*, Vol. abs/1712.01815 (online), available from ⟨http://arxiv.org/abs/1712.01815⟩ (2017).

[22] Chen, Y., Huang, A., Wang, Z., Antonoglou, I., Schrittwieser, J., Silver, D. and de Freitas, N.: Bayesian Optimization in AlphaGo, *CoRR*, Vol. abs/1812.06855 (online), available from ⟨http://arxiv.org/abs/1812.06855⟩ (2018).