情報処理学会論文誌 Vol. 49 No. 10 3366–3376 (Oct. 2008)

アルゴリズム学習における間違い探し形式の演習課題を自動生成する手法の提案と評価

長瀧寛之[†1] 伊藤亮太[†1] 大下福仁[†1]
角川裕次[†1] 増澤利光[†1]

本論文では，アルゴリズム学習のための演習問題の自動生成手法を提案する．ここでの演習問題とは，あるアルゴリズムを実装したソースコードに誤りを含ませたものに対して，誤りの修正および考察を行う形式を対象としている．演習問題の自動生成においては，演習を通じてアルゴリズムの理解を深められる誤りの自動生成が必須である．我々は正しいソースコードに誤りを自動挿入するシステムを試作した．また，提案手法による誤り挿入の有用性について評価を行い，その結果をもとに提案手法を改善した結果，手作業による誤り挿入と同等程度の有用性を得られることを確認した．

A Fault Injection Method for Generating Error-correction Exercises in Algorithm Learning

Hiroyuki Nagataki,[†1] Ryota Itoh,[†1]
Fukuhito Ooshita,[†1] Hirotsugu Kakugawa[†1]
and Toshimitsu Masuzawa[†1]

In this paper we propose a method for generating error-correction exercises for undergraduate students in computer science who learn algorithms. Our main goal is to inject faults automatically into a correct source code that implements an algorithm to be studied. The proposed method utilizes design paradigm of the algorithm to determine effective fault types and positions in a source code. We have developed a prototype system and evaluated the appropriateness of the generated exercises to algorithm study.

†1 大阪大学大学院情報科学研究科
Graduate School of Information Science and Technology, Osaka University

1. はじめに

アルゴリズムの学習は，情報科学の学習において重要な分野の 1 つである[1)]．アルゴリズムの講義では，学習者の理解度向上を狙った取り組みとして，講義で扱うアルゴリズムをプログラミング言語で実装する形式の演習が行われている．

コーディングを正しく行うためには，まず実装しようとするアルゴリズムの内容を十分に理解している必要がある．つまり，アルゴリズムの動作を再確認する契機となる点で，プログラミングはアルゴリズムの理解を深めるのに効果的な演習形式と考えられる．一方この演習形式は，実装言語の仕様やソースコードのデバッグなどに学習者の注意が向きやすく，演習本来の目的であるアルゴリズムの理解に学習者を集中させにくいという問題点がある[2)–4)]．

この問題点に対するアプローチとして，我々はソースコードにおける**間違い探し演習**を提案する．本研究で提案する間違い探し演習の手順は以下のとおりである．(1) 教師は，あるアルゴリズムを実装したソースコードを学習者に提示する．ただしソースコードには，アルゴリズムと矛盾する誤り箇所が 1 つ以上含まれている．(2) 学習者はソースコードに含まれているすべての誤りを発見し，実装すべきアルゴリズムと比較してどのような矛盾が生じているかを考察する．(3) 学習者は，アルゴリズムに合致するようソースコードの誤りを修正する．

間違い探し演習では，学習者はソースコード全体を自分で実装する必要はない．よって，構文エラーなどアルゴリズムの理解と関係のない事柄に学習者が注意を払う必要が減るので，教師はアルゴリズムの理解に焦点を絞った演習を行うことができる．また，学習者の解答は，問題として提示したソースコードの一部を書き換えただけなので，教師にとっては一から作成したソースコードをチェックする場合に比べて採点作業の時間的負担が軽減される効果も期待できる．さらにこの効果は，1 度に多数多種の演習問題を扱う演習を教師が実施しやすくなるという期待にもつながる．

本論文では，間違い探し演習の問題を自動生成することを目標として，アルゴリズムの理解を深めるという演習目的に適した誤りをソースコード中に自動挿入する手法を提案する．以下，2 章では本研究が想定する間違い探し演習の詳細および問題作成の自動化について検討する．また，3 章では具体的な問題作成の自動化手法を紹介し，4 章では提案手法の実装について概要を述べる．5 章では実際の授業での間違い探し演習の効果を確認し，そのうえで 6 章で提案手法の評価について議論する．

2. 演習形式の検討

2.1 演習問題の形式

間違い探し演習は，講義で扱ったアルゴリズムの理解度向上がその目的である．つまり学習者は演習を通して，学習したアルゴリズムの具体的な手順を振り返り，知識を再確認することが求められる．

そこで間違い探し演習では，ソースコードに含まれるアルゴリズムについて，学習者に“アルゴリズム名”だけを情報として提示する．ソースコードに含まれるアルゴリズムとの矛盾を発見，修正するために，学習者は提示されたアルゴリズムの具体的な手順を自身の知識から思い出し，そのアルゴリズムの手順がどのようにソースコード上に実現されているかを把握する必要がある．つまり学習者は演習問題の解答時はつねにアルゴリズムの動作を意識することになり，結果アルゴリズムの知識を再確認する目的が達せられると考えられる．

演習問題としては，アルゴリズム上の誤りが存在するソースコード中の該当箇所，該当箇所の修正結果を解答として記述する問題を学習者に与える．コーディングではなくアルゴリズムの動作に学習者の意識を集中させるため，学習者は演習中にソースコードの実行環境を使えないこととする．また，修正前後のソースコードについてアルゴリズム上の差異を説明する問題を提示する．ソースコードの修正がアルゴリズムに及ぼした影響を考察させることで，学習者にアルゴリズムの動作へ意識を向けさせることを狙う．

2.2 演習に適した誤り

間違い探し演習においてソースコードに含まれる誤りは，その発見と修正によって学習者の正しいアルゴリズムの理解へつながるものでなければならない．そこで間違い探し演習では，以下の 2 つの条件を満たすソースコード中の誤りを**演習に適した誤り**と定義する．(1) その誤りを含むソースコードは，問題文で指定されたアルゴリズムの手順と矛盾した動作を行う．(2) 学習したアルゴリズムの手順を覚えていなければ，誤りを発見あるいは修正することは困難である．

構文エラーは，その発見と修正の作業がプログラミングの知識のみで行えるため，アルゴリズムの学習という目的に適さない．よって演習に適した誤りではないと定義する．また，`i = i`（代入文の左辺と右辺が同じ）など，コーディング自体に不自然さが含まれている誤りの場合，学習者がアルゴリズムとの矛盾を意識せずに誤りを発見，修正できてしまう可能性がある．この場合もアルゴリズムの演習目的に沿わないため，演習に適した誤りではないと判断する．

2.3 演習問題の自動作成

間違い探し演習の問題を手作業で作成する場合，教師はあらかじめ学習させたいアルゴリズムを実装したソースコードを用意し，そのソースコードの一部を修正して誤りを含んだ形にするという作業を行う．このとき，ソースコードに含ませる誤りが演習に適したものかどうかを検討しながら慎重に問題作成を行う必要があるため，教師にとって問題作成は負担が大きい作業となりうる．

そこで本研究は，間違い探し演習の問題作成を容易に行う手段を提供すること，つまり与えられたソースコードから，演習に適した誤りを含んだソースコードを自動的に生成する手法の構築を目標とする．これにより，教師の問題作成における負担は軽減され，間違い探し演習の導入が容易になることが期待できる．

さらに演習問題作成の自動化によって，1 つのソースコードから複数の演習問題を生成したり，1 つのアルゴリズムに対して実装の異なる多数のソースコードから演習問題を作成したりといった作業が容易となる．その結果，たとえば学習者ごとに異なった演習問題を提示することで他人の解答を書き写す不正の防止を図ったり，学習者ごとの理解に合わせて難易度の異なる問題を提示したりなど，演習方式の多様化が期待できる．

2.4 関 連 研 究

アルゴリズムの設計を通じた理解支援を目的とした研究には，アルゴリズム設計支援ツールである RAPTOR[4] や JPADet[2]，SFC[5] などがある．学習者が構文エラーなどに気を取られずに学習を行う環境を実現するという目標は本研究と同じであるが，上記の研究はいずれも直感的なアルゴリズム設計を可能とする手段として，独自のグラフィカルなアルゴリズム設計環境を提供するというアプローチをとっている．一方本研究は，学習者がすでに経験している既存のプログラミング言語をそのまま活用できるという点が異なる．

演習問題の自動生成に関する研究としては，ソースコードから空欄補充問題を自動生成する文献 6) がある．空欄補充問題は学習者にとって考えるべき箇所が空欄として明確に分かるのに対し，間違い探し演習の場合は注意深くソースコード全体を読んで修正すべき箇所がどこにあるかを探し出す必要があり，学習者に求める演習問題への取り組み方が大きく異なる．また難易度に応じて選択式/空欄補充/誤り訂正問題を自動生成する文献 7) は，問題作成者があらかじめ入力した誤りの内容に置き換えることで誤り訂正問題を生成する手法を用いており，誤りの内容自体も自動生成する本研究とはアプローチが異なる．

3. 誤り自動挿入の手法

本章では，本研究で提案する，演習に適した誤りが含まれたソースコードを自動生成する手法について述べる．提案手法では，与えられたソースコードをもとに，(1) ソースコード中の誤り挿入対象の決定，(2) 挿入位置に適する誤りへの置換という手順で，誤りを含んだソースコードの生成を実現する．以下，各手順について詳しく説明する．

なお本論文では，C 言語のサブセットで記述されたソースコードを対象として説明および評価を行う．

3.1 誤り挿入対象の特定

どのような誤りが演習に適しているかはアルゴリズムごとに異なるため，ソースコードで利用されているアルゴリズムにそって誤り挿入対象を決定する必要がある．ここで誤り挿入対象とは誤り挿入によるソースコード中の変化する箇所のことであり，ソースコード中の変数，文や式だけでなく，制御構文など複数の式の集合も含む．しかし，様々なアルゴリズムのそれぞれに対して演習に適した誤りを検討するアプローチは汎用性に欠ける．

そこで本研究では，個々のアルゴリズムがベースとする**アルゴリズム設計パラダイム**（以下パラダイム）の情報をパラメータとして与え，これに対応した誤り挿入対象を決定するアプローチを考えた．パラダイムとは，アルゴリズムの設計時に用いられる枠組みであり，アルゴリズムの多くは何らかのパラダイムを利用して設計されている．つまり，多くのアルゴリズムに共通して使われるパラダイムについて誤り挿入対象の決定方法を定義しておけば，より汎用性の高い演習問題の自動生成が可能になると考えた．

提案手法では，アルゴリズムの講義において扱われることの多い[8] 分割統治法，再帰，動的計画法，貪欲法の 4 つのパラダイムについて，パラダイムの特徴を分析し，誤り挿入対象の決定法を検討した．また，いずれにも該当しないパラダイムについても誤り挿入対象の決定法を検討した．以下，それぞれのパラダイムにおける誤り挿入対象決定法の詳細について述べる．

パラダイム 1：分割統治法

分割統治法は，問題を小問題に分割して，各小問題の解を再帰的に求め，それらを統合することで最終的に問題を解こうとするパラダイムであり，全体の枠組みは次のようになる．

- **［計算］**問題の規模が小さい場合は直接解く．
- 問題の規模が大きい場合は
  - **［分割］**問題をいくつかの小問題に分割する．
  - **［再帰］**各小問題を再帰的に解く．
  - **［統合］**各小問題の解をもとに解を求める．

```
 1  #define N 10
 2  int a[N] = { RANDOM N };
 3   /* each element is a random integer from 0 to N */
 4
 5  void merge(int l[], int r[], int a[],
 6        int llen, int rlen, int len){
 7  /* snip  */
 8  /* merge two sorted arrays l[llen] and r[rlen] */
 9  /*  to make an array a[len] (len = llen+rlen) */
10  }
11
12  void mergeSort(int a[], int len){
13    int i;
14    int mid;
15    int left[len], right[len];
16    if(len > 1){
17      mid = len / 2;
18      for(i=0;i<mid;i++) left[i]=a[i];
19      for(i=0;i<len-mid;i++) right[i]=a[mid+i];
20      mergeSort(left, mid);
21      mergeSort(right, len-mid);
22      merge(left,right,a,mid,len-mid,len);
23    }
24  }
25
26  int main(){
27    int i;
28    mergeSort(a, N);
29  }
```


図 1　マージソートの例
Fig. 1　Example: mergesort.

以上の枠組みに対応する構文がアルゴリズム上重要な箇所であり，ソースコードにおける，(a) 再帰呼び出しを含む文や式，(b) 再帰呼び出しの引数に使われている変数の値を更新する文，(c) 再帰呼び出しより後に実行される関数呼び出し文や return 文，(d) (a) の実行の有無を決定する分岐命令およびその条件式を誤り挿入対象の候補とする．(a) は「分割」に，(b) は「分割」「再帰」，(c) は「統合」，(d) は「計算」にそれぞれ該当する．

分割統治法に基づくアルゴリズムの実装例として，マージソートのソースコードを**図 1** に示す．ここでは，再帰呼び出しを含む文・式（図 1 の A），再帰呼び出しの引数に用いられる変数（left，right，mid，len）について，これらの変数を更新する文（図 1 の B），再帰呼び出しの後に実行される関数呼び出し文（図 1 の C），再帰呼び出し文の実行の有無を決定する分岐命令（図 1 の D）を，誤り挿入対象の候補とする．

パラダイム 2：再帰

再帰を用いたアルゴリズムでは，再帰呼び出し文および再帰呼び出し文を実行するかを決める分岐命令がアルゴリズム上重要な場所であるので，ソースコードにおける，(a) 再帰呼

び出しを含む文や式，(b) 再帰呼び出しの引数に使われている変数の値を更新する文，(c) (a) の実行の有無を決定する分岐命令およびその条件式を誤り挿入対象の候補とする．

パラダイム 3：動的計画法

動的計画法は分割統治法と同様に問題を小問題に分割して解くパラダイムである．ただし，分割統治法と異なり，小問題の解を記録する表を用意して，上位の問題を解くときにはその表を参照することで計算量を減らす．

動的計画法では，表の役割をする変数（以下表変数）が重要な役割を持つ．つまり，表変数を更新する文に影響する処理がアルゴリズム上重要な場所である．ソースコード中で表変数は通常配列で表現され，また反復処理を行うための初期化処理を行うことが多い．そこでソースコード中で最初に現れる，配列へ定数を代入する文を検出し，その配列を表変数と判断する．そのうえで，(a) 表変数への代入文，(b) (a) の文中に現れる変数の値を更新する文，(c) (a) の実行の有無を決定する分岐命令およびその条件式を誤り挿入対象の候補とする．

パラダイム 4：貪欲法

貪欲法は局所的に最良の選択を繰り返すことによって最終的に大域的な解が得られるという発想に基づくパラダイムである．貪欲法の場合は，局所解の導出に該当する処理がアルゴリズム上重要な場所である．

そこで，ソースコードにおける繰返し文（while 文，for 文）に着目し，(a) 繰返し文中で，分岐命令によって実行の有無が決定される代入文，(b) (a) の右辺に現れる変数の値を更新する文，(c) (a) の実行の有無を決定する分岐命令およびその条件式を誤り挿入対象の候補とする．

パラダイム 5：その他

アルゴリズムの中には，特定のパラダイムに基づいて設計されていないものや，今回は考慮していないパラダイムに基づき設計されたものが存在する．その場合はアルゴリズムの全般に共通する動作決定の要素を，アルゴリズム上重要な場所であるとする．

具体的には，分岐，繰返し，関数呼び出しの処理に注目し，ソースコードにおいてこれらに該当する条件式や分岐命令文を，誤り挿入対象の候補とする．ただし for 文は，2.2 節の性質 (2) を満たさない誤りを生成しやすいことが経験的に判明したため，誤り挿入対象の候補とはしない．

3.2　誤りパターン挿入

誤り挿入対象の候補をすべて抽出した後は，あらかじめ与えられた個数の候補をランダムに選択し，その位置に実際に誤りを挿入する．具体的には，特定の構文パターンごとに対応する**誤りパターン**をあらかじめ定めておく．そして誤り挿入対象に存在する構文形式を確認し，その構文に対応する誤りパターンを適用する．構文と誤りパターンの対応表の主な項目を**表 1** に示す．適用可能な誤りパターンが複数ある場合は，いずれか 1 つをランダムに選択する．

表 1　誤り挿入対象と誤りパターンの対応表
Table 1　Rules for syntax-directed fault pattern injection.

誤り挿入対象の構文	挿入可能な誤りパターン
定数 c	定数 0，$c+1$，$c-1$
純変数	`+1` または `-1` の追加; 別変数への置換
算術演算子 `+`，`-`	`+`→`-`，`-`→`+`
論理演算子	別の論理演算子へ変換
算術二項演算 (e.g.，`A + B`)	一方の項を削除 (e.g.，`A`)
論理二項演算 (e.g.，`A && B`)	一方の項を削除 (e.g.，`B`)
関数呼び出し文	文を削除
制御構文 (e.g.，`if (S) A else B`)	条件式を削除 (e.g.，`A`)

例として，誤り挿入対象が代入文 “`i=j+1`” であった場合，含まれる構文は “変数 `i`”，“変数 `j`”，“算術演算 `j+1`”，“算術演算子`+`”，“定数 `1`” の 5 つであり，そのうち 1 つが誤りパターン挿入の対象となる．ここで算術演算 `j+1` が選択されたとすると，適用可能な誤りパターンは “一方の項を削除”，つまり “`j`”，“`1`” いずれかへの置換となる．

ただし，表 1 に従う変換を行うと，演習に適した誤りとならない場合が一部存在する．そこで，誤り適用後の構文をチェックするための制約ルールを設け，制約ルールを満たす誤りだけを実際に適用するという方法をとる．以下，制約条件の概要を述べる．

3.2.1　構文エラー

誤りパターンとして “別変数への置換” を適用すると，未定義変数の参照や互換性のないデータ型の変数への代入など，結果的に構文エラーに相当する誤りを生成する場合がある．これは 2.2 節に示した演習に適した誤りの定義に反するので，任意の 2 変数 `i`，`j` において，(1) `i` と `j` が同一スコープ内に存在，(2) `i`，`j` のデータ型が同一（配列変数であれば次元数と要素数も同一），かつ (3) 代入 `i = j` または `i` と `j` の比較演算が存在，の 3 つの条件がすべて満たされる場合に限り，`i`→`j` の変数置換を認める．これにより，構文エラーの可能性がある誤りは自動生成しないようにする．

3.2.2　不正または不自然な式

誤り挿入対象の文法が “定数” の場合の誤りパターンは “定数値の変換” となるが，分母の定数の場合，`i/2` → `i/0` のように式の分母が 0 となる誤りを生成する場合がある．しかしこれは単なる数学上の誤りにすぎず，アルゴリズムとの矛盾とは無関係であるため，変換候補から除外する．同様に，`i*1`，`i+0` などの無意味な演算式はアルゴリズムに関係なく誤り箇所の発見が容易となるため，これらも変換候補から除外する．

4. 実　　装

3 章で提案した手法を実現するシステムの試作版を，**図 2** の構成で実装した．本試作システムは Windows XP（SP2）上で動作する．字句・構文解析は，flex 2.5.4，bison 2.1 を用いて実装した．C コンパイラとして gcc 3.4.4，誤り挿入対象検索および誤りパターン適用は Scheme で実装し，処理系には SCM 5e2 を利用した．プログラムの規模は字句・構文解析部が約 1,000 行，誤り挿入対象検索・誤りパターン適用部が約 3,600 行である．

以下，実装システムの動作手順について説明する．

利用者は，1) ソースコード，2) 実装アルゴリズムのパラダイム，3)1 つのソースコードに挿入する誤りの個数，4) 生成する演習問題の数をパラメータとしてシステムに与える．

システムはまず入力されたソースコードを字句・構文解析し，出力として構文木の構造をとる中間コード（**図 3**）を生成する．実装における中間コードは，Scheme で解析可能なリスト構造（S 式）で表現する．

次に生成した中間コードを走査し，誤り挿入対象を検出する．1 回目の走査で，パラダイムに対応する構文，誤り挿入対象決定に関連する変数や周辺構文を記録しておく．その後 2 回目の走査で，変数の依存関係や構文の位置関係をチェックし，誤り挿入可能な位置を特定する．誤り挿入可能と特定された構文には，そのノードの親ノードとして，誤り挿入対象を示すリスト（MIS リスト）を挿入する（**図 4** 右上）．

2 回目の走査が完了したら，挿入した MIS リスト群から，指定された挿入誤り個数だけ MIS リストをランダムに選択する．選択された MIS リストの子孫ノードが，実際に誤りを挿入する構文となる．

選択した各 MIS リストの子孫ノードについてそれぞれ構文をチェックし，表 1 に従い対応する誤りパターンでリストを置換する（図 4 下）．1 つの誤り挿入対象に対して複数の誤りパターンが適用可能なときは，ランダムに 1 つ誤りパターンを選択する．また置換後の構文については制約ルールのチェックを行い，ルールを満たさない場合は構文を置換前の状態に戻し，別の誤りパターン適用を試みる．

誤り挿入が完了したら，最後に中間ノードをソースコードに逆変換して出力する．以上の誤り挿入対象決定と誤りパターン適用の処理を，生成する演習問題数分繰り返し行う．

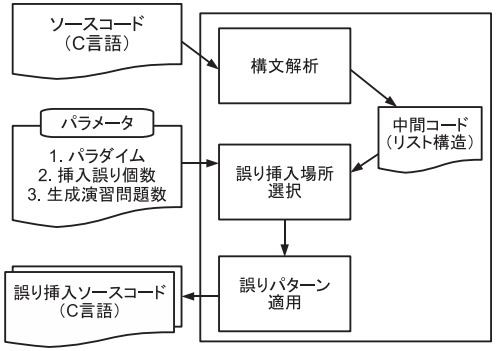

図 2　試作システムの構成
Fig. 2　System overview.

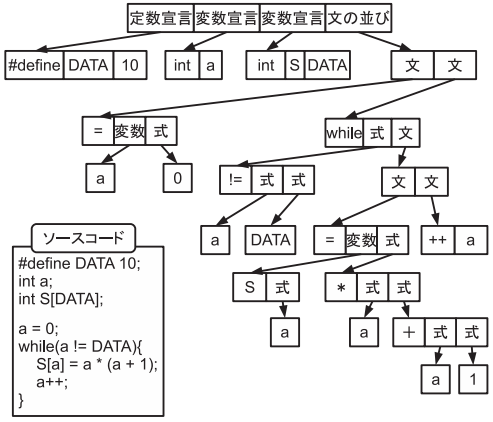

図 3　中間ノード（構文木）の例
Fig. 3　An example of syntax tree.

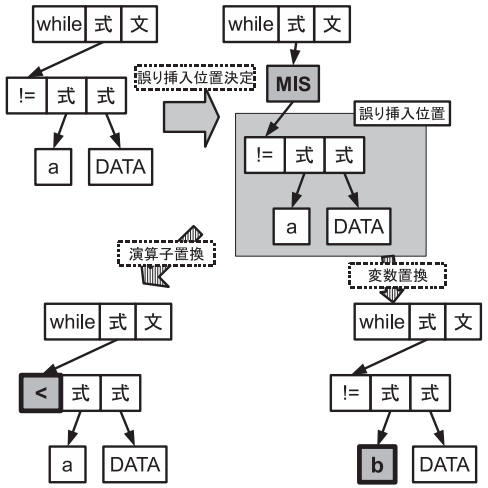

図 4　MIS ノード挿入と誤りパターン適用
Fig. 4　Inserting MIS node / Injecting faults in syntax tree.

5. 間違い探し演習の実践

提案手法の評価に先立ち，間違い探し演習に対する学習者の反応を確かめるため，大阪大学基礎工学部情報科学科の 2 年次後期必修科目「データ構造とアルゴリズム」において間違い探し演習の評価実験を実施した．受講者は 1 年次および 2 年次前期に Pascal および C 言語のプログラミングの経験がある．演習時間は 1 回 15 分程度で，その時間中に演習問題（Q1–Q3）および演習に関するアンケート（Q4，Q5）への回答を実施するという形式で計 4 回（Exp.1–4）の評価実験を実施した．また，各回とも挿入誤りの異なる問題を複数用意し，各学生にはランダムに選んだ 1 問を与えた．取り上げたアルゴリズムや回答者数などの各回の実施状況を**表 2** に示す．

なお，第 1，2，4 回は試作システムで演習問題の生成を行ったが，第 3 回のみ手作業で演習問題を作成した．これは，第 3 回で扱ったアルゴリズムが 3.1 節で述べたどのパラダイムも利用しておらず，for 文を処理の中心とするものであったため “その他” のパラダイムで適用可能な誤りパターンが存在しなかったことが原因である．このようなソースコードへの対応は今後の検討課題である．

表 2　講義での演習実施状況
Table 2　Statistics of exercises in class.

	Exp.1 Binary search	Exp.2 Selection sort	Exp.3 Shell sort	Exp.4 Quick sort
Q1，Q2，Q3 解答者数	72	73	69	67
Q4，Q5 回答者数	58	66	49	44
問題数	6	3	3	3

学籍番号　　名前

次に示すソースコードは「**選択ソート(昇順)**」を実装したものであるが、1ヶ所だけ間違いが含まれています。

```
1   for(k = n-1; k > 0; k--){
2     i = 0;
3     m=k;
4     while (i < k){
5       if (data[i] < data[m])
6         m = i;
7       i = i + 1;
8     }
9     t = data[m];
10  data[m] = data[k];
11  data[k] = t;
12 }
```

(1) 間違いを修正して下さい。

(2) 間違いが含まれていることによって、正しいアルゴリズムと、結果や動作がどのように変わってしまうかについて考察してください。

(3) 選択ソートというアルゴリズムについて簡潔に説明してください。

図 5　演習問題の例
Fig. 5　An example of exercise.

演習では，与えられたソースコードに対して（Q1）誤りを発見し修正せよ，（Q2）元になっているアルゴリズムとの相違点について考察せよ，という問題を提示する．第 2 回以降は，提示されたアルゴリズムの要旨を理解しているかを把握するために（Q3）そのアルゴリズムについて簡潔に説明せよ，という問題を追加した．**図 5** に選択ソートを題材にした演習問題の例を示す．

またアンケートでは，演習がアルゴリズムの理解に役立ったかを問う選択肢式の質問（Q4）と，間違い探し演習について自由記述での回答を求める質問（Q5）を用意した．Q4 では以下に列挙する選択肢を設けた．選択肢 1，2 は演習によって理解が深まった，選択肢 3，4 は演習によって理解が深まらなかったと解釈できる．

表 3　Q4 の回答の人数分布
Table 3　Distribution of #learners for Q4.

	Exp.1	Exp.2	Exp.3	Exp.4
Choice 1	45	48	30	32
Choice 2	8	10	7	6
Choice 3	2	1	1	2
Choice 4	3	2	4	3
Choice 5	-	5	7	1

表 4　Q4 の回答結果ごとの正答率
Table 4　The percentage of questions answered correctly for each learner group by Q4.

	Choice for Q4				
	1 or 2				3 or 4
	Exp.1	Exp.2	Exp.3	Exp.4	Exp.1–4
回答者数	53	58	37	38	18
Q1	69.8%	93.1%	75.7%	80.0%	55.6%
Q2	67.9%	69.0%	54.1%	84.2%	27.8%
（部分正答含む）	84.9%	79.3%	64.9%	73.7%	44.4%
Q3	-	75.9%	67.6%	68.4%	53.8%
（部分正答含む）	-	96.6%	97.3%	100%	100%

Choice 1： より深く理解するのに役立った．
Choice 2： 演習前は誤解をしていたが解消できた．
Choice 3： 演習をしたために理解が曖昧になった．
Choice 4： 演習を通じても内容が理解できなかった．
Choice 5： その他（自由回答，第 2 回以降で追加）．

Q4 の集計結果を**表 3** に示す．各回とも 75%以上の学生が選択肢 1，2 のいずれかを回答しており，大半の学習者にとって間違い探し演習はアルゴリズムへの理解に効果があると考えていることが分かった．

また，理解が深まったと考えた（選択肢 1，2 の回答を行った）学習者と理解が深まったとは考えなかった（選択肢 3，4 の回答を行った）学習者の 2 つのグループに分けて，Q1–Q3 の正答率を集計したものを**表 4** に示す．選択肢 1，2 を選んだ学習者に対しては，第 1–4 回各々の場合について示しているが，選択肢 3，4 を選んだ学習者は数が少ないため，第 1–4 回の合計のみ記載している．Q2，Q3 では，完全に誤答ではないが説明不足である解答を‘部分正答’として集計した．

表 4 より，Q4 で選択肢 1，2 を回答した学習者について，Q2 の正答率が第 4 回を除いて 7 割を下回っており，逆に選択肢 3，4 の回答を行った学習者について，Q2 の正答率が 2 割以上という結果が得られた．つまり一部の学習者は，理解できた/できないという主観的な感覚と，自身の実際の理解度が合致していないことを意味している．本演習を実施する場合は，採点結果のフィードバックを行うことで，学習者の理解度に対する認識を確実にするプロセスが重要だといえる．

Q5 については，4 回の実施で計 67 件の意見が得られた．分類すると次のようになった．
(1)　アルゴリズムを理解しないと解けない（14 件）．
(2)　ソースコードを精読する契機となる（9 件）．
(3)　ゲーム性が高く面白い（10 件）．
(4)　「良いと思う」などの肯定的意見（8 件）．
(5)　「よく分からない」などの否定的意見（1 件）．
(6)　演習に対する学習者側からの助言（17 件）．
(7)　その他（8 件）．

(1) および (2) の意見が多いことから，アルゴリズムに対する理解を深めるという本演習の趣旨が学習者にも意識されていたことが分かる．また，(3) の意見は，間違い探しという演習形式が学習者の興味を引きつけたことを示しているといえる．

以上の結果から，間違い探し演習は学習者にはおおむね好評であり，かつ演習の目的を十分に満たしていることが確認できた．

6. 提案手法の評価

本章では提案手法の評価方法とその結果を述べる．

評価基準として，手作業で挿入する誤り（以下**手作業誤り**）と，提案手法で自動生成可能な誤り（以下**自動誤り**）を比較し，自動誤りに対する評価が手作業誤りに対する評価と同等以上であれば，提案手法は有効であると判断する．

6.1 評 価 方 法

評価に用いるソースコードは，文献 8) を参考にアルゴリズムの学習で扱う典型的なアルゴリズムを選択した．それぞれ，クイックソート（分割統治法），二分探索（再帰），ナップザック問題（動的計画法），部分和問題（動的計画法），ダイクストラの最短経路問題（貪欲法），再帰を使わない二分探索（その他），選択ソート（その他）の 7 種類のソースコードを用意した．

用意した各ソースコードに対し，まず試作システムを利用して誤りを 1 つ含む演習問題

を生成した．提案手法で生成可能な全パターンの誤りを適用し，7 種類のソースコードから合計 280 個の自動誤りのサンプルを得た．次に同じソースコードに対して，誤りを 1 つ挿入した演習問題を 3 つずつ手作業で作成した．本評価では，計算機科学を専門とする教員 3 人と博士後期課程の学生 3 人（以下**問題作成者**）が手作業誤りの作成を行い，126 個，複数人で重複したサンプルを除くと 95 個の手作業誤りのサンプルを得た．結果，自動誤りと手作業誤りを合わせて，重複するサンプルを除く合計 328 個の誤り挿入サンプルが得られた．

そのうえで，328 個の全誤りサンプルについて，間違い探し演習の問題として有用か否かの評価を行った．評価には，著者らの所属研究室の学生 12 人（情報科学研究科博士前期課程 2 年 4 人，1 年 4 人，情報科学科 4 年 4 人．以下**評価者**）の協力を得た．評価の際，各サンプルは評価者ごとにランダムな順番で提示し，また各サンプルが提案手法と手作業のいずれで作成したものであるかは評価者には知らせなかった．

評価者による有用性判定の完了後，各サンプルの演習問題としての有用性を，有用と判断した人数の全 12 人に対する割合（以下**支持率**）として表した．なお，演習問題としての有用性は 2.2 節で定義した基準によることとしたが，実際には ‘有用/有用でない’ の境界の基準は評価者ごとの個人差が大きい．そのため本評価では，過半数の評価者から支持される誤りサンプルを，演習問題として有用であると考える．

6.2 評価結果

以下，提案手法が演習問題として有用な誤りをどの程度生成できているかについて評価する．**表 5** は自動誤りの各サンプルについて，有用性の支持率別に 4 グループに分けてカウントした結果である．全自動誤りサンプルの 73.2%（= 35.7% + 37.5%）が 50%を超える評価者から有用性を評価された．

一方，手作業誤りにおける同様の支持率分布では，サンプルの 82.1%（= 57.9% + 24.2%）が 50%を超える評価者から有用性を評価されており，自動誤りは手作業誤りに比べると若干支持率が低い結果となっている．有用性の低い誤りの自動生成を抑えるために，提案手法を改良する必要があり，次節で議論する．

6.3 提案手法の改善

6.2 節の結果をふまえ，提案手法の改善方法について検討する．

支持率が 50%以下の自動誤りサンプル 75 個について，適用した誤りパターンの種類別にサンプル個数を集計した結果を**表 6** に示す．総数が 2 個以下の誤りパターンは「その他」でまとめている．表 6 から，特定の誤りパターンに支持率の低い誤りが固まっていることが確認できる．

表 5　自動誤りサンプルの有用性支持率
Table 5　Precision rate.

	有用性の支持率 (%)			
	75%超過	75%以下 50%超過	50%以下 25%超過	25%以下
クイックソート	20	21	10	4
二分探索（再帰）	15	23	15	8
二分探索（反復）	13	26	13	3
ナップザック問題	14	10	1	0
部分和問題	10	3	3	0
最短経路問題	20	12	10	0
選択ソート	8	10	7	1
合計	100 (35.7%)	105 (37.5%)	59 (21.1%)	16 (5.7%)
手作業誤り	55 (57.9%)	23 (24.2%)	15 (15.8%)	2 (2.1%)

表 6　有用評価が 50%以下の誤りパターン
Table 6　Fault-patterns with low-rate evaluation.

誤りパターン	総数
変数 → 変数 +1	17
変数 → 別の変数	12
変数 → 変数 −1	12
定数 1→2	9
論理演算子置換	6
配列添字の変数置換	4
算術演算子置換	3
その他	12

また各サンプルを検証した結果，生成を回避できる誤りパターンが 31 個存在することが分かった．以下，対応可能な誤りパターンついて説明する．

“変数 → 変数 +1”，“変数 → 変数 −1” の誤りパターンのうち，二分探索において，検索値を格納する変数に対して加算，減算するパターンが合計 12 個存在した．しかし検索値はソース全体を通して不変であることから，不変の値を持つ変数の増減は “不自然な式” と判断された結果となった．そこで，ソースコード中に表れる変数のうち，(1) その変数に値を代入する代入文が 1 つだけ存在，(2) その代入文は，1 度しか実行されないことが保証されている，つまり変数が宣言されたスコープ内（グローバル変数の場合はソース全体）の制御構文（if，for，while）外で現れる，の 2 つを満たす変数は “格納値が不変の変数” と判断

し，変数+1，変数 −1 の誤りパターンを適用しないという制約ルールを追加することとした．これにより二分探索に限らず様々なアルゴリズムで，不変値を格納する変数を増減する誤りを生成しない対応が可能となる．

また変数置換を適用した誤りサンプルでは，連続する 2 つの代入文の左辺が同一になるサンプルが含まれていた（8 個）．結果，前の代入文が無意味な実行文となることから，有用性評価が総じて低い結果となった．この誤りパターンに対しては，「代入文の左辺の変数置換は，直前あるいは直後の代入文の左辺の構文と異なる形にする」の制約ルールを追加することで，誤りパターンの適用を避けることとした．

定数値に適用する誤りパターンのうち，1→2，2→3 の置換は，自動誤り 11 個と手作業誤り 1 個すべてのサンプルで，過半数の評価者の支持を得られなかった．評価者のコメントから，定数増減の誤りパターンはソースコード作成者の単純な入力間違いを想定させ，アルゴリズムを意識せずに誤りを修正できてしまう可能性が指摘された．よって，1→2，2→3 の誤りパターンは対応表から削除するのが適当であると判断した．

以上のアプローチを提案手法に適用した結果，支持率 50%超のサンプル数には変化がなかったが，支持率 50%以下のサンプル数が 31 個減少し，支持率分布は**表 7** に示す形に変化した．50%を超える評価者から有用性を評価された誤りが 82.3%（= 40.16% + 42.17%）となり，手作業誤りの支持率とほぼ同等となった．よって，本提案手法は手作業で演習問題を作成する場合と同程度に有用となったと考えられる．

6.4　提案手法の再現率評価

表 7 では，支持率が 75%を超えるサンプルの割合には，自動誤りと手作業誤りで若干の開きがある．そこで，現在の提案手法が，問題作成者が作成した手作業誤りをどの程度再現可能かを確認し，提案手法が現在カバーしきれていない誤り挿入手法を検討する．

まず手作業誤りについて，提案手法が同一の誤り挿入対象を決定できた割合（誤り対象再現率）と，誤り挿入対象が同一でかつ挿入する誤りも同一であった割合（挿入誤り再現率）を求めた（**表 8**）．その結果，誤り対象再現率がクイックソートを除いていずれも 70%以上であるのに対し，適用する誤りパターンも含めた挿入誤り再現率は，二分探索（反復法）の 71.4%を除いて 36.4–58.3%の範囲であった．

次に手作業誤りのサンプルを支持率別にグループ分けし，再現率の平均を求めた結果を**表 9** に示す．表 9 では支持率の高いサンプルほど再現率が高くなる傾向が確認できる．つまり提案手法は，有用性が低い手作業誤りを回避できていると考えられる．ただし，誤り対象は特定できながら挿入誤りを再現できなかった手作業誤りのサンプルは合計で 24 個にのぼっていることから，有用な誤りパターンの一部を提案手法は十分に網羅しきれていないと考えられる．

表 7　自動誤りサンプルの有用性支持率（改善版）
Table 7　Precision rate with improved fault-injection.

	有用性の支持率 (%)			
	75%超過	75%以下 50%超過	50%以下 25%超過	25%以下
クイックソート	20	21	7	0
二分探索（再帰）	15	23	9	3
二分探索（反復）	13	26	5	1
ナップザック問題	14	10	1	0
部分和問題	10	3	3	0
最短経路問題	20	12	10	0
選択ソート	8	10	5	0
合計	100 (40.16%)	105 (42.17%)	40 (16.06%)	4 (1.61%)
手作業誤り	55 (57.9%)	23 (24.2%)	15 (15.8%)	2 (2.1%)

表 8　再現率（アルゴリズム別）
Table 8　Recall rate.

	誤り対象 再現率	**挿入誤り 再現率**	**総数**
クイックソート	8 (50.0%)	6 (37.5%)	16
二分探索（再帰）	9 (75.0%)	7 (58.3%)	12
二分探索（反復）	13 (92.9%)	10 (71.4%)	14
ナップサック問題	11 (78.6%)	6 (42.9%)	14
部分和問題	8 (72.7%)	4 (36.4%)	11
最短経路問題	9 (75.0%)	7 (58.3%)	12
選択ソート	13 (81.3%)	7 (43.8%)	16
全体	71 (74.7%)	47 (49.5%)	95

6.5　再現率向上への検討

6.4 節の結果をふまえ，提案手法で再現できなかった支持率 50%超過の手作業誤り 34 個について，誤りパターンの傾向を調べた．該当サンプルが 2 個以上存在した誤りパターンについて，その個数を**表 10** に示す．

表 10 に列挙した誤りパターンについてサンプルを 1 つずつ検証した結果，構文ベースの誤り挿入が適用できるパターンが一部存在することが分かった．

表 9　支持率別の再現率
Table 9　Precision rate per recall pattern.

	誤り対象再現率	挿入誤り再現率	総数
$0 < x \leq 25\%$	0 (0%)	0 (0%)	2
$25 < x \leq 50\%$	5 (33.3%)	3 (20.0%)	15
$50 < x \leq 75\%$	15 (65.2%)	13 (56.5%)	23
$75 < x \leq 100\%$	51 (92.7%)	31 (56.4%)	55

表 10　支持率 50%以上の再現不可手作業誤りの誤りパターン
Table 10　Fault-patterns of unrecalled faults.

誤りパターン	個数
配列の添字式 [A-B]→[A]	5
変数置換（3.2.1 項定義外）	4
二項演算 → 変数置換	3
二項演算子 <→≤	2
配列 → 定数	2

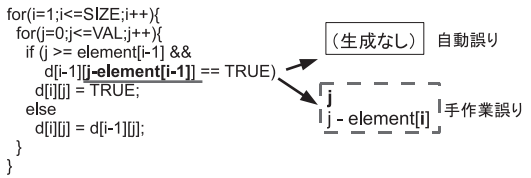

図 6　手作業誤りの例：配列引数 [A-B]→[A]
Fig. 6　Example of manual faults: Array index [A-B]→[A].

たとえば配列の添字への誤り挿入は，配列の要素数の範囲外へアクセス可能な誤りの生成を避けるため，提案手法では行わないこととしていた．しかし配列の添字が二項演算式 'A-B' の場合，片方の項を削除する誤り挿入（[A-B]→[A]）を適用した手作業誤り 5 個が高い支持率を得ていた．さらにそのうち 4 個は，置換後の添字表現 A がソース内の別の文で配列添字に使用されていることが分かった（**図 6**）．よって，配列の添字への誤り挿入は「ソース中に現れる同じ配列の添字式表現への置換」に限り可能とすることで，要素数の範囲外へのアクセスを起こさない構文ベースの置換が可能となる．ただしこの誤りパターンを適用すると，手作業誤りの再現とは別の新しい自動誤りサンプルも生成されうるため，追加で評価実験を行い，安定して有用な誤りを生成できるか確認する必要がある．

また，その他の誤りパターンについては，ソースコード中の変数がアルゴリズムにおいて

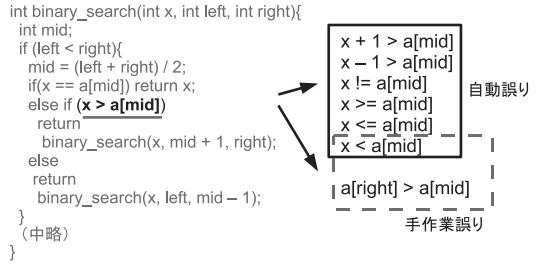

図 7　手作業誤りの例：純変数 → 配列変数
Fig. 7　Example of manual faults: simple variable → Array variable.

果たす意味的な役割（例：変数 mid は中間点という意味を持つ，など）を理解しないと挿入できない誤りが多数含まれていた（**図 7**）．提案手法はパラダイムをもとに構文ベースで誤りパターンを決定する手法のため，変数の意味理解も含めた誤り挿入への対応は別のアプローチが必要となると考えられる．これについては今後の課題である．

7. おわりに

本研究ではアルゴリズム学習を対象とし，ソースコードの誤り修正を行う演習を想定した演習課題の自動生成法を提案した．具体的には，アルゴリズム設計パラダイムを利用した誤り挿入対象の抽出手法，構文主導型のソースコード置換ルールによる誤り挿入方法を示した．これにより，入力されたソースコードに対して演習に適した誤りの自動挿入を実現した．

また提案手法について評価を行い，その結果をふまえ誤り挿入手法を改善することで，提案手法による演習問題作成が手作業で生成する場合と同程度に有用であることを確認した．

今後の課題としては，ソースコードの意味解析を含めた誤り挿入手法の改良があげられる．さらに発展的な目標として，自動生成した演習課題の難易度付けや解答の判定処理も自動化した，アルゴリズムの自主学習環境の構築を目指している．

謝辞　評価実験にご協力いただきました大阪大学大学院情報科学研究科増澤研究室の皆様に深く感謝いたします．

参考文献

1) The Joint Task Force for Computing Curricula: Computing curricula 2005 – The Overview Report, ACM (online). available from http://www.acm.org/education/

curric_vols/CC2005-March06Final.pdf (accessed 2007-11-16).
2) 斐品正照，徳岡健一，河村一樹：構造化チャートを用いたアルゴリズム学習支援システム，情報処理学会論文誌，Vol.45, No.10, pp.2454–2467 (2004).
3) 新村晃示，鈴木浩之，稲垣文雄，伊藤大輔，小西達裕，伊東幸宏：プログラミング言語学習を意識させないアルゴリズム作成・テスト環境の構築，教育システム情報学会研究報告，Vol.19, No.5, pp.63–70 (2005).
4) Carlisle, M., Wilson, T., Humphries, J. and Hadfield, S.M.: RAPTOR: A visual programming environment for teaching algorithmic problem solving, *Proc. 36th SIGCSE Technical Symposium on Computer Science Education*, pp.176–180 (2005).
5) Watts, T.: The SFC editor a graphical tool for algorithm development, *Journal of Computing Sciences in Colleges*, Vol.20, No.2, pp.73–85 (2004).
6) 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 (2001).
7) 菅沼　明，峯　恒憲，正代隆義：学生の理解度と問題の難易度を動的に評価する練習問題自動生成システム，情報処理学会論文誌，Vol.46, No.7, pp.1810–1818 (2005).
8) 浅野哲夫，和田幸一，増澤利光：アルゴリズム論，オーム社 (2003).

(平成 19 年 11 月 30 日受付)
(平成 20 年 7 月 1 日採録)

長瀧　寛之（学生会員）

2000 年大阪大学基礎工学部情報科学科卒業．2002 年同大学大学院基礎工学研究科博士前期課程修了．同年鳥取環境大学環境情報学部助手．2006 年大阪大学大学院情報科学研究科博士後期課程入学．教育工学の研究に従事．電子情報通信学会，教育システム情報学会各会員．

伊藤　亮太

2005 年大阪大学基礎工学部情報科学科退学．同年大阪大学大学院情報科学研究科博士前期課程入学．2007 年同修了．

大下　福仁（正会員）

2000 年大阪大学基礎工学部情報工学科退学．2002 年同大学大学院博士前期課程修了．2003 年同大学院情報科学研究科博士後期課程退学．同年大阪大学大学院情報科学研究科助手．現在，同大学院助教．並列アルゴリズム，分散アルゴリズムに関する研究に従事．ACM，IEEE，電子情報通信学会各会員．博士（情報科学）．

角川　裕次（正会員）

1990 年山口大学工学部電子工学科卒業．1992 年広島大学大学院工学研究科情報工学専攻博士課程前期修了．広島大学助手，講師，助教授を経て，現在大阪大学大学院情報科学研究科准教授．分散アルゴリズムと教育工学の研究に従事．IEEE Computer Society 会員．博士（工学）．

増澤　利光（正会員）

1982 年大阪大学基礎工学部情報工学科卒業．1987 年同大学大学院博士後期課程修了．同年同大学情報処理センター助手．同大学基礎工学部助教授を経て，1994 年奈良先端科学技術大学院大学情報科学研究科助教授．2000 年大阪大学大学院基礎工学研究科教授．2002 年大阪大学大学院情報科学研究科教授．1993 年コーネル大学客員準教授（文部省在外研究員）．分散アルゴリズム，並列アルゴリズムに関する研究に従事．ACM，IEEE，電子情報通信学会各会員．